Programming with Python

(DLMDSPWP01)

A Written Assignment on

Exploratory Data Analysis using Python

Author: Kaushik Puttaswamy

Matriculation Number: 321150196

Customer ID: 10549633

Tutor's Name: Cosmina Croitoru

Date: 31 July 2023

Place: Mysuru, Karnataka, India

Table of Contents

List of Figures

List of Tables

Table of Abbreviations

SQL	Structured Query Language
EDA	Exploratory Data Analysis
ORM	Object Relational Mapper
API	Application Programming Interface
LAV	Lower Adjacent Value
UAV	Upper Adjacent Value
IQR	Interquartile Range
SQRT	Square Root
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
RSE	Relative Squared Error
RRMSE	Relative Root Mean Squared Error
RMSLE	Root Mean Squared Logarithmic Error
NRMSE	Normalized Root Mean Squared Error
PC	Personal Computer

Chapter 1

Introduction

1.1 Overview

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). In regards to 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 behaviour of the components.

In this context of the study, we are correlating the given task, and the four training datasets can represent different scenarios or conditions under which the components were tested. Each training dataset, for example, could correspond to different operating parameters, environmental conditions, or variations in the manufacturing process. The test dataset is a separate set of data that is used to assess how well the chosen ideal functions generalise to new data. The test dataset can be made up of new measurements or observations that represent a different set of conditions not covered by the training datasets. The datasets for 50 ideal functions represent the expected behaviour or theoretical models of the components. Overall, the goal is to select the best-fit ideal functions that minimise deviations by comparing the training data to the ideal functions, and then use those functions to map and evaluate the test data.

1.2 Problem Definition

The task is to create a Python program that analyses the performance of electronic components by Using training data, select the four best-fitting functions from the fifty available. Furthermore, the program must use the provided test data to decide whether or not 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.

1.3 Aim and Objectives

The primary goal of this research is to create a reliable and effective Python program. Based on the training data, the program should select the four best-fit ideal functions from a set of 50 functions and then map the test data to these chosen functions while taking a deviation criterion into account.

The objectives are:

a) Data base creation:
   Developing a Python program that can independently construct a SQLite database (file), ideally using sqlalchemy, and load datasets.

b) Model Selection:

- Finding the sum of squared deviations between each ideal function and the training data.
- Selecting four ideal functions that minimize the sum of squared deviations and provide the best fit to the training data.

c) Mapping and Deviation Calculation:

- Determining if each x-y pair in the test dataset can be assigned to the ideal functions.
- Carrying out the mapping and calculating the deviation for each mapped value, making sure that the deviation does not exceed the specific criterion.

d) Data Visualization:

- Visualizing the train, test, and ideal data individually using scatter and box plots.
- Creating bar charts to display the results of the least-squares method for determining ideal functions for y1, y2, y3, and y4 train data.
- Plotting bar charts to determine the row with the highest absolute maximum deviation value between train and ideal for all rows of y values.
- Displaying a scatter plot of mapped x-y test data values with the corresponding chosen ideal function and absolute deviation.
- Plotting the predicted values (regression line), upper limits, lower limits, and mapped test data points for the selected best-fit ideal function.

e) Unit Testing:

- Developing unit tests to check the correctness and accuracy of the implemented function, method, or class.
- Ensuring that the functions responsible for computing deviations and selecting the best-fit functions work as expected.

1.4 Research Questions

The research questions framed for the given task are listed as follows:

a) Using train data, use the least squares method to obtain the four best-fit ideal functions (given criterion).
b) What are the four ideal functions based on the least squared error with respect to the train dataset?
c) Which is best alternative evaluation metrics as per the given criterion and why?
d) As per the second point in this research question, compare and check that the selected best-fit ideal function generated from the least squares approach is the same as the ideal function determined by the alternative evaluation metrics.

e) What is the R-squared value of each of the four ideal functions selected with the test dataset by matching the X values of the test data with the ideal data?
f) What is the deviation of the chosen ideal function after mapping it with the test dataset at each data point?

1.5 Scope and significance of study

The research focuses on implementing regression analysis techniques, particularly least-square regression, and applying them to train and test datasets. It also includes selecting the best squared deviation evaluation methods. The program needs to provide four ideal functions based on the chosen least square deviation from the train dataset and further visual representation of the data, as well as accurately map the test data to the chosen ideal functions based on the defined criteria. Furthermore, the designed code should be object-oriented, include at least one inheritance, standard- and user-defined exception handlings, unit tests for all useful elements, tables using sqlalchemy, visualization using bokeh, other relevant libraries, and documentation strings.

This study is significant because it has the potential to increase understanding of electronic component behaviour, improve quality control measures, support decision-making processes, enable predictive maintenance techniques, and drive further research and development efforts in the field. The findings of this study could have applications in a variety of industries that rely on electronic components, such as electronics manufacturing, telecommunications, automotive, aerospace, and others. Furthermore, it assists us in comprehending and implementing numerous assessment matrices in various real-world scenarios.

1.6 Structure of the Study

This research study is structured around three primary headings: introduction, investigation method, and conclusion. The introduction chapter is divided into several subsections, beginning with an overview, problem description, aim and objectives, research questions, and investigation technique section, which explore all viable solutions to the given task's aim and objectives. Finally, conclusions are reached, and the future area of study is described based on the evaluation of the results. Finally, git commands are included and executed in git bash in the additional task section. Furthermore, an appendix is provided to show the details of the Python program written on a Jupyter notebook using the Anaconda package, as well as several essential visualizations for the study.

Chapter 2

Investigative Method

2 Exploratory Data Analysis

Data scientists utilize exploratory data analysis (EDA) to analyze and study data sets and summarize their primary properties, frequently using data visualization approaches. It aids in determining how to best manipulate data sources to obtain the answers we require, making it easier for data scientists to discover patterns, identify anomalies, test hypotheses, and validate assumptions.

EDA primarily serves to investigate what data might reveal beyond formal modelling and to gain a better understanding of data set variables and their interactions. It can also aid in determining whether the statistical procedures under consideration for data analysis are adequate(*What Is Exploratory Data Analysis?*, n.d.).

In our project, we have csv files with three sets of data: "train," "test," and "ideal." The data is summarized and analyzed in order to compute correlations, discover irregularities, and perform additional checks.

2.1 Data storage and retrieval

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 licence.
- 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 tables

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 Â□“ Expression Language*, n.d.).

2.2 Database schema

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

Table 1: The training data's 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 functions' 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

2.3 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.

2.4 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 entirely (*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 points 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 frequently generated (*What Is Data Redundancy?*, n.d.). We address these concerns by using correlation analysis (also known as Pearson's product moment coefficient) between each pair of variables, which allows us to remove variables that have a high correlation with regard to other variables without losing any crucial dataset information.

2.5 Data visualization:

Data visualization refers to the graphical representation of information and data. Data visualization tools, which include visual components like charts, graphs, and maps, make it easy to view and comprehend trends, outliers, and patterns in data. Furthermore, it allows employees or business owners to deliver facts to non-technical audiences without causing confusion(*What Is Data Visualization?*, n.d.).

We can use numerous techniques, like scatter plots, line plots, bar graphs, heat maps, and so on, to perform pattern analysis on train, test, and ideal function datasets using data visualization. We can use these techniques to visualize the distribution, shape, and relationships between variables in our datasets.

2.5.1 Different visual elements and techniques used for visualization

a) Histogram: A histogram is a graph that depicts the frequency distribution of a few data points from a single variable. Histograms frequently divide data into "bins" or "range groups" and count the number of data points that belong to each of those bins(*What Is a Histogram Chart?*, n.d.).

b) Scatter Plots: A scatter chart, often known as a scatter plot, is a type of graph that depicts the relationship between two variables. They are an extremely powerful chart type, allowing readers to see a relationship or trend that would be impossible to see in virtually any other format(*What Is a Scatter Chart?*, n.d.)

c) Bar chart: A bar chart is a chart that displays categorized data using rectangular bars with lengths or heights proportionate to the values they represent. Bar charts can be either vertically or horizontally arranged(*What Is a Bar Chart?*, n.d.).

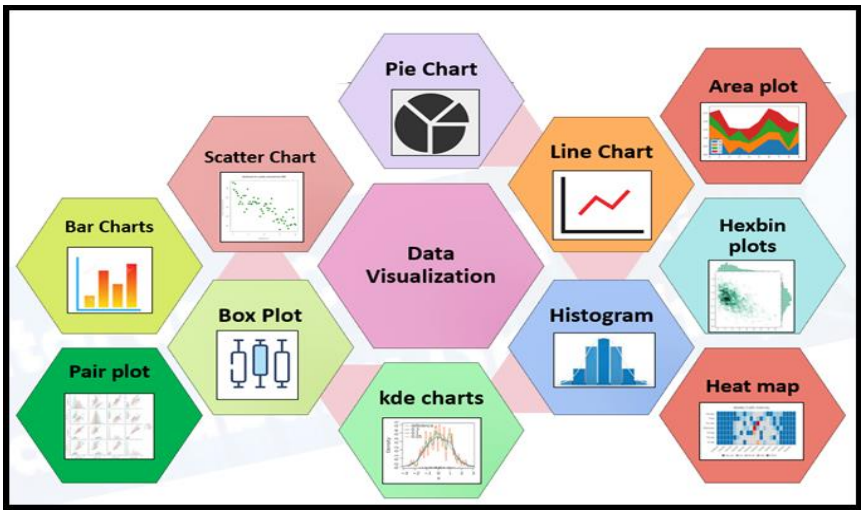
d) Pie Charts: A pie chart is a graphical representation that uses a circular graph to display data. It is a static composite graphic that works well with a few variables. Pie charts are frequently used to show single data points that relate to multiple categories. Each of these groups is represented by a "slice of the pie." The size of each slice is precisely proportionate to the number of data points in each category(*What Is a Pie Chart?*, n.d.).

e) Box Plot: A box plot is a data visualization technique that depicts data distribution. It's a great tool for displaying outliers. This approach, sometimes known as a box-and-whisker graph or plot, depicts relationships. By applying the five-number summary—minimum, first quartile (Q1), median, third quartile (Q3), and maximum—between a numerical y-variable and a grouping x-variable, A lower adjacent value (LAV), an upper adjacent value (UAV), and an interquartile range (IQR) are also shown in box-and-whisker plots. This method is quite good at displaying outliers, or spots that fall outside of LAV and UAV(*What Is a Box Plot?*, n.d.).

f) Heat Maps: Heatmap data visualization is a graphical approach of representing numerical data that employs colors to represent the value of each data point. The warm-to-cool color scheme is the most widely used in heatmap visualization, with warm colors representing high-value data points and cold colors representing low-value data points(Khan, 2020).

g) Combination chart: A combination chart combines data from several different categories across time. In general, two or three types of charts are merged to depict a link between data points. A bar chart and a line chart are commonly used together (*What Is a Combination Chart?*, n.d.).

Figure 1: Different techniques for visualization of data driven pattern

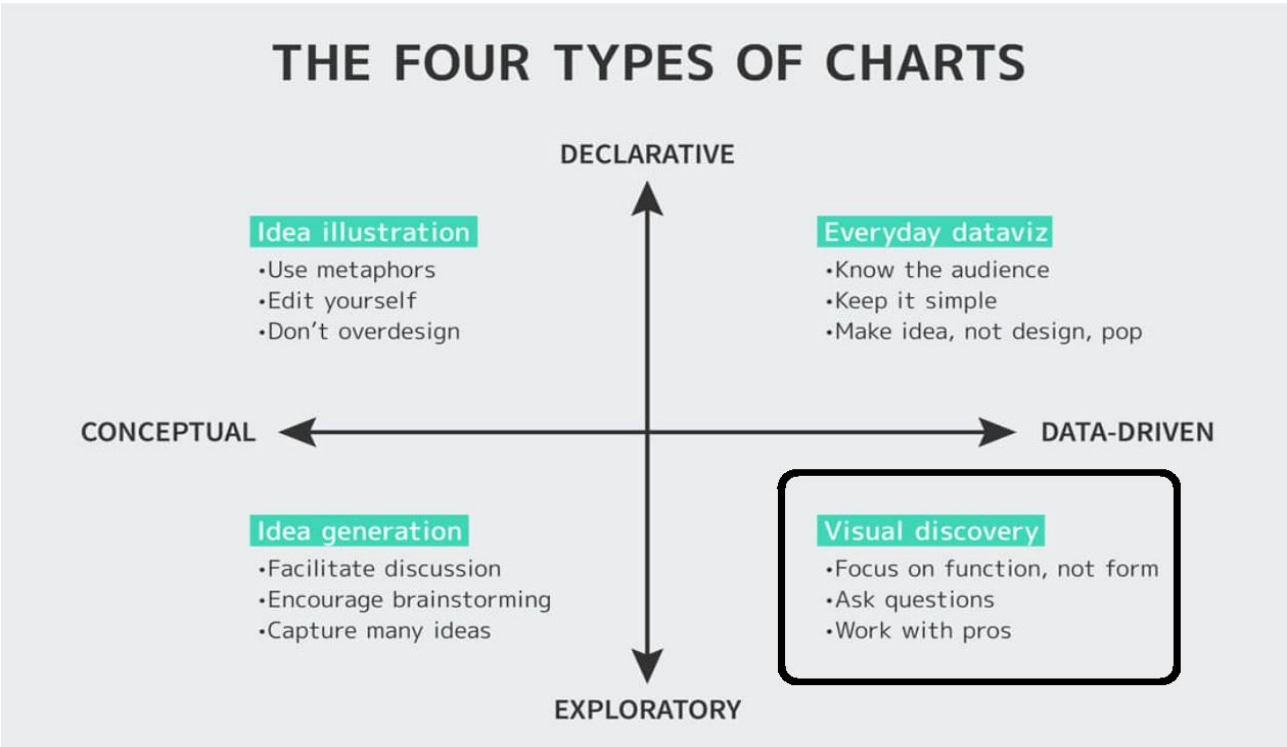


Source:(Pandian, 2021)

2.5.2 Types of visualizations (*Elements of Data Visualization - Sairam Tangirala*, n.d.):

There are four kinds of visualizations, according to Scott Berinato's book "Good Charts," released by Harvard Business Preview Press.

Figure 2: Four types of visualizations



Source:(*Elements of Data Visualization - Sairam Tangirala*, n.d.)

a) Everyday Dataviz: (1st quadrant)

- Standard charts and graphs are used to communicate an idea to an audience. Well-designed, based on a manageable quantity of data, and frequently used in presentations.
- Typically used by managers.

b) Idea Illustration: (2nd quadrant)

A visual representation of an idea that is not based on statistical facts metaphors such as trees, processes, and cycles are frequently used. For example, organizational charts, process diagrams, decision trees, etc. AkA consultant's corner. Declarative and conceptual representations simplify difficult topics by relying on people's ability to understand metaphors.

c) Idea Generation: (3rd quadrant)

- Concepts were sketched quickly for visualizing ideas unrelated to statistical data. Usually done in groups. For example, brainstorming sessions and whiteboard work.
- Happens on a white board in informal settings.

d) Visual Discovery: (4th quadrant)

- Data is used to confirm hypotheses or discover patterns and trends. Visual confirmation Visual exploration.
- Used by data scientists.

According to the task description, visual discovery (data-driven and exploratory) visualization is the best fit for the available datasets. Furthermore, exploratory data visualization is used to investigate and comprehend the data. It entails developing data visualizations without any predetermined preconceptions. The idea is to find patterns, trends, and relationships that may not be obvious from the raw data. Exploratory data visualization frequently entails developing a variety of visualizations and thoroughly reviewing them to uncover interesting characteristics and relationships.

2.6 Proposed frameworks

In this study, we must first find the best-fit four-ideal function that minimizes the sum of all y-deviations squared using train data by calculating the minimum sum of all y-deviations squared between ideal and train data, train data of four different y-columns against corresponding fifty y-columns in ideal data, with x values as key identifiers for both data using the least squares method (Square error/sum of squares).

Second, to confirm if the best-fit ideal function selected from the least squares approach is correct or not, I utilised one of the most appropriate error methods from MSE, RMSE, RMSLE, RRMSE, and NRMSE. To choose the best alternative assessment metric among MSE, RMSE, RSE, RMSLE, RRMSE, and NRMSE. Let us first define the frequently used measurement used to evaluate the quality of fit in terms of the distance between the ideal function and the actual training points. The

MSE and RMSE are two often-used members of the squared error method family. The remaining RSE, RRMSE, RRMSLE, and NRMSE give relative measurements by normalising the error with regard to the target variable's range or standard deviation. In our study, any of the two approaches (MSE and RMSE) can be employed because the measurement error for both is well within an acceptable range and very close to zero; thus, I chose the MSE method as an alternative evaluation metric.

Furthermore, It is important calculate the R-squared value between the "ideal" dataset's target variable (Y) and the target variable (Y) in the test dataset where ideal data containing 400 data points on the other hand test data containing 90 data points after removing duplicate data points such that calculating R-squared value by making even size data set(90 data points) by matching X values of test data with ideal data in other words by taking X values as key identifiers between the ideal and test datasets because to know the R-squared value of each selected 4 ideal function with the test dataset that represent the measure proportion of the variance in the dependent variable that can be explained by the independent variable(s) in a regression model(*R-Squared*, n.d.).

Finally, we must use the provided test data (B) to determine whether or not each x-y pair of values can be assigned to the four chosen ideal functions. The criteria for mapping the specific test case to the four ideal functions is that the calculated regression's existing absolute maximum deviation does not exceed the largest deviation between the training dataset and the ideal function chosen for it by more than factor sqrt(2).To proceed with this criterion, we first have to find the largest absolute deviation by calculating the corresponding deviation between the chosen ideal function and the train data, which is multiplied by the factor sqrt(2) for four selected ideal functions, and then we must calculate the predicted values for four selected ideal functions using regression.

Furthermore, we must compute the absolute deviation between the y column test data that has been sorted and cleaned by removing duplicate values and the predicted y column four ideal function, where ideal data is finally mapped to test data. As a given condition for this study, the estimated absolute deviation does not exceed the maximum absolute difference between the chosen ideal function and the train data, which is multiplied by the factor sqrt(2).

2.6.1 Different squared evaluation metrics for regression model

a) Mean Squared Error (MSE): The mean squared error, also known as L2 loss, is calculated by squaring the difference between the predicted and actual values and averaging it throughout the dataset (M, 2021).

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

b) Root Mean Squared Error (RMSE): RMSE is also known as the Root Mean Square Deviation. It calculates the average magnitude of the errors and is concerned with deviations from the actual value (M, 2021).

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

c) Relative Squared Error (RSE): To compute Relative Squared Error, take the Mean Squared Error (MSE) and divide it by the square of the difference between the actual and mean values (M, 2021).

$$\text{RSE} = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

$$\text{where} \quad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

d) Relative Root Mean Squared Error (RRMSE): RRMSE is a dimensionless variant of RMSE. The relative root mean square error (RRMSE) is the root mean squared error normalized by the root mean square value, where each residual is scaled against the actual value(M, 2021).

$$\text{RRMSE} = \sqrt{\frac{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(\hat{y}_i)^2}}$$

e) Root Mean Squared Logarithmic Error (RMSLE): The Root Mean Squared Logarithmic Error is calculated by applying log to the actual and predicted numbers and then subtracting their differences. RMSLE is robust to outliers because small and big errors are treated equally(M, 2021).

$$\text{RMSLE} = \sqrt{(log(y_i + 1) - log(\hat{y}_i + 1))^2}$$

f) Normalized Root Mean Squared Error (NRMSE): The Normalized RMSE is commonly derived by dividing a scalar quantity. It can be done in a variety of ways, such as (M, 2021),

- RMSE / maximum value in the series
- RMSE / mean
- RMSE / difference between the maximum and the minimum values (if mean is zero)
- RMSE / standard deviation
- RMSE / interquartile range

2.7 Data Pre-Processing

Data preparation is required prior to 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, human errors, and is incomplete. Data preparation resolves such difficulties and makes datasets more complete and 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.

2.7.1 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.

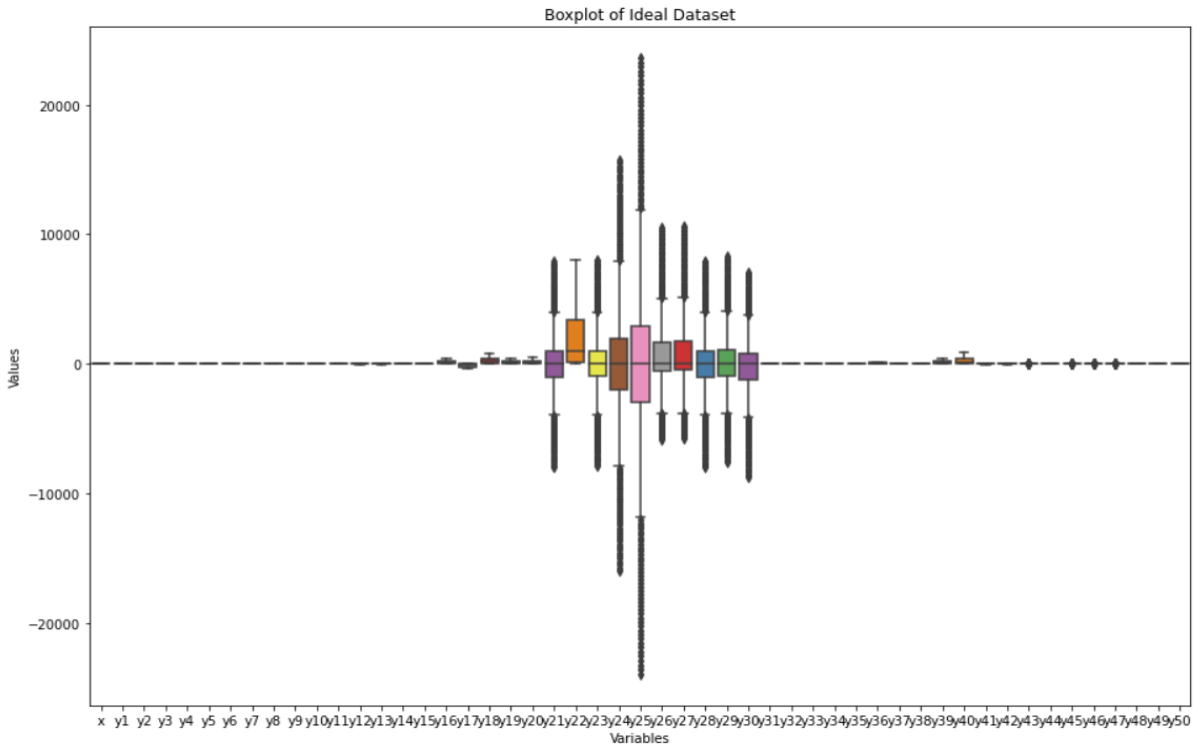


Figure 3: Boxplot of Ideal Dataset

2.7.2 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:

	count	mean	std	min	25%	50%	75%	max
x	400.0	-0.050000	11.561430	-20.000000	-10.025000	-0.050000	9.925000	19.900000
y1	400.0	0.072680	0.696046	-1.291954	-0.519309	0.118563	0.648439	1.333112
y2	400.0	-0.003936	0.778610	-1.485138	-0.680415	0.009687	0.628954	1.496933
y3	400.0	0.490464	0.838694	-1.466599	-0.123099	0.766163	1.160746	1.741839
y4	400.0	0.078216	1.441522	-2.469857	-1.323802	0.134146	1.467942	2.471263

Table 4: Train data description of the dependent and independent variables

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.

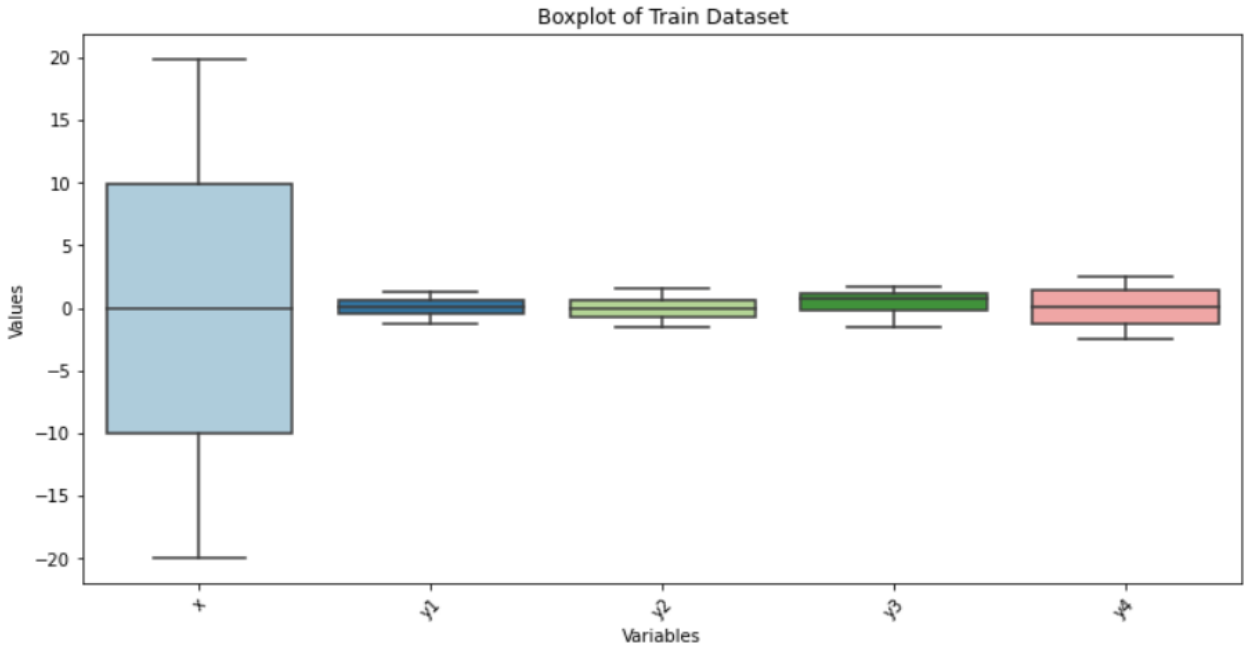


Figure 4: Boxplot of Train Dataset

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 maximum values are within the whiskers. Furthermore, a scatter plot for the y1, y2, y3, and y4 train data columns with the regression line is presented below.

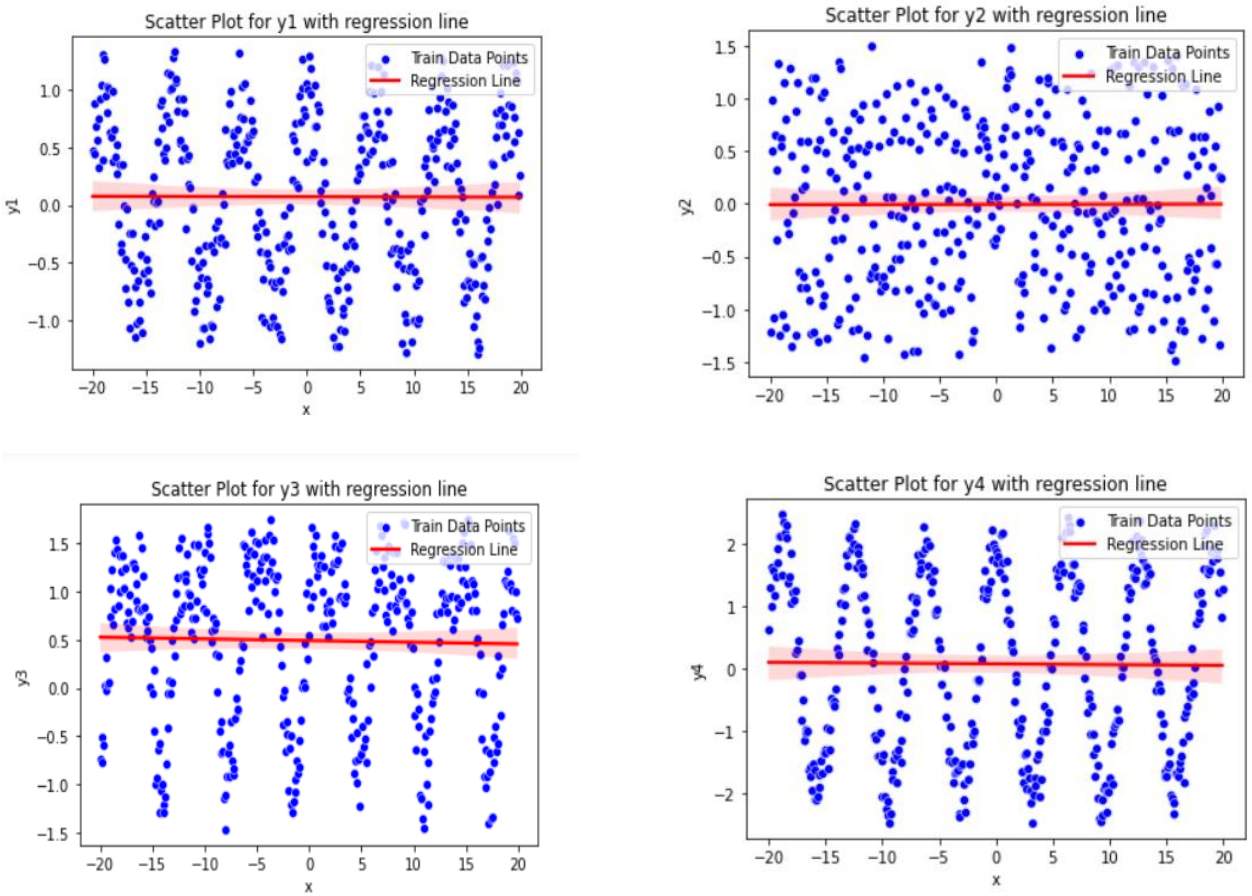


Figure 5: Scatter plot of the train dataset displayed with regression line

2.7.3 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:

	count	mean	std	min	25%	50%	75%	\
x	100.0	2.109000	11.385327	-19.500000	-7.950000	4.800000	11.700000	
y	100.0	0.174205	1.256865	-2.330886	-0.826261	0.220041	1.262826	

	max
x	19.800000
y	2.594559

Table 5: Test data description of the dependent and independent variables

We observed ten duplicate values in the 'x' variable of the test function, and it is vital that we treat them correctly in order to protect the quality and integrity of our data. A frequent strategy in data preprocessing 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.

	count	mean	std	min	25%	50%	75%	max
x	90.0	1.845556	11.732344	-19.500000	-9.000000	4.30000	11.900000	19.800000
y	90.0	0.211670	1.218689	-2.330886	-0.675988	0.23836	1.226106	2.594559

Table 6: Test data description of the dependent and independent variables after removing 10 duplicates value

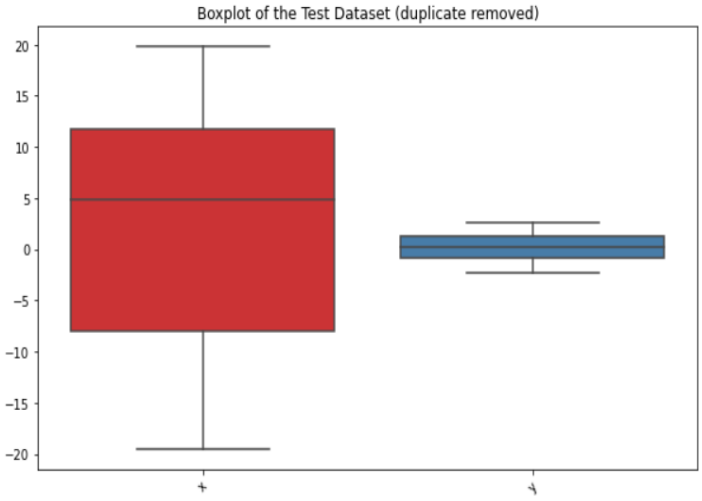


Figure 6: Boxplot of Test Dataset after removing duplicates values

The following conclusions can be drawn from the above-mentioned box plot: The box plots for test data do not show points outside of the whiskers, indicating that their lowest and highest values are contained within the whiskers.

2.8 Find Ideal Function Using Train Dataset

To meet the project's criteria of identifying ideal functions using the least squares approach, we have to understand what the least squares method is so that we can determine the best fit of four ideal functions using the least squares method by using each y-variable in the train dataset.

2.8.1 Least Squares analysis

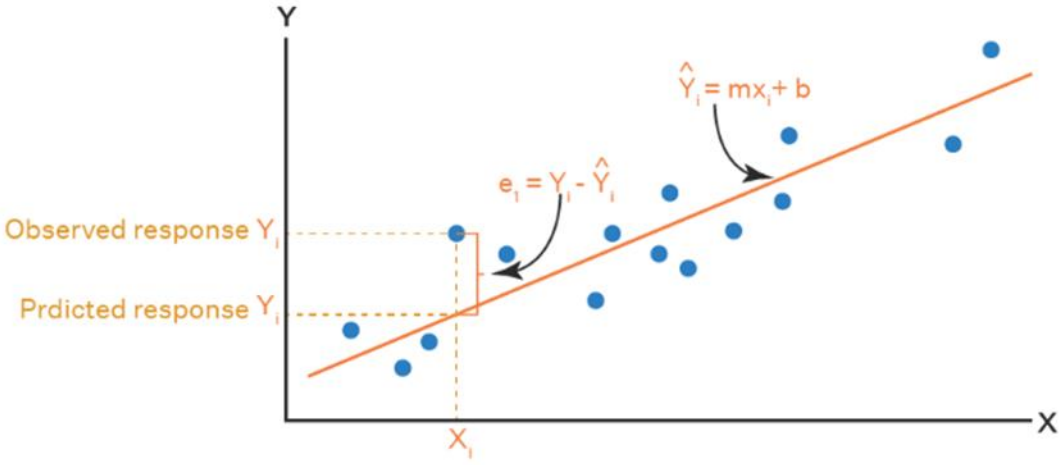
The least square method is defined as a method for minimizing the sum of squares of all deviations or errors that emerge from each equation(*Least Square Method*, n.d.). In our scenario, each column of the ideal dataset. The equation is as follows:

$$\text{Least square} = \sum (y_i - \hat{y}_i)^2$$

The least squares method is a typical methodology used in regression analysis to discover an approximation of the solution. Some advantages of this method include the fact that it is less affected by extreme values or data errors because it works by minimizing the sum of the squared differences between the predicted and actual values (MALI, 2021). Furthermore, the least-squares method

provides the most accurate relationship among the variables. The difference between the sums of squares of residuals and the line of best fit is negligible using this method(Srivastav, 2019). Furthermore, the straight line in the graph below depicts the potential relationship between the independent variable and the dependent variable. The ultimate purpose of this method is to minimize the difference between the observed and predicted responses using the regression line. A lower residual value indicates that the model fits better. The data points must be minimized by lowering the residuals of each point from the line(*Least Square Method - Formula, Definition, Examples*, n.d.).

Figure 7: Example of Least Square Method Graph



Source:(*Least Square Method - Formula, Definition, Examples*, n.d.)

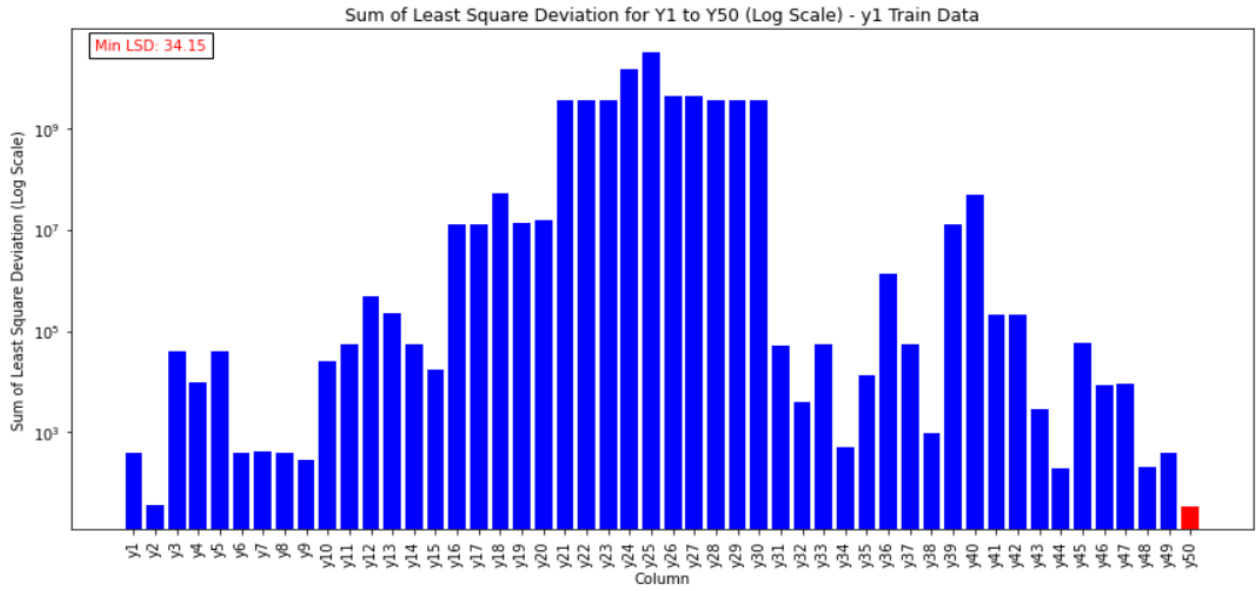
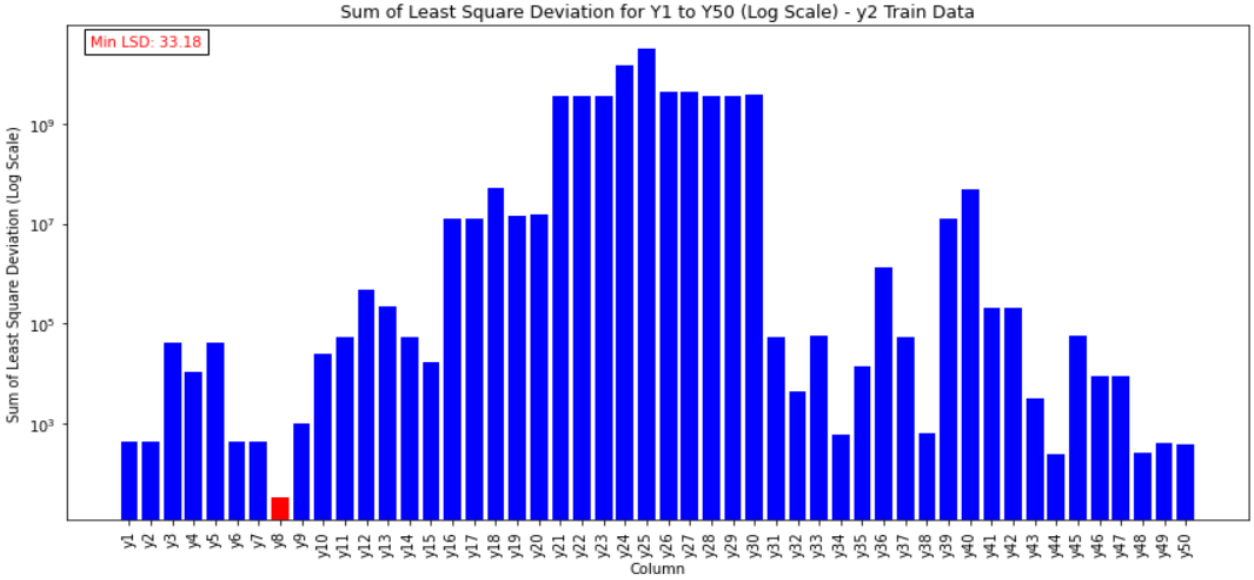
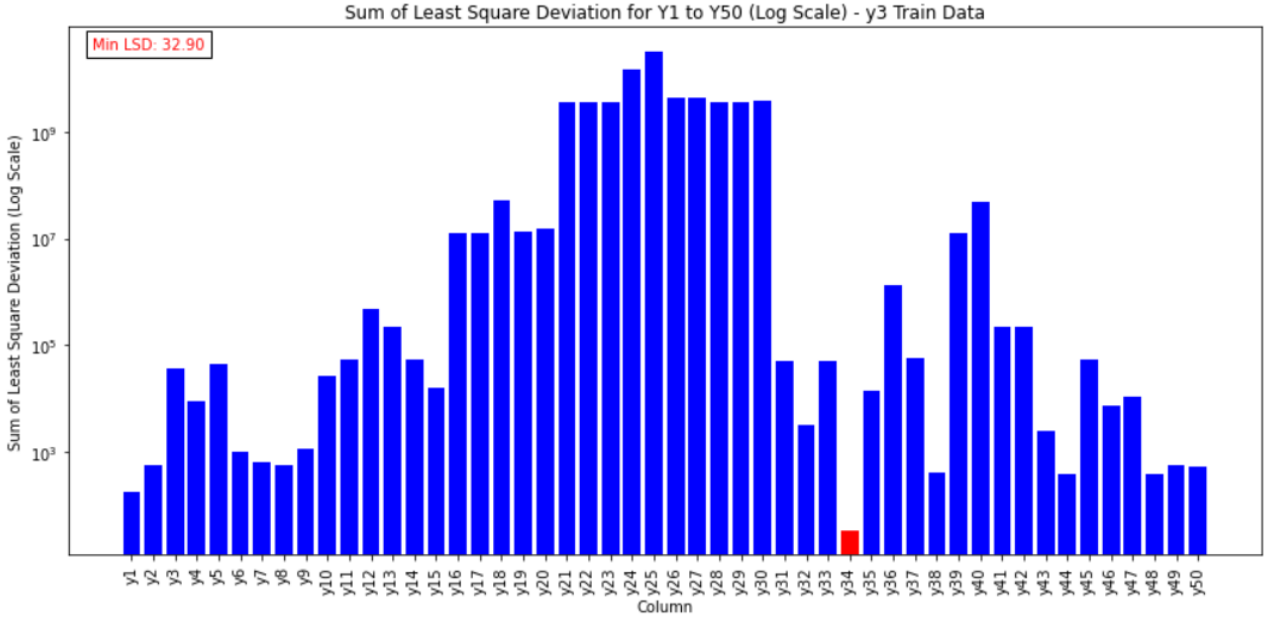


Figure 8: Bar charts showing results of least-squares based approach to determine ideal functions for y1 train data



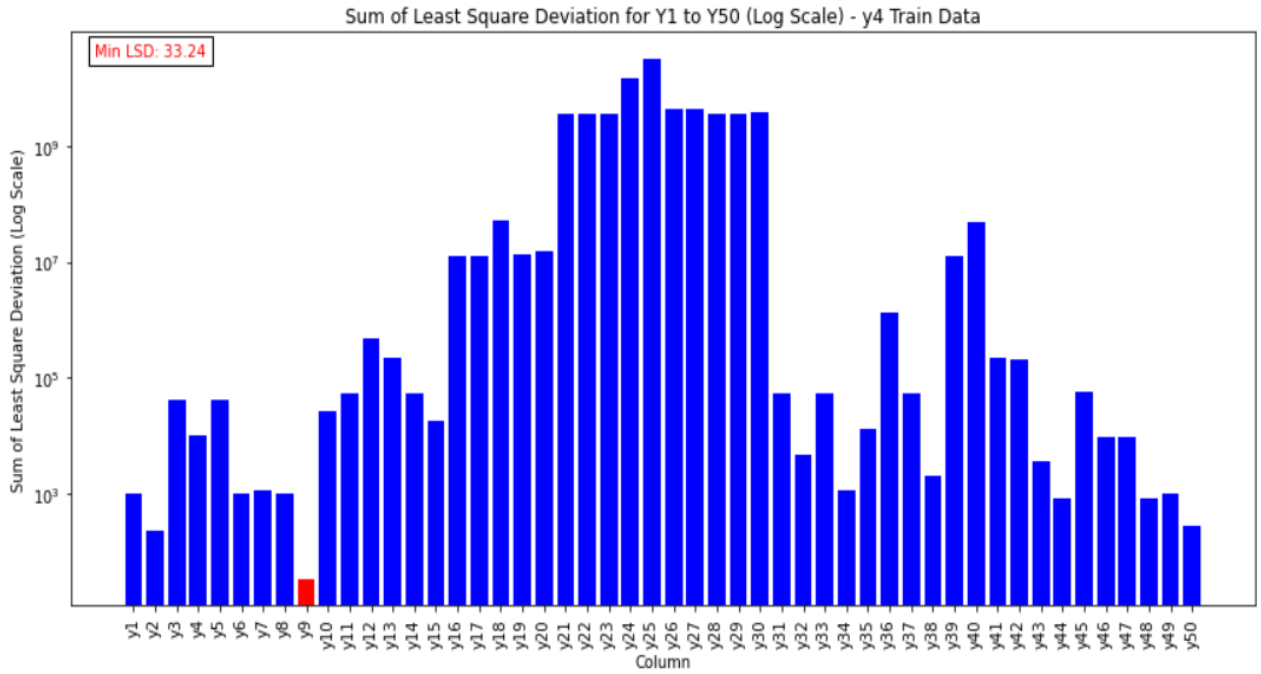

```
The minimum sum of least square deviation is 33.18365390411227
The ideal function for the train data Y2 is: y8
```

Figure 9: Bar charts showing results of least-squares based approach to determine ideal functions for y2 train data




```
The minimum sum of least square deviation is 32.90443640128569
The ideal function for the train data Y3 is: y34
```

Figure 10: Bar charts showing results of least-squares based approach to determine ideal functions for y3 train data




```
The minimum sum of least square deviation is 33.23550313769641
The ideal function for the train data Y4 is: y9
```

Figure 11: Bar charts showing results of least-squares based approach to determine ideal functions for y4 train data

The above bar charts were generated using the least squares approach based on the criteria provided, and I plotted the bar graph using a log scale because the original scaled graph did not show the bar heights of small values. Logarithmic scales provide for more balanced data representation over a wide range of values(“Logarithmic Scale,” 2023), and the graph more uniformly distributes the data, making it simpler to spot patterns over the whole range.

First, the squared differences were calculated between each individual column of the train data and each available column of ideal data, and this process was repeated for four distinct columns of train data, as the given train data contained four columns (y1 to y4). Furthermore, the calculated differences between each individual column of the train data and each available column of ideal data are totaled together, and the resulting minimal value among the rest of the ideal columns sum differences values is assigned to the train data. The red bar represents the ideal function that was chosen as the best fit for the individual train functions.

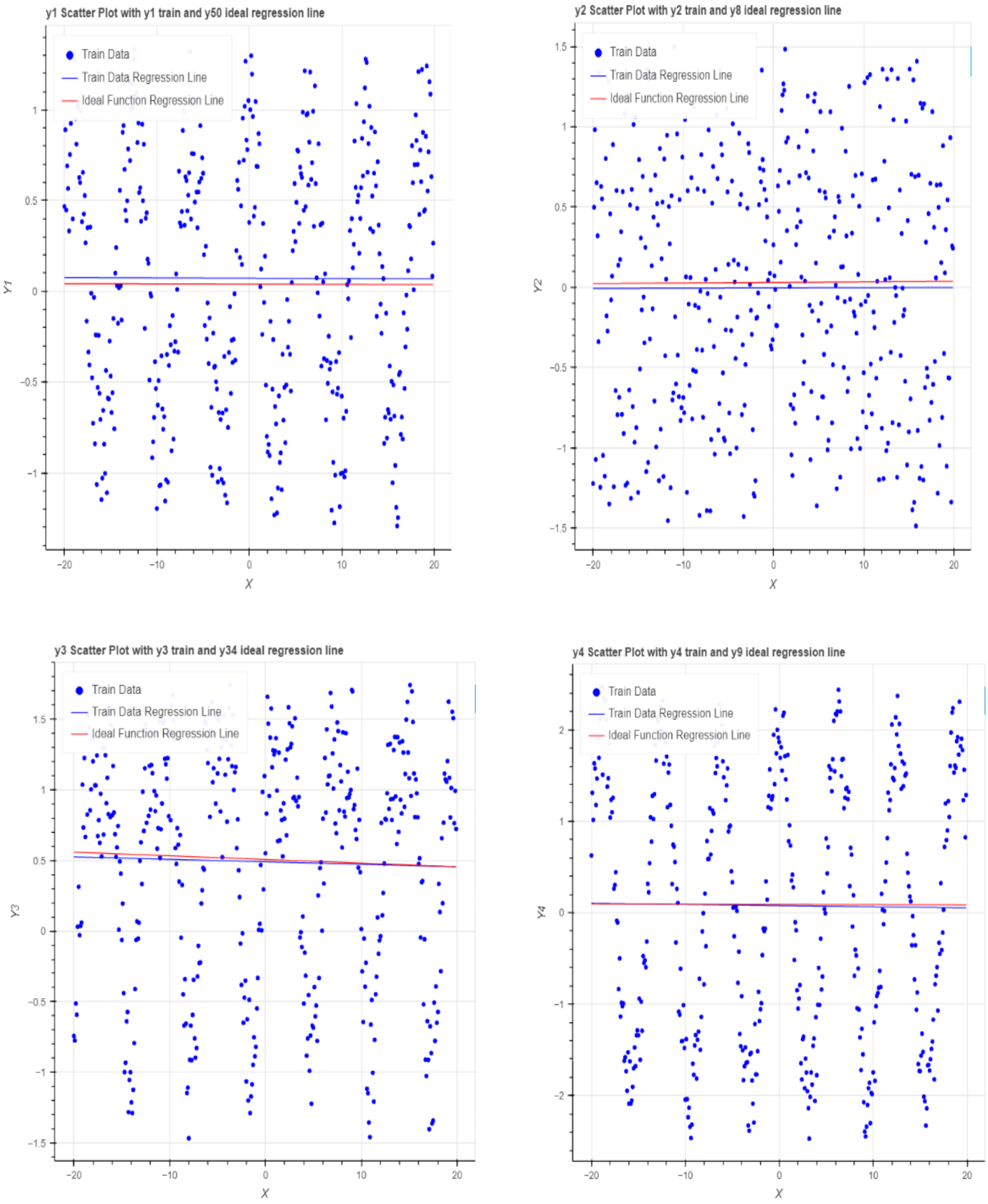


Figure 12: Scatter plot of training functions displaying regression lines for train and ideal functions in a least-squares-based approach to determining ideal functions

The above plots were generated by first creating a (x, y) scatter plot for the train function. Following that, a regression line for both the train function and the ideal function selected in the previous step was displayed. We can observe that the ideal functions chosen for each train function overlap or are quite close to one another, indicating that the degree of correlation is higher(tutorhelpdesk.com, n.d.).

2.8.2 Mean Square error method

As indicated in the section on the proposed framework, the MSE method is used as an alternative evaluation metric to evaluate the outcomes of the least squares approach, and we received the following output results from the MSE method in selecting the best fit for the four ideal functions:

Output: {'y1': ('y50', 0.08536790556118481), 'y2': ('y8', 0.08295913476028068), 'y3': ('y34', 0.08226109100321423), 'y4': ('y9', 0.08308875784424102)}

By comparing the results of the least square and MSE methods, we can determine that the best fit four ideal functions are **Y50, Y8, Y34, and Y9** for train data **Y1, Y2, Y3, and Y4**, respectively.

2.9 Finding R-square value between test and four selected ideal function

R-squared (R^2) is a statistical measure that quantifies the amount of variation explained by an independent variable in a regression model for a dependent variable. R-squared values vary from 0 to 1, and they are frequently expressed as percentages ranging from 0% to 100%(*R-Squared*, n.d.).

Case1:

A value of 0 indicates that the model explains no variation in the dependent variable.

Case2:

A value of 1 indicates that the model explains all of the variability.

$$\text{R-Squared} = \frac{SS_{regression}}{SS_{total}}$$

As per the research question, calculate the R-squared value of each of the four ideal functions with the test dataset. In the dataset of ideal and test, where test data contains 90 data points and ideal function contains 400 data points due to this uneven size, we have matched and filtered the 90 data points from ideal data with the x-values of test data. After matching and filtering it with x-values (key identifiers) of test data, the R-squared values obtained are as follows:

1) R-square value between y50 ideal function and Y test data points: 0.055671226539079854
2) R-square value between y8 ideal function and Y test data points: -0.16603176388518248
3) R-square value between y34 ideal function and Y test data points: -0.14650972344563318
4) R-square value between y9 ideal function and Y test data points: -0.6052855355333193

In this case, the negative R-squared values indicate that the ideal functions (y8, y34, and y9) do not adequately fit the Y test data points. The positive but low R-squared value of y50 shows a very poor relationship between the y50 ideal function and the Y test data points. *As a result, we can deduce that the y50 ideal function will produce more mapped values than others.*

2.10 Mapping Test Dataset with Ideal Function

The results shown below are the outcome of a specific process. To begin, calculate the absolute difference between the test points and the predicted regression values for all four ideal functions. Furthermore, we must comprehend linear regression in order to obtain predicted values for four ideal functions, as explained below.

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).

Because this project only has one dependent variable and one independent variable, linear regression analysis may be a suitable approach for analyzing the data.

The formula for linear regression is as follows:

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

Where,

‘y’ is the dependent variable,

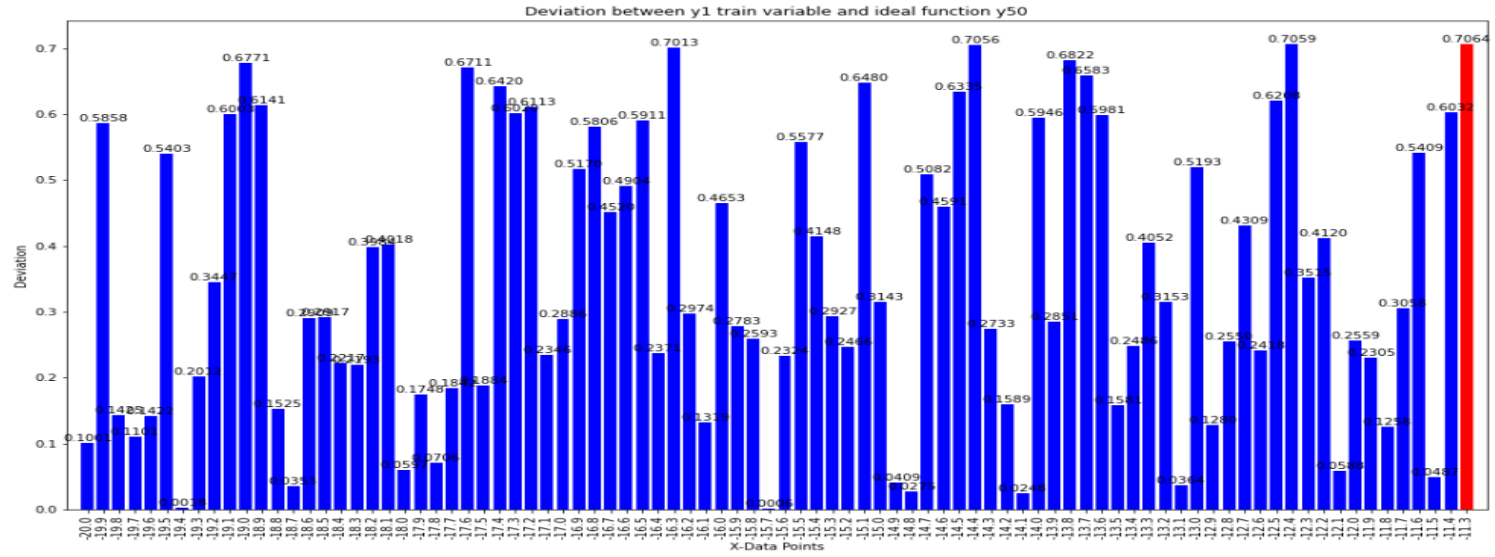
‘x’ is the independent variable,

'β_0' is the intercept, and

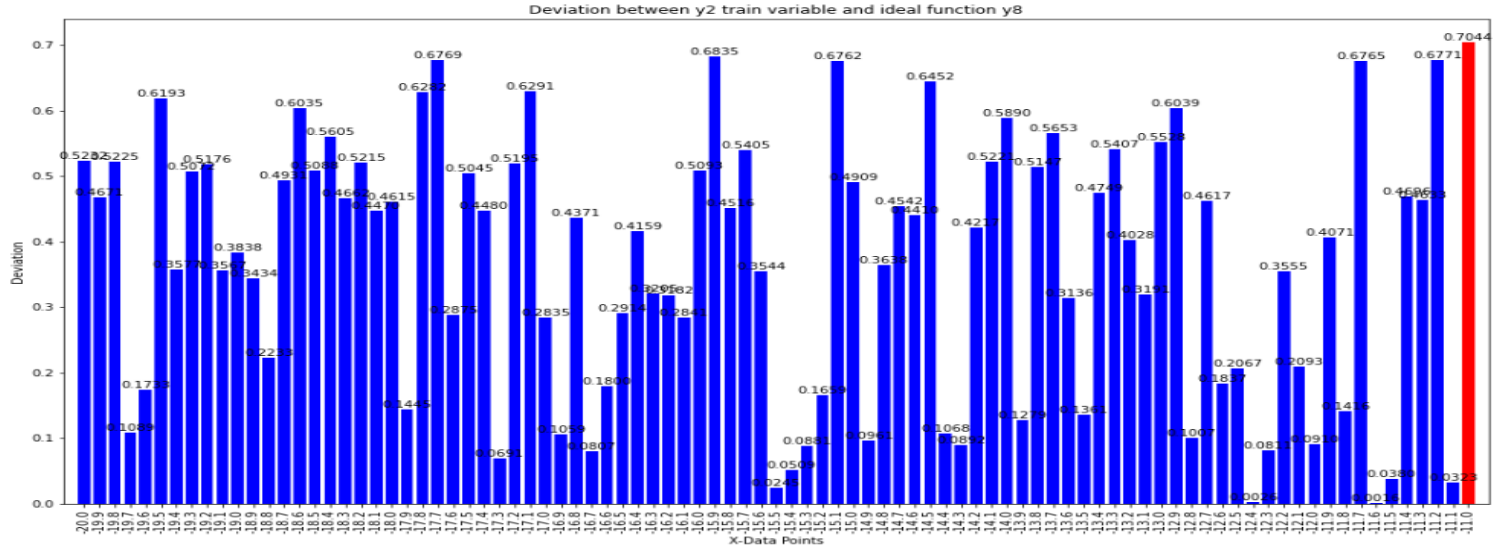
'β_1' is the slope

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.

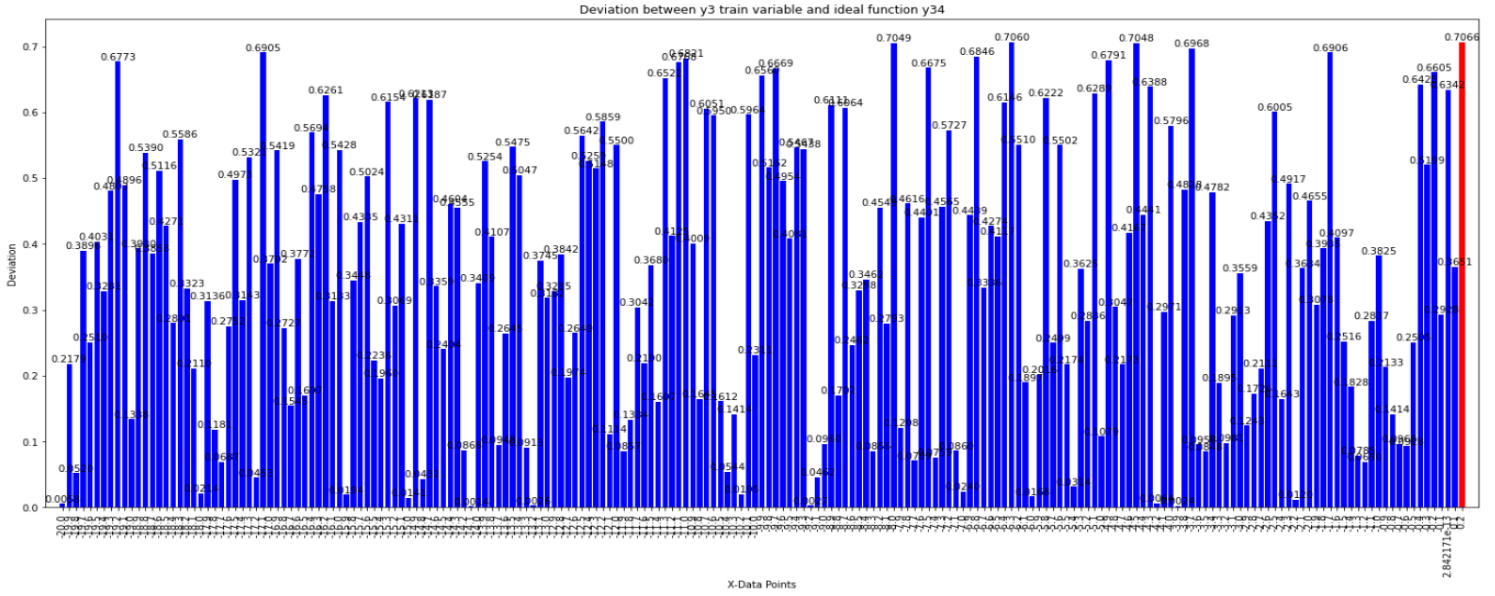
```
Maximum deviation allowed for y1 train variable and selected ideal function y50 is 0.7063503749205766
```




```
Maximum deviation allowed for y2 train variable and selected ideal function y8 is 0.7044444675868375
```




```
Maximum deviation allowed for y3 train variable and selected ideal function y34 is 0.7065589148524843
```



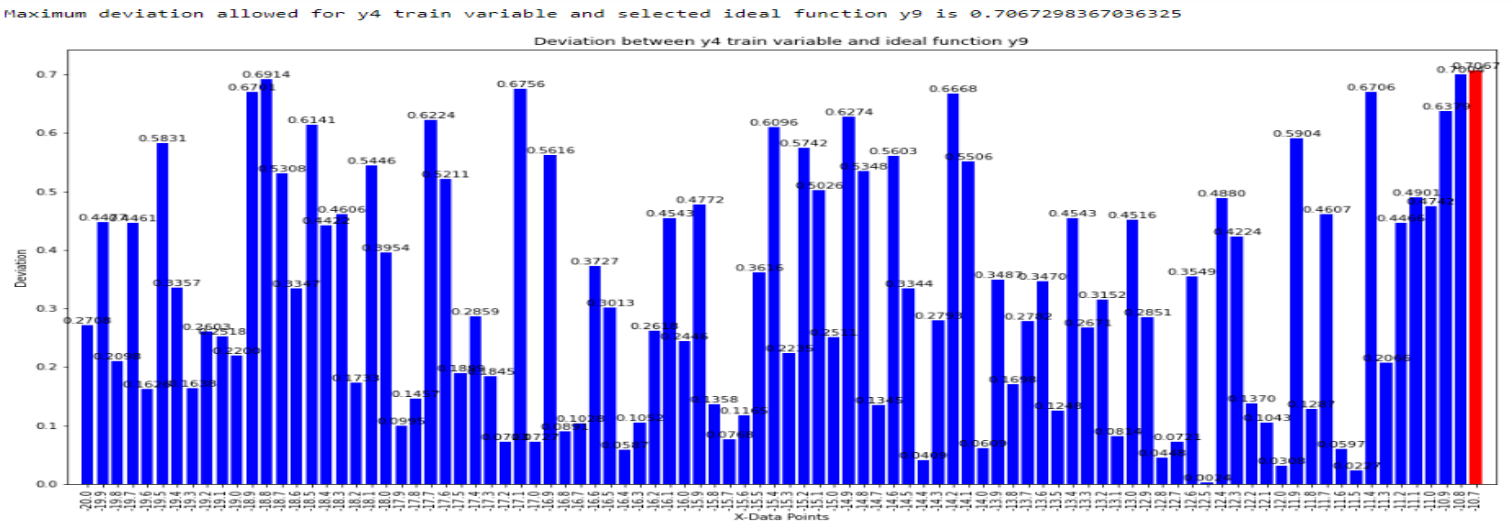
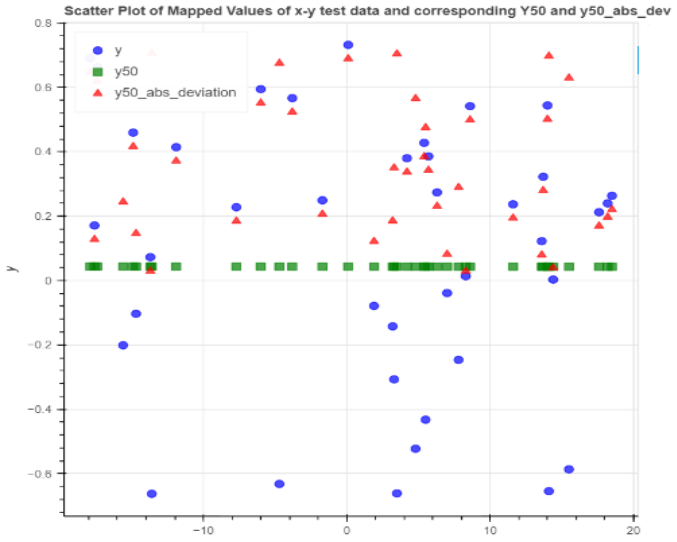
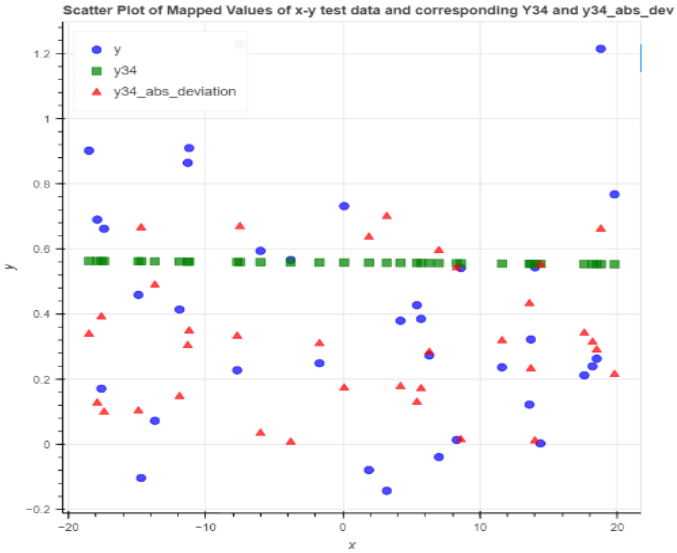


Figure 13: Bar graphs depicting the absolute maximum deviation value between train and ideal for all rows of y values in order to determine the row with the highest deviation value

Furthermore, the scatter plot of mapped values of x-y test data is displayed with each of the four selected ideal functions, and values of the absolute difference determined between the predicted value of the ideal function and the y value of the test data are plotted.





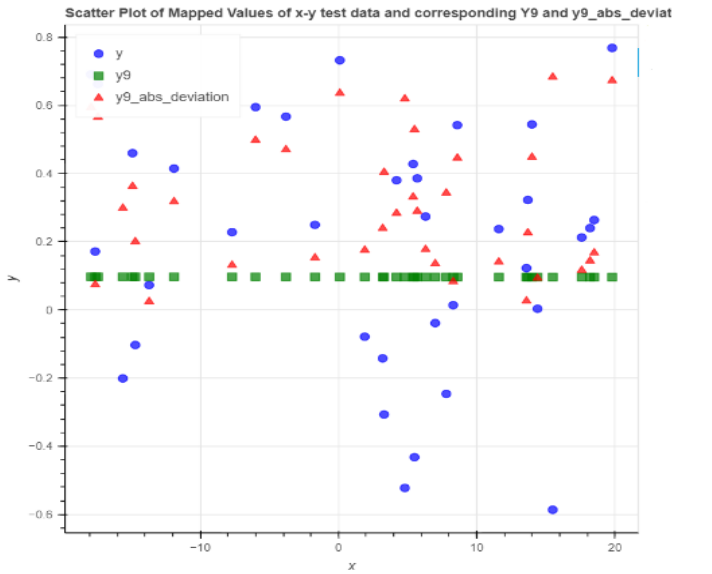


Figure 14: Scatter plot of mapped values of x-y test data with the corresponding selected ideal function and respective absolute deviation

The test points that met the criteria were then displayed alongside the regression line of the ideal function, and upper and lower limits were plotted based on the maximum deviation value.

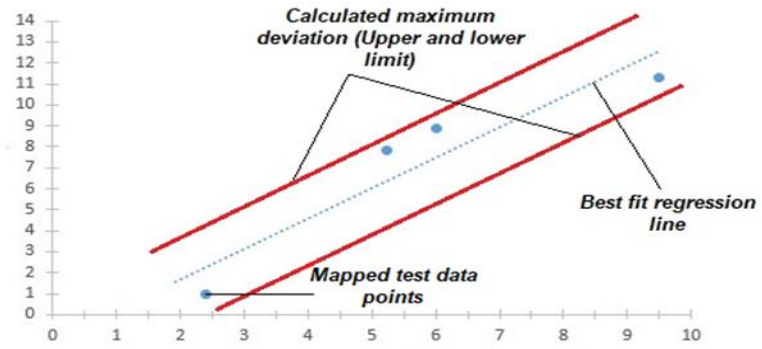
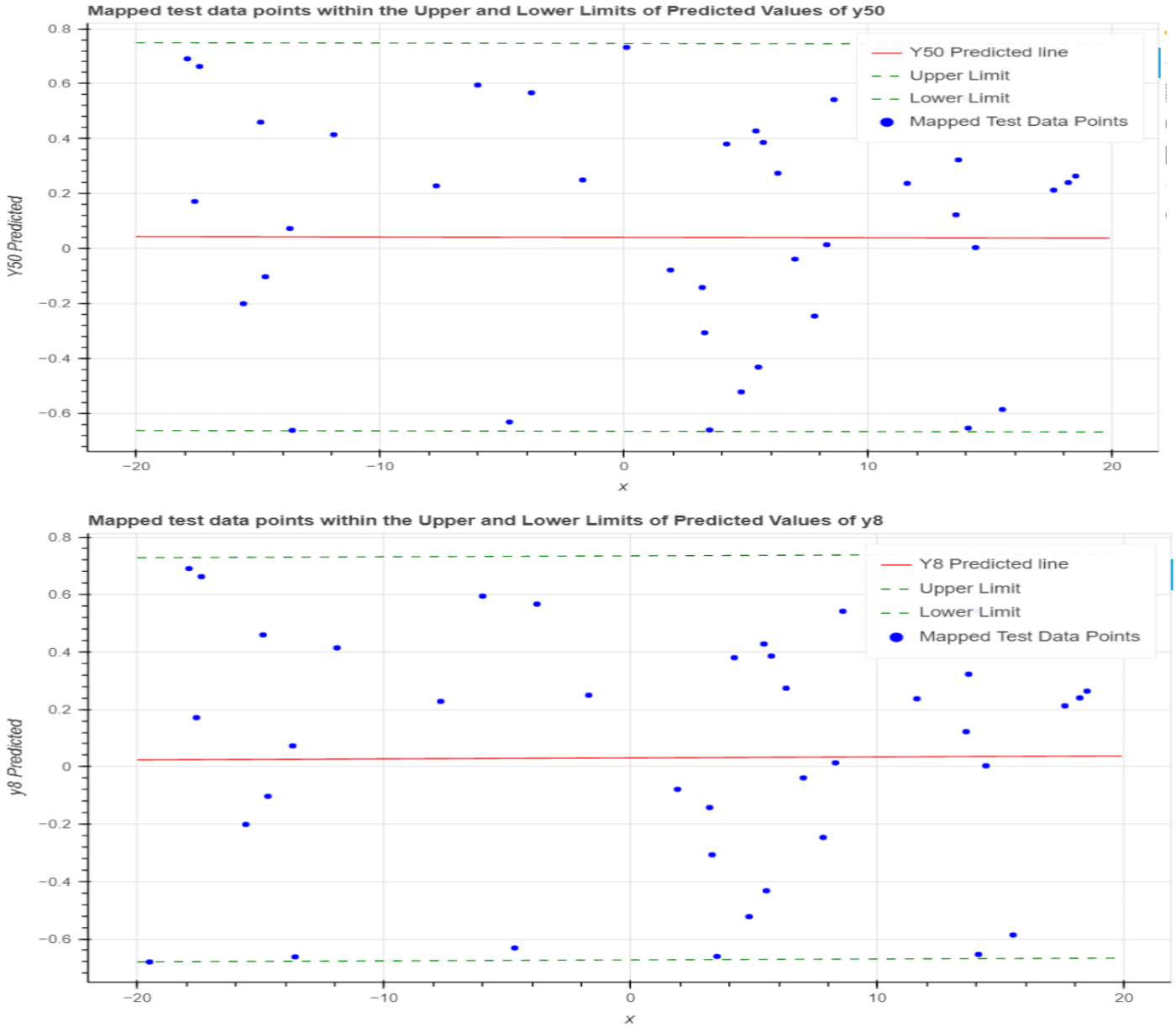


Figure 15: Example of dataset mapping within the constraints (Maximum deviation)



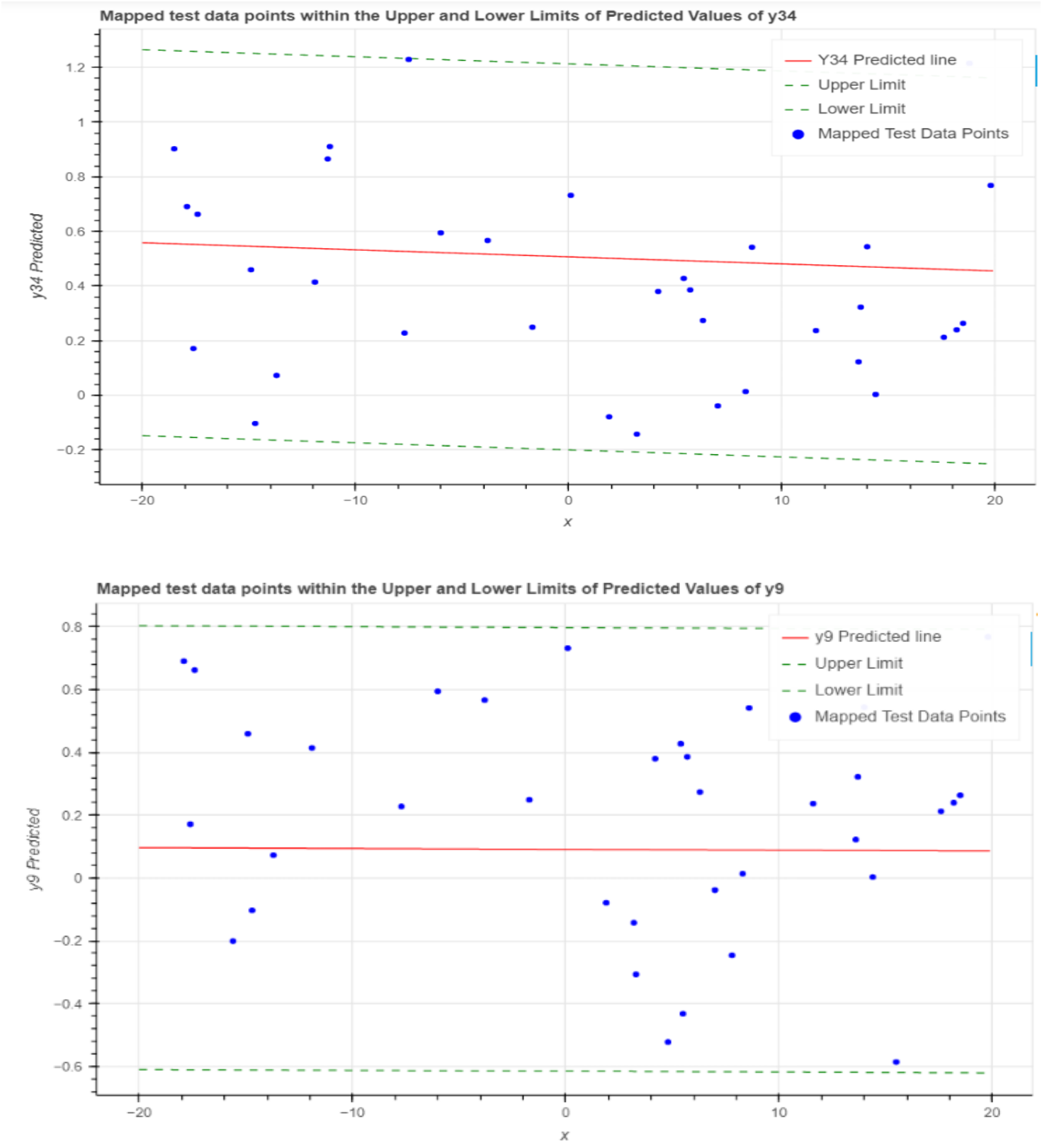


Figure 16: Results showing test data mapped within the maximum deviation for the least-squares approach to determining ideal functions

2.10.1 Mapping Summary:

Least Square Method							
Mapping the test case to the four ideal functions with the Maximum deviation of the train and ideal data	Y50	Y8	Y34	Y9	Not mapped to any selected ideal function	Multiple mapped	The total test case is uniquely mapped to all ideal functions
Count	39	39	35	36	44	103	46

Table 7: Outcome of the final test-ideal mapping for the least square approach

The results of the final test-ideal mapping for the least square approach used to determine the ideal functions are shown in the table above. It shows how many test points were mapped to their respective ideal functions based on the highest deviation value, as well as how many multiple mappings happened, showing that one test point was mapped to multiple ideal functions. It also displays the number of test points that did not map to any of the ideal functions that were chosen.

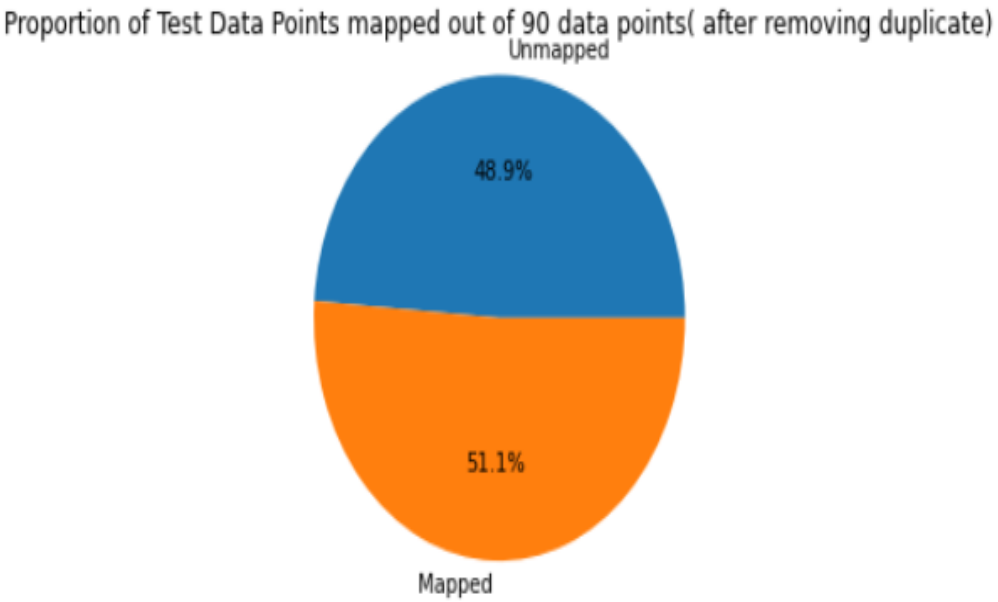


Figure 17: Visual representation of the proportion of the total test case uniquely mapped to all ideal functions along with the unmapped points

Chapter 3

Conclusion and Future work

3.1 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, in order 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 choice 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 **y50, y8, y34, and y9.**

Furthermore, 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.

Finally, mapping the test data with four selected ideal functions and drawing the mapping output of how many counts of test cases are mapped to ideal functions are represented in the table of the mapping summary, where it was found that the y50 ideal function is mapped to more test points, i.e., 39 test data points according to our previous estimation. Furthermore, the y8 ideal function has mapped 39, y34 has mapped 35, and y9 has mapped 36 test data points. Furthermore, it was discovered that 51.1% of the test data points are uniquely mapped to all selected ideal functions, whereas 48.9% are not mapped to any specified ideal function.

Various statistical approaches and techniques, such as "Least square", "MSE", and "R-square method", were used in the subsequent stages of the study to identify the required solutions to the provided objective of analyzing the performance of electrical components or systems.

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

3.2 Future Work

The investigation was carried out using traditional and statistical problem-solving methods based on a set of criteria. Analyzing the performance of an electronic component in the modern era can benefit from techniques other than traditional and statistical approaches, such as machine learning algorithms that can be trained on ideal, train, and test data to analyze component performance, Deep learning techniques that use neural networks, anomaly detection algorithms, predictive analytics techniques, Bayesian networks, and so on.

Chapter 4

Additional Task

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 required 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>

Assuming that 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.

Bibliography

Elements of Data Visualization—Sairam Tangirala. (n.d.). Retrieved April 3, 2023, from https://sites.google.com/site/tangiralasairam/visualizations/data-visualization

Gawali, S. (2021, October 9). How to Deal with Missing Data using Python. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2021/10/how-to-deal-with-missing-data-using-python/

Introduction to Data Imputation | Simplilearn. (2022, November 3). Simplilearn.Com. https://www.simplilearn.com/data-imputation-article

Joby, A. (n.d.). *What Is Data Preprocessing? 4 Crucial Steps to Do It Right*. Retrieved July 2, 2023, from https://learn.g2.com/data-preprocessing

Khan, S. (2020, February 11). *What is Heatmap Visualization? When & How to Use? | VWO*. Blog. https://vwo.com/blog/heatmap-visualization/

Least Square Method. (n.d.). VEDANTU. Retrieved June 25, 2023, from https://www.vedantu.com/maths/least-square-method, https://www.vedantu.com/maths/least-square-method

Least Square Method—Formula, Definition, Examples. (n.d.). Cuemath. Retrieved March 30, 2023, from https://www.cuemath.com/data/least-squares/

Logarithmic scale. (2023). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Logarithmic_scale&oldid=1154345647

M, P. (2021, October 28). End-to-End Introduction to Evaluating Regression Models. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2021/10/evaluation-metric-for-regression-models/

MALI, K. (2021, October 4). Everything you need to Know about Linear Regression! *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2021/10/everything-you-need-to-know-about-linear-regression/

Mulani, S. (2020, October 19). *Detection and Removal of Outliers in Python—An Easy to Understand Guide—AskPython*. https://www.askpython.com/python/examples/detection-removal-outliers-in-python

Pandian, S. (2021, August 1). Effective Data Visualization Techniques in Data Science Using Python. *Analytics Vidhya*. https://www.analyticsvidhya.com/blog/2021/08/effective-data-visualization-techniques-in-data-science-using-python/

Python, R. (n.d.). *Python REST APIs With Flask, Connexion, and SQLAlchemy – Part 2 – Real Python*. Retrieved June 11, 2023, from https://realpython.com/flask-connexion-rest-api-part-2/

Regression analysis. (2023). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Regression_analysis&oldid=1145196383

Removing Duplicated Data in Pandas: A Step-by-Step Guide. (n.d.). Retrieved June 11, 2023, from https://blog.hubspot.com/website/duplicated-pandas

R-Squared: Definition, Calculation Formula, Uses, and Limitations. (n.d.). Investopedia. Retrieved June 25, 2023, from https://www.investopedia.com/terms/r/r-squared.asp

Shafqat, H. (n.d.). *What are SQLite and SQLAlchemy*. Retrieved June 11, 2023, from https://linuxhint.com/sqlite-and-sqlitealchemy/

Singh, P., Singh, N., Singh, K. K., & Singh, A. (2021). Chapter 5—Diagnosing of disease using machine learning. In K. K. Singh, M. Elhoseny, A. Singh, & A. A. Elngar (Eds.), *Machine Learning and the Internet of Medical Things in Healthcare* (pp. 89–111). Academic Press. https://doi.org/10.1016/B978-0-12-821229-5.00003-3

SQLAlchemy Core â□□ Expression Language. (n.d.). Retrieved June 11, 2023, from https://www.tutorialspoint.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.tutorhelpdesk.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 https://www.tibco.com/reference-center/what-is-a-box-plot

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

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

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. Retrieved 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.simplilearn.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*

```
# Install and import the Python libraries

# Importing the libraries needed for the project

'''

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.

'''

# Library for testing

import unittest


# Library for accessing database

from sqlalchemy import create_engine

import sqlite3

# Libraries for reading and manipulating data

import pandas as pd  # pandas is used for data manipulation and analysis

import numpy as np   # numpy is essential for numerical computing

from sklearn.metrics import r2_score # r2_score function from scikit-learn for calculating R-squared

from scipy.stats import linregress  # linregress function from SciPy for linear regression analysis


# Libraries for data visualization

import matplotlib.pyplot as plt #plotting library for creating static, interactive, and animated visualizations

import seaborn as sns #seaborn is a high-level interface built on top of matplotlib, providing additional functionalities and making it easier to create attractive statistical graphics

from bokeh.models import Label #for interactive data visualization that targets modern web browsers. Label is a Bokeh model used to add text annotations to plots

from bokeh.plotting import figure, show #bokeh.plotting provides a high-level interface for creating visualizations, and show is used to display Bokeh plots in the output
```

```
from bokeh.models import ColumnDataSource #ColumnDataSource is a Bokeh data structure that allows for efficient and convenient data handling for Bokeh plots

from bokeh.io import output_notebook #for data exploration, analysis, and presentation in a collaborative and interactive environment


# Library for warning: 'warnings' module

import warnings  # Import the warnings module

warnings.filterwarnings('ignore')  # Set a filter to ignore warning messages

# Enable inline plotting for matplotlib in Jupyter Notebook

%matplotlib inline
```

```
# Set the file paths for the datasets

train_data_file = r"C:\Users\HP\OneDrive\Desktop\dataset\train.csv"

ideal_data_file = r"C:\Users\HP\OneDrive\Desktop\dataset\ideal.csv"

test_data_file = r"C:\Users\HP\OneDrive\Desktop\dataset\test.csv"


# Load training data

train_data = pd.read_csv(train_data_file)


# Load ideal functions

ideal_data = pd.read_csv(ideal_data_file)


# Load test data

test_data = pd.read_csv(test_data_file)


# Remove duplicate X values from test data and calculate the mean of corresponding Y values

test_data = test_data.groupby('x').mean().reset_index()


# Specify the absolute path to the database file

database_path = r"C:\Users\HP\OneDrive\Desktop\mydatabase.db"


# Create a SQLite database using SQLAlchemy with the modified path

engine = create_engine(f'sqlite:///{database_path}', echo=True)
```

```
# Create tables in the database
with engine.begin() as connection:
    train_data.to_sql('train_data', con=connection, if_exists='replace', index=False)
    ideal_data.to_sql('ideal_data', con=connection, if_exists='replace', index=False)
    test_data.to_sql('test_data', con=connection, if_exists='replace', index=False)

# Confirm successful database creation
print("The database was successfully created, and the data was loaded!")
```

```
#Display the created table names in the database
def display_table_names():
    """
    Connects to an SQLite database engine and displays the table names in the database.

    Returns:
    None
    """
    # Create a SQLite database engine
    engine = create_engine(r'sqlite:///C:\Users\HP\OneDrive\Desktop\mydatabase.db',
echo=True)

    # Get the table names from the database
    table_names = engine.table_names()

    # Display the table names
    for table_name in table_names:
        print(table_name)

# Call the function to display table names
display_table_names()
```

```
#Display the contents of the ideal_data table from a SQLite database
def ideal_data_table():
    """
    Fetches and displays the contents of the ideal_data table from a SQLite database.
    """
    # Create a SQLite database engine
    engine = create_engine('sqlite:///C:\\Users\\HP\\OneDrive\\Desktop\\mydatabase.db',
echo=True)


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


ideal_data_table()
```

```
#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('sqlite:///C:\\Users\\HP\\OneDrive\\Desktop\\mydatabase.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()
```

```
#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('sqlite:///C:\\Users\\HP\\OneDrive\\Desktop\\mydatabase.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()
```

```
# Find number of rows and columns in idea data
print("Number of rows in ideal_data set:", len(ideal_data))
print("Number of columns in ideal_data set:", len(ideal_data.columns))
```

```
# Check first five rows in ideal data
first_five_rows = ideal_data.head(5)
print(first_five_rows)
```

```
# check last five rows in ideal data
last_five_rows = ideal_data.tail(5)
print(last_five_rows)
```

```
ideal_data.info()
```

```
# Describe ideal data frame transpose
ideal_data.describe().T
```

```
#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)
```

```
# Find number of rows and columns in train data
print("Number of rows in train_data set:", len(train_data))
print("Number of columns in train_data set:", len(train_data.columns))
```

```
# Check first five rows in train data
train_data.head(5)
```

```
# Check last five rows in train data
train_data.tail(5)
```

```
# Info of each train dataset
train_data.info()
```

```
# Describe train data
train_data.describe().T
```

```
#Define a function to plot individual sactter plot with regression line for train data variable
def plot_scatter_with_regression(train_data, x_column, y_column, title):
    """
    Plot a scatter plot with a regression line.

    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
    """
    fig, ax = plt.subplots()
    sns.scatterplot(x=x_column, y=y_column, data=train_data, color='blue', ax=ax)
    sns.regplot(x=x_column, y=y_column, data=train_data, scatter=False, line_kws={'color':
'red'}, ax=ax)
    ax.legend(labels=['Train Data Points', 'Regression Line'], loc='upper right')
    plt.title(title)
    plt.show()

# Plot scatter plots with regression lines
plot_scatter_with_regression(train_data, 'x', 'y1', 'Scatter Plot for y1 with regression line')
plot_scatter_with_regression(train_data, 'x', 'y2', 'Scatter Plot for y2 with regression line')
plot_scatter_with_regression(train_data, 'x', 'y3', 'Scatter Plot for y3 with regression line')
plot_scatter_with_regression(train_data, 'x', 'y4', 'Scatter Plot for y4 with regression line')
```

```
# Boxplot of train dataset
'''
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()
```

```
# Find number of rows and columns in test data after removing 10 duplicate rows
print("Number of rows in test_data set:", len(test_data))
print("Number of columns in test_data set:", len(test_data.columns))
```

```
# Check first five rows in test data
test_data.head()
```

```
# Check last five rows in test data
test_data.tail()
```

```
# Info of each test dataset
test_data.info()
```

```
# Describe test data
test_data.describe().T
```

```
# 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()
```

```
#Calculating least square deviation between ideal data and train data
#Selecting best fit four ideal function for individual train data which shows sum of minimum least sqaure deviation using OOP concept
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.
        """
```

```
    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):
    """
    A class for calculating and analyzing the sum of least square deviations between train_data
and ideal_data.
    Inherits from DeviationAnalysis.
    """
    def __init__(self, train_data, ideal_data, y_index):
        """
        Initializes LeastSquareDeviation 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.
        """
```

```
        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()
```

```
#Unit testing deviation analysis code to check the correctness of least square method
class DeviationAnalysis:
    def __init__(self, train_data, ideal_data, y_index):
        """
        Initializes the DeviationAnalysis class with the given data and index.
        Parameters:
            train_data (dict): A dictionary containing train data for different columns.
            ideal_data (dict): A dictionary containing ideal data for different columns.
            y_index (int): The index of the column to analyze (1 to n).

        Returns:
            None
        """
        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 Least Squared Deviation (LSD) for the specified column index.

        The LSD measures the difference between the train data and the ideal data for the specified column.
        Parameters:
            None
```

```
        Returns:
            list: A list containing the LSD values for each column in the range from 1 to 4 (inclusive).
        """
        lsd_sum = [ ]
        for i in range(1, 5):  # Update the range based on the available columns in ideal_data
            column_name = f"y{i}"
            train_values = np.array(self.train_data[column_name])
            ideal_values = np.array(self.ideal_data[column_name])
            lsd = np.sum((train_values - ideal_values) ** 2)
            lsd_sum.append(lsd)
        return lsd_sum


class DeviationAnalysis_Test(unittest.TestCase):
    def test_calculate_lsd(self):
        """
        Test case for the calculate_lsd method of DeviationAnalysis.


        Compares the calculated LSD values with the expected values based on the number of
columns.


        Parameters:
            None


        Returns:
            None
        """
        train_data = {
            'y1': [2, 3, 4],
            'y2': [5, 6, 7],
            'y3': [8, 9, 10],
            'y4': [11, 12, 13],
        }
```

```
        ideal_data = {
            'y1': [3, 4, 5],
            'y2': [6, 7, 8],
            'y3': [9, 10, 11],
            'y4': [12, 13, 14],
        }
        y_index = 1
        deviation_analysis = DeviationAnalysis(train_data, ideal_data, y_index)
        lsd_sum = deviation_analysis.calculate_lsd()

        expected_lsd_sum = [3, 3, 3, 3]  #  Expected values based on the number of columns

        self.assertEqual(lsd_sum, expected_lsd_sum)

if __name__ == '__main__':
    unittest.main(argv=[''], exit=False)
```

```
#Plotting a scatter plot for train data with the corresponding selected regression line of ideal data
def plot_regression_scatter(x, y, ideal_func_data, title, x_label, y_label):
    """
    Plot a scatter plot for train data with corresponding regression lines of ideal data.
    Parameters:
        x (numpy.ndarray): Array containing the input features (x values) for the scatter plot.
        y (numpy.ndarray): Array containing the corresponding target values (y values) for the scatter plot.
        ideal_func_data (numpy.ndarray): Array containing the y values of the ideal function used for the ideal regression line.
        title (str): Title of the plot.
        x_label (str): Label for the x-axis.
        y_label (str): Label for the y-axis.
    Returns:
        None. Displays the plot using Bokeh's show() function.
    """
```

```
    # Create a new figure
    p = figure(title=title, x_axis_label=x_label, y_axis_label=y_label)

    # Scatter plot
    p.circle(x, y, legend_label='Train Data', color='blue')

    # Regression line for train data
    train_coeffs = np.polyfit(x, y, 1)
    train_line = np.polyval(train_coeffs, x)
    p.line(x, train_line, legend_label='Train Data Regression Line', color='blue')

    # Regression line for selected ideal function column
    ideal_func_coeffs = np.polyfit(x, ideal_func_data, 1)
    ideal_func_line = np.polyval(ideal_func_coeffs, x)
    p.line(x, ideal_func_line, legend_label='Ideal Function Regression Line', color='red')

    # Add legend
    p.legend.location = 'top_left'

    # Add title
    p.add_layout(Label(text=title, text_font_size='16pt', text_font_style='bold', render_mode='css',
y=550))

    # Show the plot
    show(p)

# Plot scatter plot with regression lines
plot_regression_scatter(train_data['x'], train_data['y1'], ideal_data['y50'], 'y1 Scatter Plot with y1
train and y50 ideal regression line', 'X', 'Y1')
plot_regression_scatter(train_data['x'], train_data['y2'], ideal_data['y8'], 'y2 Scatter Plot with y2
train and y8 ideal regression line', 'X', 'Y2')
plot_regression_scatter(train_data['x'], train_data['y3'], ideal_data['y34'], 'y3 Scatter Plot with y3
train and y34 ideal regression line', 'X', 'Y3')
plot_regression_scatter(train_data['x'], train_data['y4'], ideal_data['y9'], 'y4 Scatter Plot with y4
train and y9 ideal regression line', 'X', 'Y4')
```

```
#Evaluation metric MSE method to confirm the selected 4 ideal function from least square method including user defined exception handling
class EmptyMappingDictionaryError(Exception):
    """Exception raised when the mapping dictionary is empty."""
    def __init__(self, message="Empty mapping dictionary."):
        self.message = message
        super().__init__(self.message)
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 feature.
    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 values.
        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)}")
```

```
#Calculate the R-squared value of each selected four ideal functions with the test dataset based on matching x values


# Connect to the SQLite database
conn = engine.connect()
# Query the ideal data for columns y50, y8, y34, y9
ideal_query = "SELECT x, y50, y8, y34, y9 FROM ideal_data"
ideal_results = conn.execute(ideal_query).fetchall()
# Convert the ideal data into a pandas DataFrame
ideal_df = pd.DataFrame(ideal_results, columns=['x', 'y50', 'y8', 'y34', 'y9'])
# Query the test data for columns x, y
test_query = "SELECT x, y FROM test_data"
test_results = conn.execute(test_query).fetchall()
# Convert the test data into a pandas DataFrame
test_df = pd.DataFrame(test_results, columns=['x', 'y'])
# Initialize a dictionary to store the R-squared values
r_squared_values = {}
# Iterate over the columns of the ideal data
for col in ['y50', 'y8', 'y34', 'y9']:
    # Merge the test and ideal data based on matching x values
    merged_df = pd.merge(test_df, ideal_df[['x', col]], on='x', how='inner')
    # Calculate the R-squared value
    r_squared = r2_score(merged_df['y'], merged_df[col])
    # Store the R-squared value in the dictionary
    r_squared_values[col] = r_squared


# Print the R-squared values
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}")


# Close the database connection
conn.close()
```

```
#Calculating absolute deviation between selected ideal data and train data to know individual largest deviation and plotting the results via bar graph
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
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['y50'].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 y50 is',
max_dev_1)

# Set up the figure and axis for y1 train variable and ideal function y50

fig1, ax1 = plt.subplots(figsize=(18, 10))

plot_bar_chart(ax1, deviations_y1, max_deviation_index_y1, x_labels_y1, 'Deviation between
y1 train variable and ideal function y50')

plt.show()

# Maximum deviation allowed for 'y2' variable

y2_train_data = np.array(list(train_data['y2'].values))

y2_ideal_data = np.array(list(ideal_data['y8'].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 y8 is',
max_dev_2)

# Set up the figure and axis for y2 train variable and ideal function y8

fig2, ax2 = plt.subplots(figsize=(18, 10))

plot_bar_chart(ax2, deviations_y2, max_deviation_index_y2, x_labels_y2, 'Deviation between
y2 train variable and ideal function y8')

plt.show()

# Maximum deviation allowed for 'y3' variable

y3_train_data = np.array(list(train_data['y3'].values))

y3_ideal_data = np.array(list(ideal_data['y34'].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 y34 is',
max_dev_3)
```

```
 # Set up the figure and axis for y3 train variable and ideal function y34
fig3, ax3 = plt.subplots(figsize=(25, 10))
plot_bar_chart(ax3, deviations_y3, max_deviation_index_y3, x_labels_y3, 'Deviation between y3 train variable and ideal function y34')
plt.show()
# Maximum deviation allowed for 'y4' variable
y4_train_data = np.array(list(train_data['y4'].values))
y4_ideal_data = np.array(list(ideal_data['y9'].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 y9 is', max_dev_4)
# Set up the figure and axis for y4 train variable and ideal function y9
fig4, ax4 = plt.subplots(figsize=(18, 10))
plot_bar_chart(ax4, deviations_y4, max_deviation_index_y4, x_labels_y4, 'Deviation between y4 train variable and ideal function y9')
plt.show()
```

```
 #Calculate absolute deviations between predicted values of selected ideal function and test data 'y' values


def calculate_and_store_abs_deviations(database_path):
    """
    Calculate absolute deviations between predicted values of selected ideal function and test data 'y' values,
    and store the updated test data back to the database.

    Parameters:
        database_path (str): The absolute path to the SQLite database file.

    Returns:
        None
    """
```

```
# Calculate regression values for each 'y' variable of selected ideal function and store them in a dictionary
    regression_results = {}
    y_variables = ['y50', 'y8', 'y34', 'y9']
    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)
# Calculate predicted values for each 'y' variable of selected ideal function and store them in a dictionary
    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


    # Access predicted values for each 'y' variable of selected ideal function:
    y50_predicted = predicted_values['y50']
    y8_predicted = predicted_values['y8']
    y34_predicted = predicted_values['y34']
    y9_predicted = predicted_values['y9']
    # Map the predicted values to the test data
    test_data['y50_predicted'] = y50_predicted
    test_data['y8_predicted'] = y8_predicted
    test_data['y34_predicted'] = y34_predicted
    test_data['y9_predicted'] = y9_predicted
```

```
# Calculate the absolute deviation between mapped predicted values and test data 'y' values
    test_data['y50_abs_deviation'] = np.abs(test_data['y'] - test_data['y50_predicted'])
    test_data['y8_abs_deviation'] = np.abs(test_data['y'] - test_data['y8_predicted'])
    test_data['y34_abs_deviation'] = np.abs(test_data['y'] - test_data['y34_predicted'])
    test_data['y9_abs_deviation'] = np.abs(test_data['y'] - test_data['y9_predicted'])


    # Print the test data with mapped predicted values and absolute deviations
    print(test_data)
    # Save the updated test data with absolute deviations to the database
    conn = sqlite3.connect(database_path)
    test_data.to_sql('test_data', con=conn, if_exists='replace', index=False)
    conn.close()
    # Confirm successful update
    print("Mapped predicted values and absolute deviations calculated and stored in the data-
base!")
# Call the function to calculate and store absolute deviations
calculate_and_store_abs_deviations(database_path)
```

```
#Filtering and Plotting Scatter plot of mapped values of x-y test data with corresponding se-
lected ideal function and respective absolute deviation


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 infor-
mation.
    Parameters:
        test_data (pd.DataFrame): DataFrame containing test data with 'x', 'y', 'y50_predicted',
'y50_abs_deviation',
                                  'y8_predicted', 'y8_abs_deviation', 'y34_predicted', 'y34_abs_deviation',
                                  'y9_predicted', and 'y9_abs_deviation' columns.
        max_dev_1 (float): Maximum deviation allowed for the 'y50_abs_deviation' variable.
        max_dev_2 (float): Maximum deviation allowed for the 'y8_abs_deviation' variable.
        max_dev_3 (float): Maximum deviation allowed for the 'y34_abs_deviation' variable.
        max_dev_4 (float): Maximum deviation allowed for the 'y9_abs_deviation' variable.
```

```
 # Filter the test data based on the conditions
filtered_data_1 = test_data[test_data['y50_abs_deviation'] <= max_dev_1]
filtered_data_2 = test_data[test_data['y8_abs_deviation'] <= max_dev_2]
filtered_data_3 = test_data[test_data['y34_abs_deviation'] <= max_dev_3]
filtered_data_4 = test_data[test_data['y9_abs_deviation'] <= max_dev_4]
# Create Bokeh figures
p1 = figure(title="Scatter Plot of Mapped Values of x-y test data and corresponding Y50 and y50_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 Y8 and y8_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 Y34 and y34_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 Y9 and y9_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']
y50_values = filtered_data_1['y50_predicted']
y50_abs_deviation = filtered_data_1['y50_abs_deviation']


mapped_x_points2 = filtered_data_2['x']
y_values2 = filtered_data_2['y']
y8_values = filtered_data_2['y8_predicted']
y8_abs_deviation = filtered_data_2['y8_abs_deviation']


mapped_x_points3 = filtered_data_3['x']
y_values3 = filtered_data_3['y']
y34_values = filtered_data_3['y34_predicted']
y34_abs_deviation = filtered_data_3['y34_abs_deviation']


mapped_x_points4 = filtered_data_4['x']
y_values4 = filtered_data_4['y']
y9_values = filtered_data_4['y9_predicted']
y9_abs_deviation = filtered_data_4['y9_abs_deviation']
```

```
    # Create DataFrames for the mapped values
    mapped_values1 = pd.DataFrame({'x': mapped_x_points1, 'y': y_values1, 'y50': y50_values, 'y50_abs_deviation': y50_abs_deviation})
    mapped_values2 = pd.DataFrame({'x': mapped_x_points2, 'y': y_values2, 'y8': y8_values, 'y8_abs_deviation': y8_abs_deviation})
    mapped_values3 = pd.DataFrame({'x': mapped_x_points3, 'y': y_values3, 'y34': y34_values, 'y34_abs_deviation': y34_abs_deviation})
    mapped_values4 = pd.DataFrame({'x': mapped_x_points4, 'y': y_values4, 'y9': y9_values, 'y9_abs_deviation': y9_abs_deviation})

    # Print the mapped values for each condition
    print("Mapped Values of x-y test data and corresponding Y50 and y50_abs_deviation:")
    print(mapped_values1[['x', 'y', 'y50', 'y50_abs_deviation']])
    print(f"Total mapped test data points within {max_dev_1} Maximum deviation allowed for y1 train variable and selected ideal function y50: {len(mapped_x_points1)}")
    print("\nMapped Values of x-y test data and corresponding Y8 and y8_abs_deviation:")
    print(mapped_values2[['x', 'y', 'y8', 'y8_abs_deviation']])
    print(f"Total mapped test data points within {max_dev_2} Maximum deviation allowed for y2 train variable and selected ideal function y8: {len(mapped_x_points2)}")
    print("\nMapped Values of x-y test data and corresponding Y34 and y34_abs_deviation:")
    print(mapped_values3[['x', 'y', 'y34', 'y34_abs_deviation']])
    print(f"Total mapped test data points within {max_dev_3} Maximum deviation allowed for y3 train variable and selected ideal function y34: {len(mapped_x_points3)}")
    print("\nMapped Values of x-y test data and corresponding Y9 and y9_abs_deviation:")
    print(mapped_values4[['x', 'y', 'y9', 'y9_abs_deviation']])
    print(f"Total mapped test data points within {max_dev_4} Maximum deviation allowed for y4 train variable and selected ideal function y9: {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['y50'], legend_label='y50', color='green', size=8, alpha=0.7)
    p1.triangle(mapped_values1['x'], mapped_values1['y50_abs_deviation'], legend_label='y50_abs_deviation', color='red', size=8, alpha=0.7)
    p1.legend.location = "top_left"
```

```
        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['y8'], legend_label='y8', color='green', size=8, alpha=0.7)

    p2.triangle(mapped_values2['x'], mapped_values2['y8_abs_deviation'], legend_label='y8_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['y34'], legend_label='y34', color='green', size=8, alpha=0.7)

    p3.triangle(mapped_values3['x'], mapped_values3['y34_abs_deviation'], legend_label='y34_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['y9'], legend_label='y9', color='green', size=8, alpha=0.7)

    p4.triangle(mapped_values4['x'], mapped_values4['y9_abs_deviation'], legend_label='y9_abs_deviation', color='red', size=8, alpha=0.7)

    p4.legend.location = "top_left"

    # Show the plots

    output_notebook()

    show(p1)

    show(p2)

    show(p3)

    show(p4)

# Call the function with test data and maximum deviations

filter_and_plot_data(test_data, max_dev_1, max_dev_2, max_dev_3, max_dev_4)
```

```
#Visual representation proportion of the total test case uniquely mapped to all ideal function along with the un-mapped points

def analyze_mapped_data(test_data, max_dev_1, max_dev_2, max_dev_3, max_dev_4):

    """

    Analyzes the mapped test data based on given conditions for y50, y8, y34, and y9 absolute deviations.
```

```
    Parameters:
        test_data (pd.DataFrame): DataFrame containing the test data with columns 'x',
'y50_abs_deviation',
                                  'y8_abs_deviation', 'y34_abs_deviation', and 'y9_abs_deviation'.
        max_dev_1 (float): Maximum absolute deviation allowed for y50.
        max_dev_2 (float): Maximum absolute deviation allowed for y8.
        max_dev_3 (float): Maximum absolute deviation allowed for y34.
        max_dev_4 (float): Maximum absolute deviation allowed for y9.
    Returns:
        None: The function prints the analysis results and displays a pie chart.
    """
    # Filter the test data based on the condition for y50_abs_deviation
    filtered_data_50 = test_data[test_data['y50_abs_deviation'] <= max_dev_1]
    mapped_x_points_50 = filtered_data_50['x']

    # Filter the test data based on the condition for y8_abs_deviation
    filtered_data_8 = test_data[test_data['y8_abs_deviation'] <= max_dev_2]
    mapped_x_points_8 = filtered_data_8['x']

    # Filter the test data based on the condition for y34_abs_deviation
    filtered_data_34 = test_data[test_data['y34_abs_deviation'] <= max_dev_3]
    mapped_x_points_34 = filtered_data_34['x']

    # Filter the test data based on the condition for y9_abs_deviation
    filtered_data_9 = test_data[test_data['y9_abs_deviation'] <= max_dev_4]
    mapped_x_points_9 = filtered_data_9['x']

    # Create a set of all mapped x points
    mapped_x_points = set(mapped_x_points_50).union(set(mapped_x_points_8),
set(mapped_x_points_34), set(mapped_x_points_9))

    # Count the number of x values not mapped for any y abs deviation
    unmapped_x_points = test_data[~test_data['x'].isin(mapped_x_points)]
    unmapped_count = len(unmapped_x_points)
```

```
    # Concatenate all the mapped x points
    mapped_x_points_all = pd.concat([mapped_x_points_50, mapped_x_points_8, mapped_x_points_34, mapped_x_points_9])
    # Count the number of x values mapped to multiple y abs deviations
    duplicated_count = mapped_x_points_all.duplicated().sum()
    # Count the number of x values uniquely mapped to y abs deviations
    unique_mapped_count = mapped_x_points_all.nunique()
    # Display the counts
    print(f"Total test data points not mapped for any ideal function (y50, y8, y34, y9): {unmapped_count}")
    print(f"Total test data points mapped to multiple selected ideal functions (y50, y8, y34, y9): {duplicated_count}")
    print(f"Total test data points uniquely mapped to all ideal functions (y50, y8, y34, y9): {unique_mapped_count}")
    # Create data for the pie chart
    labels = ['Unmapped', 'Mapped']
    sizes = [unmapped_count, unique_mapped_count]
    # Plot the pie chart
    plt.pie(sizes, labels=labels, autopct='%1.1f%%')
    plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
    plt.title('Proportion of Test Data Points mapped out of 90 data points (after removing duplicates)')
    plt.show()
# Call the function with test data
analyze_mapped_data(test_data, max_dev_1, max_dev_2, max_dev_3, max_dev_4)
```

```
#Plot the predicted values, upper limits, lower limits, and mapped test data points for selected best fit ideal function


def plot_predicted_values(test_data, ideal_data, max_dev, y_column):
    """
    Plot the predicted values, upper limits, lower limits, and mapped test data points for the given 'y_column'.
```

```
    Parameters:
        test_data (DataFrame): The DataFrame containing the test data with columns 'x',
'y50_abs_deviation', 'y8_abs_deviation',
                               'y34_abs_deviation', and 'y9_abs_deviation'.
        ideal_data (DataFrame): The DataFrame containing the ideal data with columns 'index', 'x',
and the 'y_column'.
        max_dev (float): Maximum deviation for 'y_column' predictions.
        y_column (str): Name of the 'y' column to be used for plotting.
    Returns:
        None (displays the plot using Bokeh's show() function).
    """
    # Filter the test data based on the conditions
    filtered_data = test_data[test_data[y_column + '_abs_deviation'] <= max_dev]
    # Extract the mapped values
    mapped_x_points = filtered_data['x']
    y_values = filtered_data['y']
    # Calculate predicted values for the given 'y_column'
    y_regression = linregress(ideal_data.index, ideal_data[y_column])
    slope_y = y_regression.slope
    intercept_y = y_regression.intercept
    y_predicted = slope_y * ideal_data.index + intercept_y
    # Add upper and lower limits for the predicted values of 'y_column'
    y_predicted_upper = y_predicted + max_dev
    y_predicted_lower = y_predicted - max_dev
    # Create a DataFrame to store the results
    results_df = pd.DataFrame({'Data Point': ideal_data.index,
                               'x': ideal_data['x'],
                               y_column: ideal_data[y_column],
                               y_column + ' Predicted': y_predicted,
                               'Upper Limit': y_predicted_upper,
                               'Lower Limit': y_predicted_lower})
    # Print the predicted values, upper limits, lower limits, and corresponding x and y values for
'y_column'
    print(results_df)
```

```
    # Convert the DataFrame to a ColumnDataSource
    source = ColumnDataSource(results_df)
    # Create the plot
    p = figure(title=f'Mapped test data points within the Upper and Lower Limits of Predicted Values of {y_column}',
               x_axis_label='x',
               y_axis_label=f'{y_column} Predicted',
               plot_width=800,
               plot_height=500)
    # Plot the predicted line
    p.line(x='x', y=y_column + ' Predicted', line_color='red', legend_label=f'{y_column} Predicted line', source=source)
    # Plot the upper and lower limits
    p.line(x='x', y='Upper Limit', line_color='green', line_dash='dashed', legend_label='Upper Limit', source=source)
    p.line(x='x', y='Lower Limit', line_color='green', line_dash='dashed', legend_label='Lower Limit', source=source)
    # Plot the mapped test data points
    p.scatter(mapped_x_points, y_values, color='blue', legend_label='Mapped Test Data Points')
    # Show the plot
    show(p)
plot_predicted_values(test_data, ideal_data, max_dev_1, 'y50')
plot_predicted_values(test_data, ideal_data, max_dev_2, 'y8')
plot_predicted_values(test_data, ideal_data, max_dev_3, 'y34')
plot_predicted_values(test_data, ideal_data, max_dev_4, 'y9')
```

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


def filter_and_save_test_data(engine, max_dev_1, max_dev_2, max_dev_3, max_dev_4):
    """
    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 'y50_abs_deviation'.
        max_dev_2 (float): The maximum deviation for 'y8_abs_deviation'.
        max_dev_3 (float): The maximum deviation for 'y34_abs_deviation'.
        max_dev_4 (float): The maximum deviation for 'y9_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 = pd.read_sql_query(test_data_query, engine)
    # Filter the test data based on the condition
    filtered_data1 = test_data_df[test_data_df['y50_abs_deviation'] <= max_dev_1]
    filtered_data2 = test_data_df[test_data_df['y8_abs_deviation'] <= max_dev_2]
    filtered_data3 = test_data_df[test_data_df['y34_abs_deviation'] <= max_dev_3]
    filtered_data4 = test_data_df[test_data_df['y9_abs_deviation'] <= max_dev_4]
    # Extract the mapped values of x points and corresponding y50_abs_deviation
    mapped_x_points1 = filtered_data1['x']
    y_values1 = filtered_data1['y']
    y50_values = filtered_data1['y50_predicted']
    y50_abs_deviation = filtered_data1['y50_abs_deviation']
    # Create a new DataFrame for the mapped values
    mapped_values1 = pd.DataFrame({'x': mapped_x_points1, 'y': y_values1, 'Delta Y':
y50_abs_deviation, 'No. of ideal func': 'y50'})


    # Extract the mapped values of x points and corresponding y8_abs_deviation
    mapped_x_points2 = filtered_data2['x']
    y_values2 = filtered_data2['y']
    y8_values = filtered_data2['y8_predicted']
    y8_abs_deviation = filtered_data2['y8_abs_deviation']
```

```
    # Create a new DataFrame for the mapped values
    mapped_values2 = pd.DataFrame({'x': mapped_x_points2, 'y': y_values2, 'Delta Y': y8_abs_deviation, 'No. of ideal func': 'y8'})

    # Extract the mapped values of x points and corresponding y34_abs_deviation
    mapped_x_points3 = filtered_data3['x']
    y_values3 = filtered_data3['y']
    y34_values = filtered_data3['y34_predicted']
    y34_abs_deviation = filtered_data3['y34_abs_deviation']

    # Create a new DataFrame for the mapped values
    mapped_values3 = pd.DataFrame({'x': mapped_x_points3, 'y': y_values3, 'Delta Y': y34_abs_deviation, 'No. of ideal func': 'y34'})

    # Extract the mapped values of x points and corresponding y9_abs_deviation
    mapped_x_points4 = filtered_data4['x']
    y_values4 = filtered_data4['y']
    y9_values = filtered_data4['y9_predicted']
    y9_abs_deviation = filtered_data4['y9_abs_deviation']

    # Create a new DataFrame for the mapped values
    mapped_values4 = pd.DataFrame({'x': mapped_x_points4, 'y': y_values4, 'Delta Y': y9_abs_deviation, 'No. of ideal func': 'y9'})

    # 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)

# Call the function with arguments for max_dev_1, max_dev_2, max_dev_3, and max_dev_4
filter_and_save_test_data(engine, max_dev_1, max_dev_2, max_dev_3, max_dev_4)
```

```
#Query the database to display all the contents of ideal_data, train_data and test_data table
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)
# Query and display the train_data DataFrame
train_data_df = query_table("train_data")
print("\ntrain_data DataFrame:")
print(train_data_df)
# Query and display the test_data DataFrame
test_data_df = query_table("test_data")
print("\ntest_data DataFrame:")
print(test_data_df)
```

Appendix B: Additional Visualization

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

# Calculate the correlation matrix
correlation_matrix = ideal_data.corr()

# Create the heatmap
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")

# Add a title
plt.title("Correlation Heatmap of Ideal Dataset")

# Display the plot
plt.show()
```

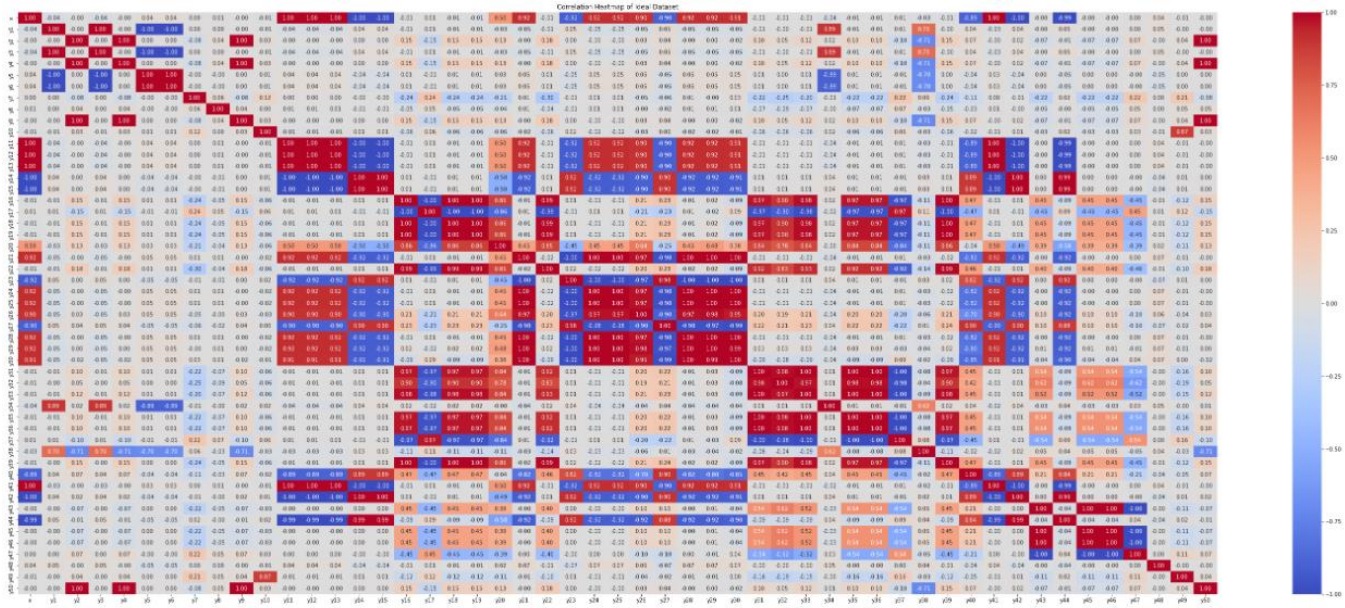


Figure 18: Correlation heatmap of ideal dataset

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

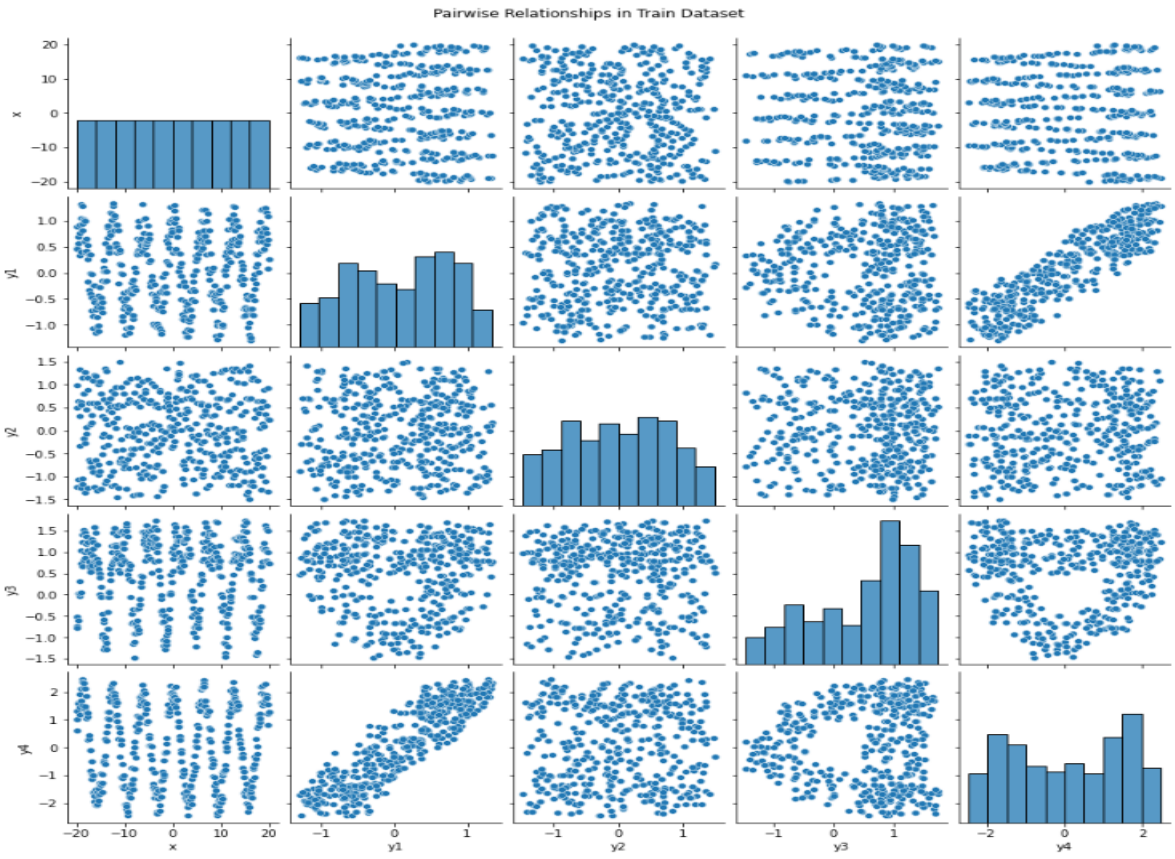


Figure 19: Graph of pairwise relationships in train dataset

```
#Define a function to plot box plot for test data
def hist_box(test_data, col):
    """
    Generate a combination of histogram and boxplot for a given column in the test dataset.

    Parameters:
        test_df (pd.DataFrame): DataFrame containing the 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 in the test dataset
    sns.boxplot(test_data[col], ax=ax_box, showmeans=True)
    ax_box.set(xlabel='')
    # Adding a histogram in the test dataset
    sns.histplot(test_data[col], ax=ax_hist, kde=True)
    ax_hist.set(xlabel=col)
    plt.tight_layout()
    plt.show()
# Visualize hist_box for 'x' variable in test dataset
hist_box(test_data, 'x')
```

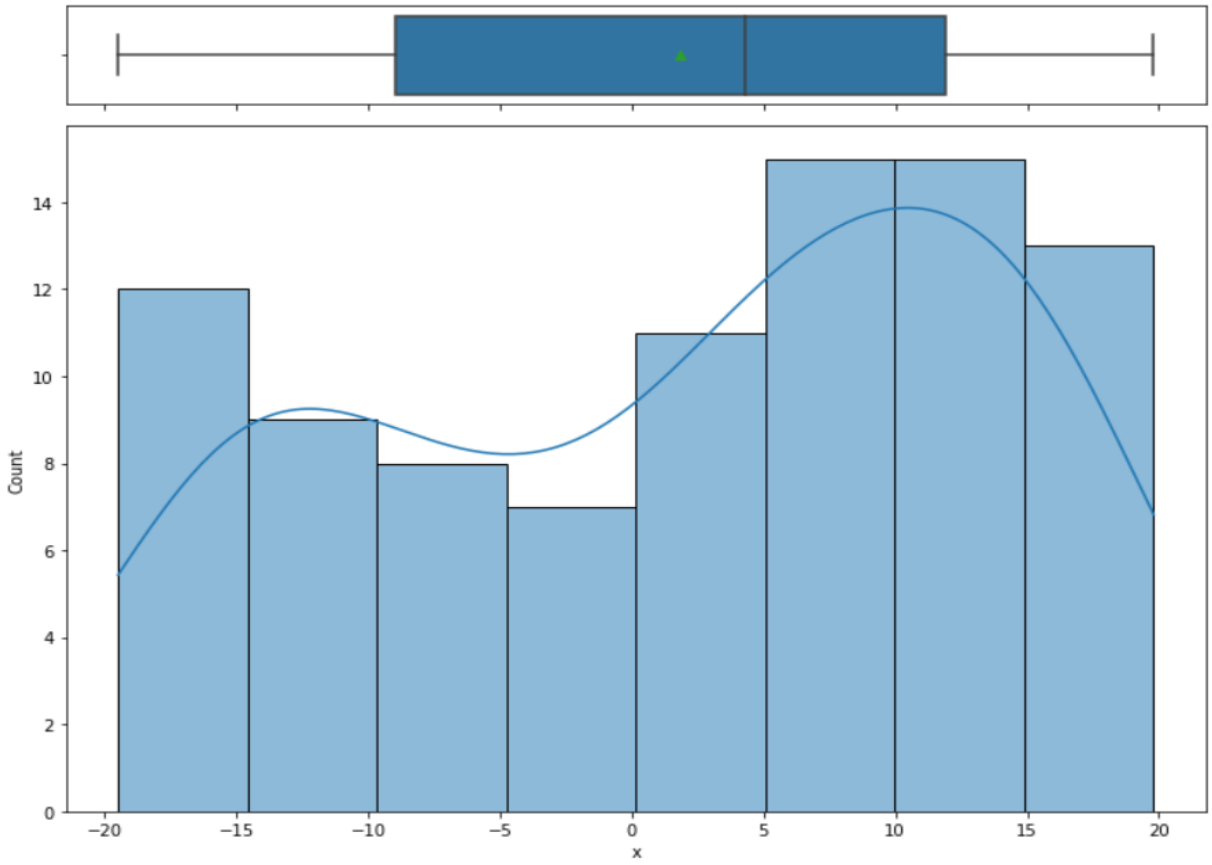


Figure 20: Box plot for test data 'x' variable

```
# Visualize hist_box for 'y' variable in test dataset
hist_box(test_data, 'y')
```

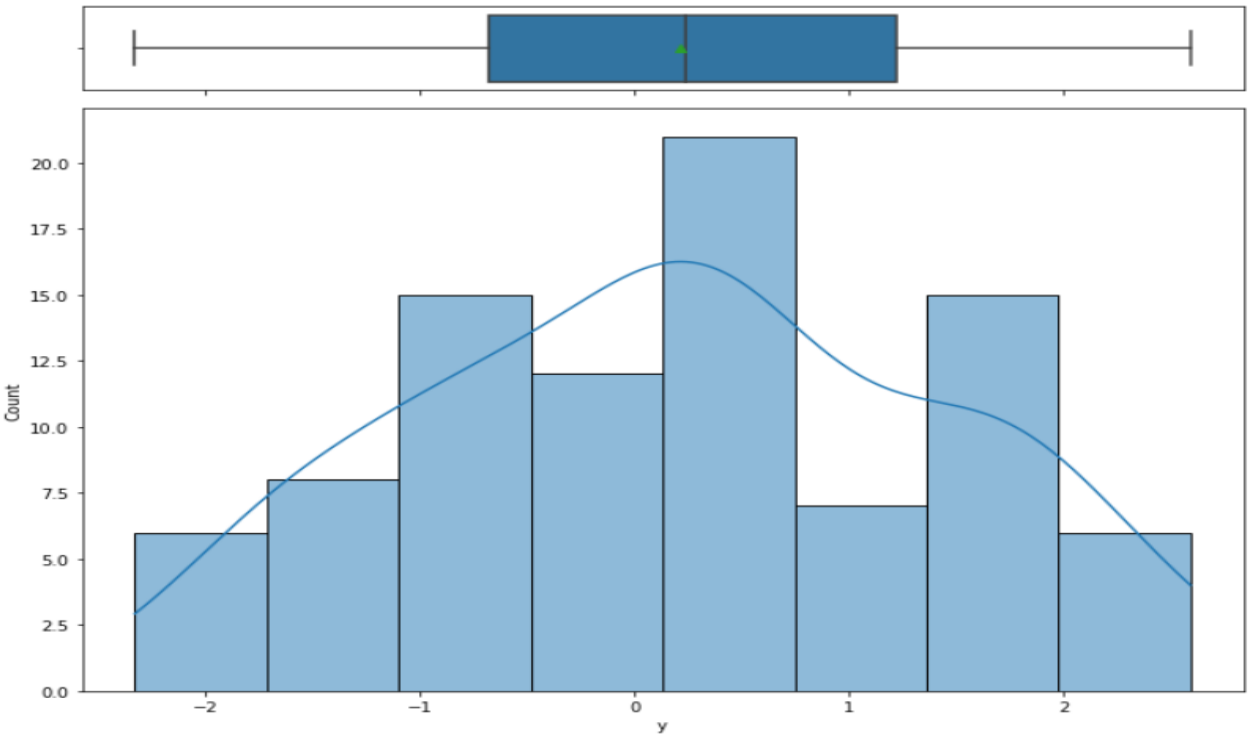


Figure 21: Box plot for test data 'y' variable

```
# Single graph for test dataset considering a pair of variables at a time
sns.pairplot(test_data)
plt.suptitle("Pairwise Relationships in Test Dataset", y=1.02)
```

Figure 22: Graph of pairwise relationship in test data