Exploratory Data Analysis and Multivariate Regression with Object Orientated Programming

Adrian C. Zevenster ADRIAN.ZEVENSTER@IUBH.DE
Department of Artificial Intelligence
International University of Applied Sciences

Contents

List of Diagrams

List of Tables

Abstract

The object orientated paradigm has allowed for quick and efficient data analysis with *data scientist* writing scripts using libraries that uses object orientated design. OOP allows for readability and usability of programs. Relieving the load on the analysis codesbase. Computational notebooks have become the predominant means of analysis preferred by many analysts. Exploratory data analytics(EDA) is usually incorporated in the analysis process, which checks for trends and patterns in data, often overlooked or hidden from the normal analytical process.

Keywords: Object orientated programming, Exploratory Data Analysis, Preprocessing, Test Driven Development, Unified Modelling Language

1. Introduction

Object Orientated Programming(OOP) - also referred to as Object Orientated Design(OOD) - has been preferred by many developers since it's advent over *'Algorithmic decomposition'*, the key difference between the two: algorithmic decomposition keeps the data and the operations on said data separate; where with object orientated design, the data and the operations performed are merged. Data Scientist and Developers often use computational notebooks, leading to code that is often repeated and rehashed, resulting in chaotic source code.(Quillin, 2001)

In this project it is proposed to use a OOP approach to perform: *Exploratory Data Analysis(EDA)*, and build a regressiong model on separate datasets, including *unittesting* touching on *Test Driven Development(TDD)* approaches to improve early error detection in datasets and class interactions. Govindaraj (2015)

The approach is to use 3 separate datasets: training data(A), testing data(B), and ideal data(C). Dataset (A) contains 4 ideal functions, where dataset(C) contains 50 ideal function. Using an OOP approach, the 4 ideal function from dataset(A) that best suite dataset(C) will be determined. Furthermore, dataset(B) will be used to determine to which ideal function the datapoint can be mapped to. Throughout this project we will implement EDA principals: preprocessing, data transformations, and visualization. The conjunction of visual and numerical interpretation of model results will be investigated briefly. The emphasis is on creating classes and methods that allows for experimentation and reuse-ability to explore data in a robust manner. With further elaboration on creating *unittest* to explore the utility of *test driven development* in creating interperatible and reliable models.

After the introduction, the first section 2; discusses the classes, and class interactions that is used throughout this program, this also includes the utility of the represented classes and their associated methods. The 1st task determines the ideal function; obtained from dataset(A) and dataset(C), the 2nd task maps dataset(B) to the ideal function obtained in the 1st task. The methods associated to the aforementioned tasks are discussed in section 3.

In section 4, we will discuss the results obtained from the ideal function and test mapping operations, along with a visual interpretation of the model results. We will also look at the EDA that was performed during the data curation process. EDA is invaluable in analysis of trends and pattern recognition but is often an iterative process, resulting in many common steps should be standardized.Fandango (2017)

The next section 5, argues topics presented in the paper, and suggests possible improvements. The final section 6 provides potential conclusion on the model outcome, and an assessment on suggested methods and attribute operations. Substantiating whether OOP paradigm provides benefits over computational notebook analysis. This project does not aim to create the **optimal** ideal function, the ideal function uses *least square error* to minimize the deviation between dataset(A) and dataset(C), with the test mapping based on a multiplication factor. EDA that is relevant to the presented dataset is incorporated - removing null values, datatype conversion - advanced preprocessing concepts including principal component analysis, missing value imputation methods, are not implemented but discussed. Visualization is not implemented as a model evaluation method, instead as a model interpretation and exploration tool, this tool is discussed in later sections 4.4.2, 5.4. *All application code is available in GitHub (see appendix B).*

2. Design and Structure

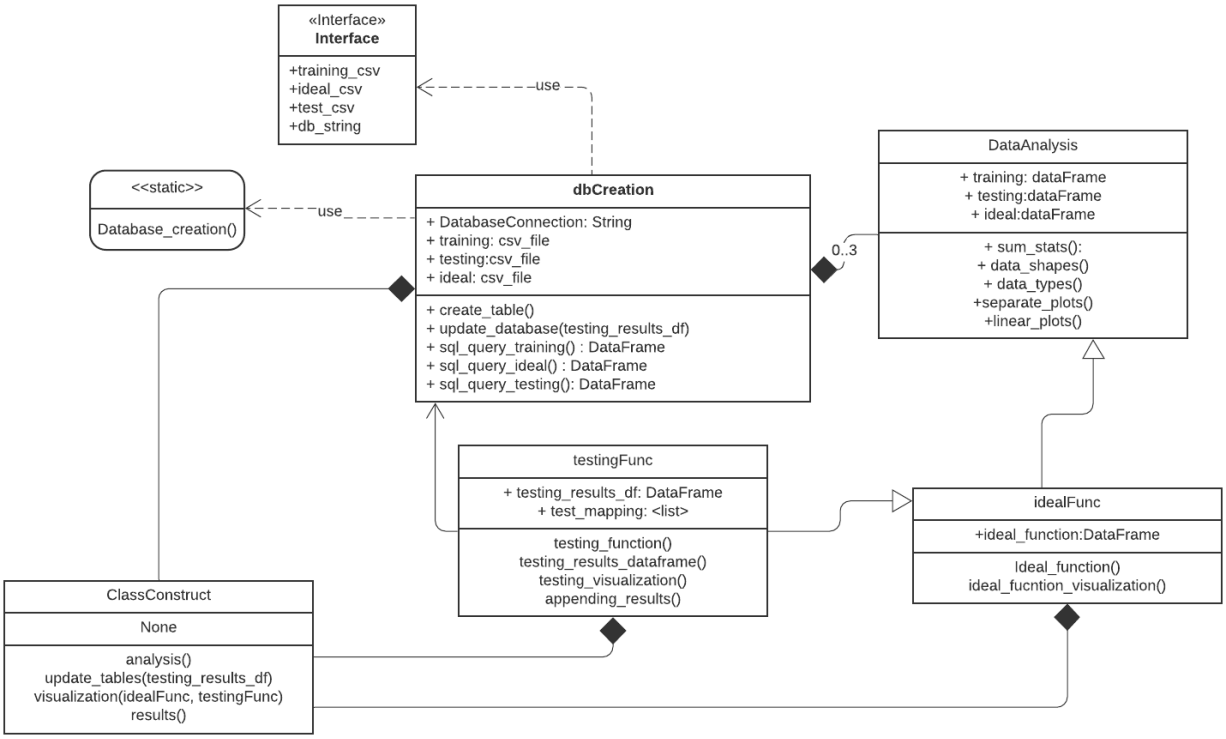


Figure 1: UML diagram representing class interactions

As the rational of this project is to create a program in the OOP paradigm: where inheritance and encapsulation is paramount. This section will unpack how classes and their components interact to achieve the stated tasks within the Introduction1. Modules will contain classes, and methods for: importing data, creating the database, storing of datasets, finding the ideal function - 1st task - and mapping the testing data to the ideal function - 2nd task. The class interaction are represented with the aid of the *Unified Modelling language* (UML), seen in figure 1. With a reference to the UML diagram, description of classes, and class methods are given; along with brief descriptions.

2.1 dbCreate

Before the construction of the class responsible for the database creation, variables defining the directory path and csv files required for the project is prompted, considered as the interface of the project. Allowing training(A), testing(B), ideal(C) data to be interchangeable, important for the 'automatic' nature of functions to be executed on any data being presented.
The first class, `dbCreate`, has a "non-decorative" `@staticmethod` called `database_creation()`, responsible for creating the database to store the mentioned datasets, and the necessary results from operation to be performed in later classes 2.3, for this a *sqlite* database is initiated - called *PwP_ task*. The decorator that first reads in all 3 of the required dataset is specified, this is achieved by using *exception handling*, that raises an *IOError*, checking for the specified csv files - which are then initiated as the attributes of the class, stored as *pandas dataframes*, and the *create_ engine* function from *sqlalchemy* is used to create the database connection. Stepanek (2020)
The 2nd method, `create_tables()`, creates tables that stores dataset (A) and (C): stored as *trainingTable*, and *idealTable*. The methods: `sql_query_training(), sql_query_ideal(), sql_query_testing()` is used as the sql query interface, allowing for queries to all sql tables. All sql methods has a <str>

variable, containing the statement to be executed to the respected table. The method `update_tables()`, which appends results obtained for the Test Mapping and is stored under *TestingTable* 3.2, this method will be an associative method from another class 2.3.
We are initiating these calls to ensure reliability, integrity and maintainability within the structure of our code, decreasing the likelihood that corruption in the data might occur, this also creates a safe environment to understand the data and data interactions Kleppmann (2017). The application code for this module has been included in the appendix C. All data being stored are in the form of key value pairs - in this case, pandas dataframes.
Source code pertaining to this module is included in appendix C

2.2 Ideal Module

The next class main aim is to identify the ideal function from dataset(A) and dataset(C) - 1st task. This module will contain methods for visualization of the individual datasets along with some exploratory data analysis for better data insight. Apart from these operations; methods that will be included will allow for the determination of the ideal function, the chosen method for the ideal function will be discussed in section 3.1 with the results from the ideal function in section 4.2, this section is to illustrate class interaction present in this module.

The *IdealFunction* module will consist of 2 classes:
The first class - `dataAnalysis` ; is a composition of the *dbCreate* class, that retrieves the training and ideal data - using their respective sql query methods mentioned in `dbCreate` 2.1, the testing data(C) is also called as an attribute from the same class. Although the testing data is not needed to determine the ideal function, it is important to ensure preprocessing is done on the data when used in later classes. This class performs exploratory data analysis: summary statics (count, mean, max, min), data types, and data shapes. Methods to perform visualization on both the training and ideal *sql* queries are performed, mention in section3. From here preprocessing based on the exploratory analysis is performed, only basic preprocessing is implemented, including data-type conversions, dropping of possible unwanted components, such as 'null values' or repeated indices, elaborated in section 4.1. This is an important step in curating data for any model, to ensure reliable model prediction, removing noise from datasets - *data conversion, null values* - allowing for models that are more interpretable. All only preliminary preprocessing is incorporated in the program, further preprocessing will be discussed in later sections 5. Balusamy et al. (2021)
The final two methods present in this class, `separate_plot()` and `linear_plots`, provides visualization of the data present in datasets (A) and (C), these plots, and their implementation will be discussed in section 4.4.

The second class - `IdealFunc`, is an inherited class (child class) of the *dataAanlysis* parent class, as indicated in figure 1. From which, the training and ideal data in inherited - along with the necessary transformation performed on the attributes. This class will contain the method - `ideal_function()` - to determine the ideal function, the operations performed in this method and object associated to this method will discussed in section 3.1, seen in figure 1. The second method associated to this class - `ideal_function_visualization()` - creates a logical visualization of the results obtained in the `ideal_function()` method. The operations performed for this visualization and the visualization interpretation will be considered in later sections4.4.2. The results obtained from the ideal function method is stored as attribute.
Source code for this module in included in the appendix D

2.3 Testing Module

The third module - `testModule.py` - contains the class `testingFunc`, inheriting from the `IdealFunc` class, as shown in figure 1, responsible for mapping the testing(B) data to the ideal function, and visualizing the test mapping results; the criteria for the mapping of the testing data will be discussed in a later section 3.2. Inheritance is preferred over composition as the class uses the testing(B) data associated to the `dbCreate` class to perform the test mapping, an the visualization of the test mapping results, needing the ideal function results stored as an attribute from the `IdealFunc` class .Quillin (2001)
The method `testing_function()`, will determine to which ideal function the dataset(B) should be mapped to - method discussed in section 3.2. The second method, `testing_results_dataframe()` will be responsible for creating a dataFrame from the results of the `testing_function()` method, initially stored as a numpy list, the results for the test mapping is stored as an attribute of the class - `self.test_results`. The next method - `testing_visualization()` - will illustrate to which ideal function the testing data is mapped to visually. Results and visualization of the results will be consider in section 4.
This class also has an associative relationship to the `dbModule`, where the `update_tables()` method is used to append results to the *"testingTable"* in the database. The last method `appending_results()`, uses the associative relationship and appends the `self.test_results` to the database using the method mentioned in the `dbCreate` class. This will also allow for the `sql_query_testing()` method to be executed, where the results for the testing mapping can be queried and explored safely without altering results, ensuring reliability. Kleppmann (2017)
Source code for this module in included in the appendix E

2.4 ClassConstruct

This module contains the class that initiates all classes and methods that are relevant to the tasks mentioned in the Introduction1.This primarily includes the classes and modules appropriated for: data analysis,finding the ideal function, and the test mapping. The class objects are initialized with the following naming convention: «ModuleName»_«ClassName», «ModuleName» indicates from which module is being initialized, with the «ClassName» indicating which class object is being called from it's associated module. All of the class attribute are initialized within the `__init__` constructor of the `functionalDataAnalysis` class.
Included in the class are the methods that are responsible for calling the relevant methods from their associated classes. The following methods are all contained inside the `functionalDataAnalysis` class:

- `analysis()` - calls relevant methods from the `dataAnalysis` class 1.
- `results()` - return dataframes for both the ideal function and test mapping results.1
- `visualization()` - provides visualization for the ideal function, test mapping and exploratory data visualization.

The mentioned methods contain variables that calls the methods associated to that class, using the attribute declared in the constructor, the naming convention is as follow :
«ModuleName»_«ClassName»_«method_name». Application code for this model is shown in section F

2.5 Testing

Testing is an integral part in the development of application code, during testing we ensure that individual units and/or components are running as expected. This ensures that the programs can run optimally, with minimal bugs or hindrances. In this project we attempt to employ the methodology by performing some basic testing on components within each class of the project. Kapil (2019)

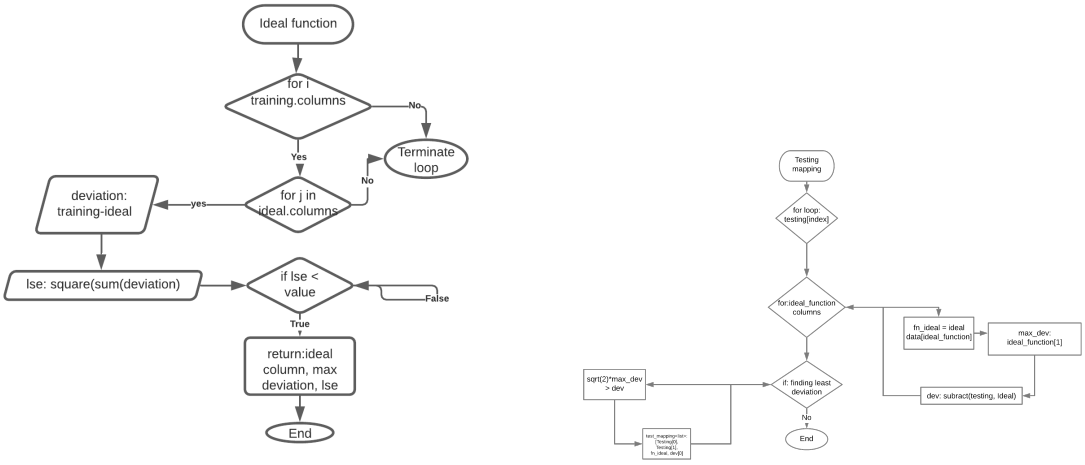
It is suggested that unittests should correspond to their respective class name; represented as test_«ClassName»; however throughout this project we assign the *unittest* module to its corresponding module name, represented as test_«moduleName»; as some modules contain more than one class, providing a more succinct solution.
In the spirit of *"Test Driven Development"*(TDD), an unit test will be included that will determine whether preprocessing/transformation of data should be considered - testing for null values - this includes a *test* function incorporated into the class construction that is checked in the unit test, instead of writing the test separately, the utility of this consideration is discussed in section 4.1

The first unittest is `test_idealModule`, here a test is run on the database class - `TestdbCreate` - ensuring that a connection has been made to the database, once this has been checked the second class in the test module is run.

The second test is performed on the ideal function class 2.2, called `TestidealFunction`: the first test ensure that there are no null values present in the `dataAnalysis` class (TDD) - this serves as the prompt to performing cleaning on the dataset should the test not pass, the second test checks if the data types present in the ideal(C) and training(A) data is the same, prompting further preprocessing should the test be unsuccessful. The third test ensures that the training(A) data columns are the same in the ideal function, effectively checking that functions are not repeated. The final test checks whether the ideal function contains the expected length - this is based on the number of functions present in the training data.

`test_testModule` is the second unittest module. This testing module operates solely on the `testingFunc` class. Performing 4 test in total: the first test checks for an instance of the test mapping results 2.3 ensuring a dataframe, the second test ensures that the shape of the test(B) data is as expected ,the 3rd test checks whether the specified column names are present - this also ensures that the test(B) data has not been replicated when checking for the instance, the final test ensures that a string is present in the 3rd column(second index), and not a integer value, which is needed to point to the ideal(C) data value from the ideal function.
All unit test have been include in the Appendix G, H

3. Methods



(a) Ideal Function Flow Diagram (b) Test Mapping Flow Diagram

Figure 2: Approach for creating Ideal Function and Test Mapping functions

Interaction of different classes and objects has been mentioned in the previous section. The following section will address the means of determining the ideal function; an explanation on the `ideal_function()` method, within the `IdealFunc` class, along with a logical analysis of visualization results for the ideal function. Furthermore, the process for mapping the testing data to the ideal function is analyzed methodically; along with the visual representation of the mapping results.

3.1 Ideal Function

The purpose of the *ideal function*, is to identify 4 functions from dataset(C) - containing 50 functions - that matches the function from dataset(A). The criteria for achieving this task is choosing the values that minimizes the y-deviations, which is essentially the *least-squares* approach. In general the representation of least-squares cam be expressed as:

$$E(a,b) = \sum_{n=1}^{N} (y_n - (ax_n + b))^2$$

which reduces the error between the predicted value and actual value is reduced.Miller (2006). In our case we will determine the functions that reduces the error by minimizing the deviation between the y- values present in dataset(C) in regards to the functions provided by dataset(A); achieved by comparing each entry in the column, 1→50, of the ideal(C) set to each entry in the column 1→5 from the training(A) set - which represents different function - returning those values with least deviation. Due to to fact that some of the values will have higher errors; thus be penalized more, the absolute values can be used. Furthermore, values with large negative or positive values, penalizes high errors more than lower errors, the absolute deviation is considered to mitigate this pitfall. Kerzel (2020)

To find the ideal function we find the deviation between the two datasets mentioned, after this has been done the function that minimizes the y-deviation is identified using the *sum of the squares*. The process for these operations are as follows

Deviation : To find the values that minimizes the the y-deviation, the sum over the actual values minus potential value (which could be considered as the predicted values), needs to be determined and compared for each potential ideal function, this is achieved by first by determining the deviation between functions, illustrate by equation 1:

$$\sigma(I) = \frac{1}{N} \sum_{n=1}^{N} |(A_n - C_n)| \tag{1}$$

where:

$$A_n = 1, ..., A_n \quad and \quad C_n = 1, ..., C_n$$

$\sigma(I)$ shows the deviation between the datasets, N indicates the number of observations, where A_{ij} indicating the number of iterations over the training data(A) and C_{ij} the iteration over the ideal data(C). After the deviation is calculated, it is used to calculate the sum of the squares is calculated:

$$TSS(A,C) = \sum_{n=1}^{N} |(A_n - C_n)^2| \tag{2}$$

While squaring, larger errors carriers a higher weight - due to large deviating values, improperly emphasises unwanted errors, the absolute squared values is used the mitigate this *bias*.Gonzalez Zelaya (2019)

The method `ideal_function()` - within the `IdealFunc` 2.2 class - is used to calculated the ideal function from dataset (A) and (C). To achieve this a *nested for loop* is utilized, shown in flowchart 2a. The first for loop iterates columns in dataset(A), with the first column omitted; mentioned in section 4.4.1. A

variable is then initialized that will act a storing variable for the least square error, `least_square_error`, the second for loop iterated over dataset(C) - allowing for the deviation calculation in equation 1, stored as `dev`. Another variable, `lse`, is used to calculate sum of the square of the deviation. An `if` statement is used to check for the values with the optimal least square. Once the optimal value has been obtained the results is written as tuple containing the columns from dataset(A) as index value; with the chosen ideal column name from dataset(C), the maximum deviation in the second index, and the least square error value on the third index. This process it illustrated in diagram 2a.

3.2 Test mapping

This section focuses on the analysis of the mapping criteria an process, this pertains to method `testing_function()`, mentioned in section 2.3. The test mapping is achieved by nested for loops, showing in figure 2b.
In the first loop there is an iteration over the `range()` of the testing data(B), there can be N associated number of functions to the testing data. The second iteration is over the ideal function, obtained from `ideal_function()` method. There are 3 variables that are created inside the nested loops:

fn_ideal: Variable responsible for finding the ideal data(C) function that has been identified by the ideal function 3.1. Extracting the column values pointing to the ideal data(C) functions.

max_dev: Storing variable $\sigma_{max}(T)$, from ideal function tuple 2.2. The σ_{max} from the ideal function will be compared to the calculated in test mapping.(see eq.3)

dev: This variable is responsible for finding the deviation between the ideal function and the testing data. The for loops allow for each instance in the testing data(C) to be compared to each instance in the ideal function - that correspond to the ideal data(C); the testing data(B) can be represented as T_t, with the ideal function as I_t. The deviation between these two instances is then calculated in the similar approach as the ideal function(1), The only difference the number of observation is excluded(cite reason for exclusion), from this we obtain:

$$\sigma(T) = \sum_{t=1}^{N} |T_t - I_t|$$

.

The final operation is to determine the test mapping; this is done by comparing the the maximum deviation of the ideal function - `dev_max` - to the calculated deviation $\sigma(T)$ represented in `dev`, a point is successfully map if the deviation $\sigma(I)$ multiplied by a factor of $\sqrt{(x)^2}$ - where x shows the deviation variable - does not exceed the maximum deviation $\sigma(T)$:(need to add a reason for the multiplication of this factor; also need to add why the length is not used for the deviation calculation)

$$T_{map} : \sqrt{\sigma I_{max}^2} > T_{dev} \tag{3}$$

Equation 3, shows the criteria for mapping the test data(C) to the ideal function

If the criteria is satisfied, a list is generated containing the results of the point from the testing data(B) that is successfully mapped to the ideal function: where the first index value is the testing data(B) x-values, with the y-value in the second index, the third index contains the ideal data function(column) to which the point has been mapped, along with the calculated deviation $\sigma(T)$ in the fourth index. Which is then converted to a *dataFrame* in the `testing_results_dataframe()` method of the class, as depicted in section 2.3. The application code responsible for performing this operation has been include in the appendix E

4. Analysis

This section main objective is the analysis of the ideal function 3.1 obtained, along with the analysis on the test mapping result 3.2 - this includes analysis on the visualization of the ideal function results,and the testing function results.

4.1 Exploratory data analysis

The exploratory data analysis(EDA) allowed for better conceptualization of the data used in finding the idea function, assisting especially in illustrating the results from task 1 and task 2 with a visual interpretation. The visualization performed assisted in pinpointing areas where noise might be present during the initial visualization 4.4.1. Allowing for more concise method of finding the ideal function; such as using absolute deviation to handle highly negative or positive values present in data.(citation needed)
EDA also indicated what preprocessing could potentially benefit the model, such as reshaping data if needed and the transformation of datatypes, that is essential for continuity with certain packages A. Another consideration would be to *reduce dimensionality*, the data in the ideal set is of 'high dimensionality' due to the presence of 50 function. If the unnecessary function is eliminated, model results could be improved and test mapping. EDA also identified where further preprocessing could be warranted: normalization could be used to account for the multiplicity among functions and *Principal Component Analysis* can assist in eliminating uncorrelated functions.

The variability in statistical interpretation, as show in the maximum deviation observation made - illustrates the importance of incorporating as many statistical inference methods as possible as using only certain measures can skew interpretations towards a desired conclusion.

4.2 Ideal Function

	y1	y2	y3	y4
0	y49	y39	y40	y4
1	0.496396	0.499521	0.4975	0.499249
2	0.0002	0.000207	0.000202	0.000189

Table 1: Ideal function results

The results of the tuple from the ideal function can be seen in table 1. From inspection the maximum deviation and least square error for all four of the ideal functions are quite similar. Suggesting that there are no severe outlying data in any of the ideal functions, or at the least, the deviation of outlying points are similar in all ideal functions. Ideal function y1 and y4 has the lowest least square value, indicating that they deviate the least from the mean, suggesting better fit - although the maximum deviation is comparable in the other cases. The least square value for all 4 functions is close to 0.5, which show that the model should be generalizing well, avoiding cases of *overfitting* - not capturing enough complexity, and *underfitting* - capturing too much complexity.Cunningham and Delany (2020)

4.3 Test mapping

The results obtained for the test mapping, mentioned in section 3.2, shows which points satisfy the criteria stipulated in equation 3. In the original testing dataset(B) provided by the csv file, 1600 observations are present; whereas in the tuple containing the Test Mapping results, 417 observations are present, concluding that 417 point met the mapping criteria, with 1,183 observation outside the mapping criteria, implying that missing values ϵ ideal function. The results for the Test mapping in include in the appendix I.

From the testing results: observation are in the majority of cases mapped to Ideal function $y49$, with some instances mapped to $y39$, correspond the training data(A) functions $y1$ and $y2$. These results have been appended to the database mention in section 2.1 To enquire further on these results, the method `sql_query_testing()` can be used, where the `sql_testing_statement()` string can be modified. The visualization of the test mapping results is discussed further in the next subsection. 4.4.

4.4 Visualization

The following section describes the visual component of the *Exploratory Data Analysis* 4.4.1 performed. Alongside visual representation and discussion on the ideal function obtain 4.2 with an analysis on the test mapping results 5.

4.4.1 Initial Visualization

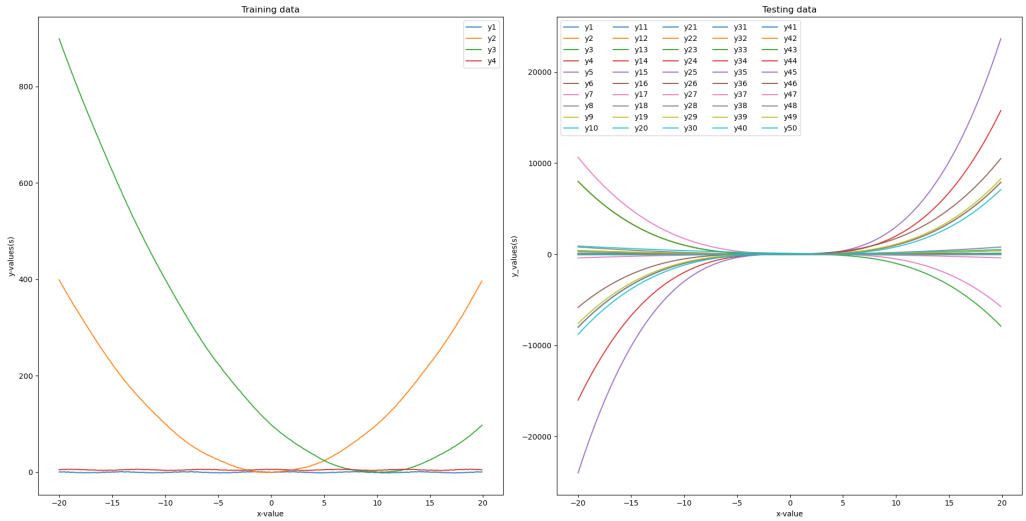
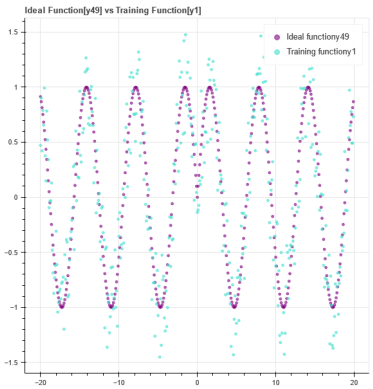


Figure 3: Visual illustration of testing and ideal data

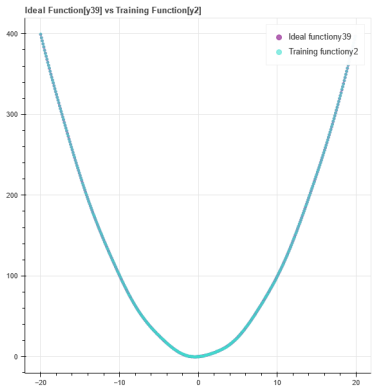
Figure 3: Graphs for training(A) and ideal data(C), with the first[index] column on the x-axis. Functions are presented in the graph legend.

The preliminary visualization explores data correlation intuitively. In the comparison of dataset (A) & (C) - represented in figure 3 (created by the `separate_plots()` method), the *Training Data* title shows the functions in datasets(A), effectively depicting the functions expected from the *ideal function*, with the *Testing Data* title illustrating the functions present in dataset(C). The important disparity in the training data image show is that 2 of the functions that seem linear from the training data indeed are not linear; they are sinusoidal in their representation. The function represented in the Testing data appear to be power functions, the ideal function reveals the fallacy, obscurities can be better conceptualized using the *pairplot* from seaborne, however, for larger datasets - such as ideal dataset(C) - pairplot is not feasible. Whereas the method `linear_plots()`, gives an superimposed representation of figure 3, which relates to what we are attempting to achieve with the ideal function - finding what functions(columns) from the training data(A) matches which functions(columns) from the ideal data. From the training data visualization, we can observe that we are dealing with 4 types of function: sinusoidal, hyperbola and a descending exponential, thus we should expect four ideal function to contain a similar representation, as as approximate representation as possible.

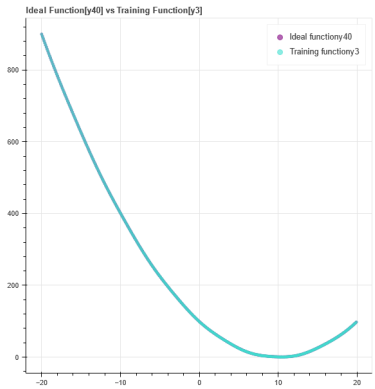
4.4.2 Ideal function visualization



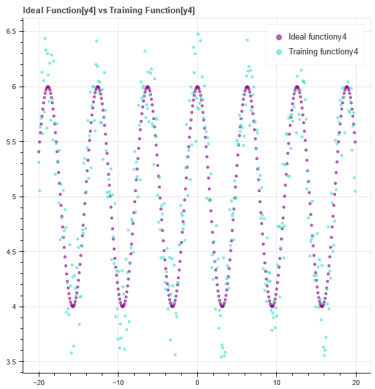
(a) Training[$y1$] vs Ideal[49]



(b) Training[$y2$] vs Ideal[39]



(c) training[$y3$] vs Ideal[40]



(d) Training[$y4$] vs Ideal[$4y$]

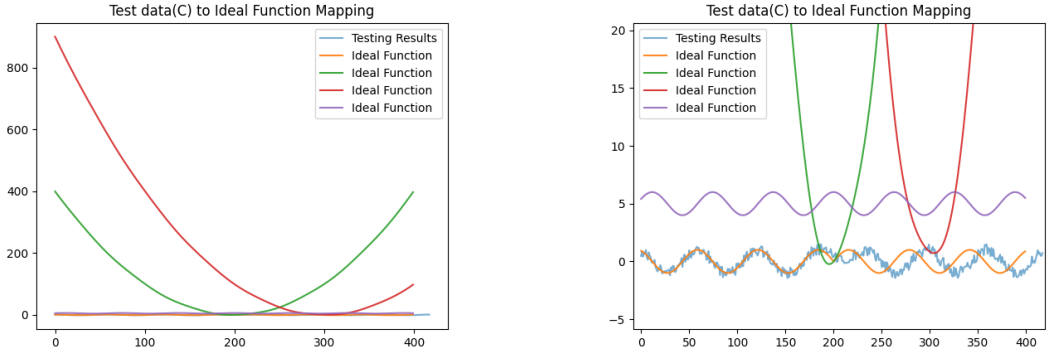
Figure 4: Ideal function visual representation

Figure 4 represents the training data(A) that is conterminous to ideal data(C); identified by the ideal function. The legend of the sub-figures in figure 4 indicates which columns from the ideal set(C) is formfitting to the training set(A). In the case of image(s) 4b, 4c, the training and ideal function seem homogeneous - smaller deviation are to be expected; however could be indicative of *overfitting* in the model. During the preparation of the ideal data visualization, a for loop is initiated. This for loop iterates over training data(A) columns - excluding the first index, as mention in section 4.4.1. A variable is then initiated that stores the index from the ideal function pointing to the represented ideal data. Two scatter plots are then superimposed, one containing the training data functions, which are the training data(A) functions from image 3; the second plot uses the ideal data(C) with the ideal function variable passed in. From images 4b, 4c, we observe an almost identical fit to the parabola and descending exponential function from the training data. Here we can expect a higher least squared error in comparison the the other 2 ideal function Miller (2006), however as noted from the Ideal function results in table 1, all ideal functoin have very similar maximum deviations and similar TSS values. The ideal function visualization in images 4a, 4b give a more detailed representation from image 3, revealing the sinusoidal of function $y1$ and $y4$ - which could previously be interpreted as a straight line, here the 'noise' present in the training(A) data is revealed.

Duly what this indicates: the identified ideal function is able to fit the data on more than just linear cases, but also on multivariate cases and polynomial cases, as is the case in both images 4b and 4c, which means we should not be limited to linear cases.

For the ideal function visualization, the *bokeh* library was utilized, bokeh provides a richer visualization report than *matplotlib*. In the Ideal function visualization, the legend can be filiterd to represent either the training(A) data or the ideal(C) data function. See section 5.3 for further more comments on visualization methods

4.4.3 Test mapping visualization



(a) Test mapping to ideal function

(b) Test mapping enhanced

Figure 5: Test mapping results

As indicated in section 4.3, only a fraction of the available datapoints from the test data(B) satisfies the mapping criteria, these point are depicted in figure 5. The function that are indeed mapped is indicated in a superimposed manner over the ideal function data, the points that are successfully mapped to an ideal functions are indicated as 'testing results' in the graph legends, a further enhanced image 5b is shown, which shows a 'zoomed' in image of the testing data points and their proximity to the ideal function, observations not meeting the criteria are excluded.
This presentation is obtained from the `testing_visualization()` method, with the *enhanced* image obtained by zooming into the presented figure. The source code for this visualization is presented in accompanying `testModule` in the appendix E. This visualization can be initialized from the `testing_visualization()` method or from the `ClassConstruct` module that contains all visualizations.

5. Discussion

This section looks at discussing the performance and utility of the components suggested throughout the article and whether there is benefit to be added. Discussion on possible improvement in some cases are suggested.

5.1 OOP Data Analysis

During this project one of the aim was to create a framework that allows for usage of different components in the model that allows for experimentation and interactivity with the model. Along with creating a methodical and repeatable regression process, that allows for testing and analysis on the the methods. Critical errors are tested for that could obscure model results and performance. Visualization allowed for better model interpretability. Both Python libraries Matplotlib and Bokeh was used for Visualization. With bokeh used for the visualization of the ideal function, however through class composition interaction, the bokeh library resulted in inconsistent results, further visualizations were all performed using *matplotlib*, achieving more consistent results.
Further consideration would be to improve the graphical interface to the user - at the risk of concealing the underlying mechanics. Along with adding interactivity on the results presented. This was

considered with the visualization of the ideal function. `ideal_function_visualization()`. However, the repeatability of methods associated to the visualization is only extented to the representation in images 3, 4.4.2, the visualization of the test mapping 5 is specific the the columns index of the ideal(C) dataset. This can be adjusted to include for more generalized cases but was omitted during this project.

The OOD paradigm allowed for results to be easily stored and maintained in a *sqllite* or *mySQL* database, although appending the test mapping results through composition was more difficult to achieve with the latter database. This approached also allowed for consistent queries and results being stored for custom interpretations through simple sql queries. Should different features or indices be required from the stored results it can be done with ease and simplicity. Using Object-Orientated databases removes the need for API tools, resulting in faster queries and generally better performance. Sedighi (1993)

5.2 Test Driven Development

Unit testing was included to ensure some requirement are satisfied throughout the life cycle of the program, considering using test driven development as a framework. Most test in this project are incorporated as test on the classes and attributes already performed, initial unit test assist with indentification of null values, this could be important in data sets that have a large amount of missing values, removing missing values categorized to the potential cause of missing values could be additive to the model, thus before performing general cleaning of null values through a method, unit test could indicate what type of imputation methods can be considered with the cleaning of null or missing values. Test to to reduce the likelihood of overfitting and underfitting could be included by testing for expected model coefficients or minimum values, and test that consider bias and noise in the data, like outlying data points or repeated observations.Cunningham and Delany (2020).

Unit testing could further be considered in merging of different branches when projects need collaboration, for instance, unit test ensuring that certain requirement are met before mergers could be incorporated to detect errors that could arise promptly. Unit test could further be used that catch any egregious errors whilst merging branches, requiring less time reviewing each merge request. Incorporating integration tools in GitHub such as *codecov* - indicating the portion of the codebase that is being tested, ensuring more reliable code. This is unfortunately not inspected in the body of this project.cod

5.3 Preprocessing and Exploratory Data Analysis

While we considered some preliminary data cleaning operations - including checking for missing values, converting data types and transforming data shapes - to better work with the presented data. There are various preprocessing components to consider that could potentially improve model performance, that is not already implicitly used such as the *absolute deviation*, than considers values that might have extremities in positive or negative values. Standardizing data; the processing of scaling values to an equal range could assist with better predictions.

Although there is some noise in - specifically in the training data(A), the data considered is very regular, determining whether the cause of noise is attribute noise or class noise could be instructive on handling issues presented by the data. Class noise is usually introduced by noisy training(A) data, whereas attribute noise, where cases that perturb the model like nature of missing values is usually due to attribute noise, creating a more sophisticated model that caters for more complexities could greatly benefit from this approach. Zhu and Wu (2004) Further consideration such as dealing with missing values could further assist model performance. If a given dataset has a lot of missing values; it

is worth-wile to consider how this case can be handled apart from simply dropping the missing values. The nature of missing data values, such as *Missing Completely(MCAR* or *Missing at Random(MAR*, has disparate effects on model performance, improving model performance if considered. adequately. van Buuren (2018)

The visualization presented in section 4, helps to explore data correlation effectively. The visualization indicates a stronger correlation than those represented by die ideal function results presented in table 1, shows a more coequal least squares(TSS) as the visualization would intuitively suggest. Although the visualization component assist substantially in early data interpretation, large scale visualization often obscures underlying patters, as made evident by figure 3, where in higher dimensionality datasets, meaningful information becomes difficult to extract, and nuance are often hidden, comparing the training(A) data representation in figure 3 and in figure 4.4.2 makes this stark.

5.4 Model Performance

From the visual analysis 4 it could be concluded that over-fitting might be present, which severely effects model performance. Over-fitting occurs when the model fits the training data too precisely, failing to generalize the model; leading to bias and unwanted predictions.(citation needed). Looking at methods to prevent/improve overfitting or underfitting could assist the model, these methods could include cross validation or regulating data splits, such as the ideal(C) data and training(A) data(citation). From the visualization and ideal function results presented, the visualization at times seems contradictory to the data represented, ideal function $y2$ and $y3$ could indicate overfitting, but results table 1 the opposite is suggested. Ideal function $y1$ and $y4$ visual representation gives the impression that the average, and also maximum deviation would be much higher, all maximum deviation values minimal deviation. Further preprocessing can also be considered than merely removing missing values. Maximum likelihood and multiple imputation could even be considered to mitigate the effect of missing on model performance. Myung.
However, what the model results conclude, was that the model was able to capture more than just linear cases, as evident by the identification of the descending exponential and hyperbola. Furthermore, it was able to capture nuances in seemingly linear cases and correctly identified them as sinusoidal. Suggesting generalization in model performance - with reuseability on novel cases.

6. Conclusion

Throughout this project, the focus on creating a *Object Orientated* model for data analysis (regression analysis). Using concepts such as inheritance and encapsulation. Hemmerich et al. (2021) The process for the regression analysis is performed by calling classes and their associated method, the only requirement is a `docstring` containing the directory path and file names for the raw data, thourgout this framework the aim is to produce classes and methods that are repeatable, this is the case for methods in this program. Methods associated to: data exploration, visualization, preprocessing and data structures transformation is repeatable on unseen cases. We touched on concepts or *Test Driven Development* -incorporating *unittesting*: testing requirement for execution, testing components throughout class interactions, and tests to ensure interpretability of components throughout. Unittesting assist with identifying whether null values are present, this method can be applied to any set of novel datasets, adding too repeatability components of the program.

We further explore *preprocessing* concepts that ensure data is standardized and normalized: removing null values, ensure data shape consistencies, and data type transformation. We also employee visualization for *exploratory data analysis*; assisting in interpretation of functions contained within the data, exploring data correlations - proving that even rudimentary visualization plays in intricate role in working with data. Visualization is also used to illustrate the results from the regression analy-

sis, empirically relaying model output and performance. However this does not necessary serve as a model evaluation method, rather an interpretation of model output. Unit test were only considered on application code level, which did assist with ensuring data consistency throughout class interactions, however test did not go into broader scope, which includes tests for merges and pull request. These are all added 'benefits' of incorporating the OOP diagram over the often preferred computational notebook approach. We also showed that many of the EDA steps can be standardized when performing certain exploratory operations, that can be reused on different datasets - which could benefit steps often repeated such as the visualization process, as done in the visualization of image 3, although the attribute for the visualization had to be specified; which datasets should be visualized, however any visualization specified as dataset(A) and dataset(C) will be performed without intervention. Further improvement can then seamlessly be incorporated and experiments effortlessly performed, as opposed to rewriting the operations for each instance.
Visualization made interpretation of function less straining, but could at times substantiate *confirmation bias*, thus the statistical results should be combined with the visual representation to add a more holistic overview. Kahneman (2011)

A git repository has been created for the project mentioned in the Appendix B, where an overview of the structure is given. Assisting in viewing source code and data files, with instructions on cloning the *develope* branch. All files have been committed with a 'commit message' describing each file with extensive docs string comments. Pull-Request from another branch - *feature* to the develope branch is described. In section 5.2, the integration of tools like *codecov* could benefit the development process greatly by adding the screening pull requests made, identifying syntax error more seamlessly. Applying these concepts, especially as they pertain to unit tests, is difficult to achieve where code is note in the form of functions and classes, further substantiating potential the benefit of the OOP paradigm.

A. Versions and Libraries

This project was created using:*Python* integrated programming language, with *PyCharm* as the integrated development environment - Jupyter was also used for testing purposes, Anaconda as the interpreter. Using the Windows OS. The version(s) are to follow:

- Windows: Microsoft Windows [Version 10.0.19042.1415]
- PyCharm: Version 2020.3.5
- Anaconda: Version 2021.05
- Pyton: Version 3.8.0

The program uses various libraries associated to Python, to following is a table **??** containing the packages used, with a mention to the function used for package, an description on the use of the function, and how the pip install can be performed

Library	Package	Usage	pip install
pandas	Various	Dataframes, preprocessing, sql queries	pip install pandas
bokeh	figure, show, save	Ideal function visualization	pip install bokeh
matplotlib	Various	Other visualizations	pip install matplotlib
sqlalchemy	create_engine	Database:creation, connection	pip install sqlalchemy
numpy	abs, subtract, sqrt	Ideal function and test mapping	pip install numpy
unittest	n/a	Testing of classes	pip install unittest

Table 2: Libraries and Packages

B. GitHub

This appendix gives instruction to viewing the source code for this project on *GitHub*, Instruction for the following aspects of the project is included:

- Git repository containing project files.
- Branch name *'developer'* stored in the project repository
- Pulling and Commit-ing to Remote repository

The Github repository for the project can be found by following the link: **Git repository**. With source code saved under the **developer** branch
Each of the files that has been *commited*, will contain a commit message; specifying what the function of the included files are

The following contains all files within the **developer** branch, with a description on all scripts contained:

- Project _files
  - - train.csv; training(A) dataset
  - - test.csv; testing(B) dataset
  - - ideal.csv; ideal(C) dataset
- Script_files
  - - dbModule.py; source code for class `dbCreate`

- - idealModule.py; source code for classes `dataAnalysis` and `IdealFunc`
- - testingModule.py; source code for class `testFunc`

- Tests

- - test_idealModule.py: all unit tests performed `dbCreate`, `dataAnalysis`, `IdealFunc` classes.
- - test_testingModule: unit tests on `testFunc` class.

Clone Repository: The repository can be cloned with the following <repositry_url> [branch_name]: `https://github.com/adrianzevenster/pwp_project.git develope`

The source branch in this project is called *develope*, pull requests can be made from the *feature* branch located under the repository. *develope* has been specified as the upstream branch, where merges can take place.

Push to branch: Changes can be pushed to the remote branch by using the follwing statement: `git push -u <repo_url>`.

Repository url(repo_url): `https://github.com/adrianzevenster/pwp_project.git`

C. dbModule

```
from sqlalchemy import create_engine
import pandas as pd
import os

"""
Variable initialization that acts as interface:
- directory: Project directory
- user_input_A: Training data(A) file name
- user_input_B: Testing data(B) file name
- user_input_C: Ideal data(C) file name
"""

directory = str("<directory_path>")
user_input_A = str("<file_name>")
user_input_B = str("<file_name>")
user_input_C = str("<file_name>")


class dbCreate:

    # static method to create database
    @staticmethod
    def database_creation(engine=create_engine("sqlite:///PwP_task_IUBH")):
        engine.execute("CREATE DATABASE PwP_Project")
        engine.execute("USE PwP_Project")

    def __init__(self):
        try:
            training_csv = pd.read_csv(os.path.join(directory, user_input_A))
        except IOError as e:
            print(e)
        try:
            testing_csv = pd.read_csv(os.path.join(directory, user_input_B))
        except IOError as e:
            print(e)
        try:
            ideal_csv = pd.read_csv(os.path.join(directory, user_input_C))
```

```
        except IOError as e:
            print(e)

        self.engine = create_engine("sqlite:///PwP_task_IUBH.db")
        self.engine.connect()
        self.training = pd.DataFrame(data=training_csv)
        self.testing = pd.DataFrame(data=testing_csv)
        self.ideal = pd.DataFrame(data=ideal_csv)



    def create_tables(self):
        """Creating tables in Database"""

        ''' training table creation'''
        self.training.to_sql(con=self.engine, name='trainingtable', if_exists='
    replace', index=False)

        '''ideal table creation'''
        self.ideal.to_sql(con=self.engine, name='idealtable', if_exists='replace',
    index=False)

        return 'Databases table creation for: training, ideal data has been created
    successfully'

    def update_tables(self, testing_results):
        """Creating table for test mapping results: <testModule.py"""
        testing_results.to_sql(con=self.engine, name='testingtable', if_exists='
    replace', index_label=False)

    def sql_query_training(self):
        """Training sql query function"""
        # sql query statement variable: <sql_statement_training>
        sql_statement_training = "SELECT * FROM trainingtable"
        training_sql = pd.read_sql_query(sql_statement_training, self.engine)
        return training_sql

    def sql_query_testing(self):
        """Testing sql query function"""
        # sql query statement variable: <sql_statement_testing>
        sql_statement_testing = "SELECT * FROM testingtable"
        testing_sql = pd.read_sql_query(sql_statement_testing, self.engine)
        return testing_sql

    def sql_query_ideal(self):
        """Ideal query function"""
        # sql query statement: <sql_statement_ideal>
        sql_statement_ideal = "SELECT * FROM idealtable"
        ideal_sql = pd.read_sql_query(sql_statement_ideal, self.engine)
        return ideal_sql
```

D. idealModule

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import sys
import dbModule as db
from bokeh.plotting import figure, save, show

"""
```

```
Class for dataAnalysis, methods include data descriptions(1), preprocessing(2), and visualization(3)
- test_null_values: Checks dataset(s) [A, B, C] for null values: unittest (1 & 2)
- sum_stats: Summary statistics on all datasets (1)
- stand_dev: Return standard deviation of column in datasets [A, C] (1)
- data_shapes: Return shapes of all datasets (1)
- data_types: Check data types of all datasets, if type not appropriate: converts to <float> (1 & 2)
- Linear_plots: Visualization of datasets [A, C]
- separate_plots: Visualization of datasets[A, C]
"""


class dataAnalysis:

    def __init__(self):
        """ Class for basic data analysis"""
        db_import = db.dbCreate()
        self.training = db_import.training  # db_import.sql_query_training()
        self.testing = db_import.testing
        self.ideal = db_import.sql_query_ideal()


    def test_null_values(self):
        """TDD for testing null values"""
        training, testing, ideal = self.training, self.testing, self.ideal
        if training is None:
            return training
        elif testing is None:
            return testing
        elif ideal is None:
            return ideal
        else:
            print("Dataset(s) has no null values")

    def sum_stats(self):
        """Summary statistics of Data"""
        return "Summary_statistics:" \
               'Training', self.training.describe(), \
               'Testing', self.testing.describe(), \
               'Ideal', self.ideal.describe()

    def stand_dev(self):
        """Pandas standard deviation function: return <str>"""

        standard_dev = {'training std': self.training.std(), 'ideal std': self.ideal.std()}
        return standard_dev

    def covariance(self):
        return self.training.cov()

    def data_shapes(self):
        return "Data Shapes:", \
               'Training', self.training.shape, \
               'Testing', self.testing.shape, \
               'Ideal', self.ideal.shape

    def data_types(self):
        """Preprocessing: Data type check and conversion: <float>"""
        if type(self.training) != float:
            self.training = self.training.convert_dtypes(float)
```

```
68
69          elif type(self.testing) != float:
70              self.testing = self.testing.convert_dtypes(float)
71          elif type(self.ideal) != float:
72              self.ideal = self.ideal.convert.dtypes(float)
73          else:
74              print("All DataFrames are floats")
75          return self.training, self.testing, self.ideal
76
77      def linear_plots(self):
78          """
79          Data Visualization: Training[A], Ideal[C]
80          - Superimposed representation
81          """
82          fig, ax = plt.subplots(figsize=(10, 4))
83          '''Training data[A] plot'''
84          ax.plot(self.ideal['x'],
85                  self.ideal[[i for i in self.ideal if i != 'x']],
86                  alpha=0.4, )
87          '''Ideal data[C] plot'''
88          ax.plot(self.training['x'],
89                  self.training[[i for i in self.training if i != 'x']],
90                  alpha=0.3,
91                  label=f'{self.training}')
92          ax.legend()
93          plt.xlabel("x")
94          plt.ylabel("y")
95          plt.title("Testing data and Training data")
96          plt.show()
97
98      def separate_plots(self):
99          """
100         Separate Visualization representation
101         - ax1:Training data[A]
102         - ax2:Ideal data[C]
103         """
104         fig, (ax1, ax2) = plt.subplots(1, 2)
105         ax1.plot(self.training['x'],
106                  self.training[[i for i in self.training if i != 'x']],
107                  alpha=0.9)
108         ax2.plot(self.ideal['x'],
109                  self.ideal[[i for i in self.ideal if i != 'x']],
110                  alpha=0.9)
111         ax1.set_title("Training data")
112         ax2.set_title("Testing data")
113         ax1.set_xlabel("x-value")
114         ax1.set_ylabel("y-values(s)")
115         ax1.legend(self.training.columns[1:].values)
116         ax2.set_xlabel("x-value")
117         ax2.set_ylabel("y_values(s)")
118         ax2.legend(self.ideal.columns[1:].values, loc='best', ncol=5)
119         plt.show()
120
121
122 """
123 Class to determine ideal function <-- dataAnalysis
124 """
125
126
127 class IdealFunc(dataAnalysis):
128     """IdealFunc inheritance from dataAnalysis"""
129
```

```
    def __init__(self):
        super().__init__()
        # Ideal function storing variable: <pd.DataFrame>
        self.ideal_function_df = pd.DataFrame(data=None)
        self.ideal_function_df = self.ideal_function()

    def ideal_function(self):
        """Method to determine ideal function: return <pd.DataFrame>"""
        for i_col in self.training.columns[1:]:
            least_square_error = sys.maxsize
            for i_icol in self.ideal.columns[1:]:
                # dev: calculation of y-value deviation between training and ideal data
                dev = np.absolute(np.subtract(self.training[i_col].to_numpy(), self.ideal[i_icol].to_numpy()))
                std_dev = dev / len(self.ideal)

                # sum of squares calculated for <dev>
                lse = np.absolute(np.sum(np.square(std_dev)))
                if lse < least_square_error:
                    '''finding the value the function that minimizes the y-deviation'''
                    least_square_error = lse
                    # Ideal function results: training column = (ideal column, maximum deviation, least square value)
                    self.ideal_function_df[i_col] = (i_icol, dev.max(), least_square_error)
        return self.ideal_function_df

    def ideal_function_visualization_bokeh(self):

        for col in self.training.columns[1:]:
            ideal_function_col = self.ideal_function_df[col][0]
            ideal_graph = figure(title=f'Ideal Function[{ideal_function_col}] vs 'f'Training Function[{col}]')
            ideal_graph.scatter(self.ideal['x'], self.ideal[ideal_function_col], color='purple', alpha=0.6,
                                legend_label=f'Ideal function{ideal_function_col}')
            ideal_graph.scatter(self.training['x'], self.training[col], color='turquoise', alpha=0.6,
                                legend_label=f'Training function{col}')
            ideal_graph.legend.location = 'top_right'
            ideal_graph.legend.click_policy = 'hide'
            save(ideal_graph)
            show(ideal_graph)
```

E. testingModule

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from idealModule import IdealFunc
import dbModule as db


class testingFunc(IdealFunc):
    """Testing mapping with inheritance from IdealFunc Class"""

    def __init__(self):
        super().__init__()
        # Declaring variable containing the Ideal Function:<pd.DataFrame>.
        self.ideal_function = self.ideal_function_df
```

```
        self.mapping_testing = []
        self.mapping_testing = self.testing_function()
        self.test_results = self.testing_results_dataframe()

    def testing_function(self):
        """Mapping of testing data to ideal function: <list>"""
        for i in range(self.testing.shape[0]):
            for k in self.ideal_function.keys():
                # ideal function column value from ideal data
                fn_ideal = self.ideal_function[k][0]
                # maximum deviation corresponding to ideal function
                max_dev = self.ideal_function[k][1]
                index = np.where(self.ideal["x"] == self.testing.iloc[i][0])
                # deviation calculation: subtracting ideal data containing ideal function from testing data
                dev = np.absolute(np.subtract(self.testing.iloc[i][1], self.ideal[fn_ideal].iloc[index].to_numpy()))
                if np.sqrt(2) * max_dev > dev:  # pow(dev, 2) < dev_max also works
                    '''Criteria for mapping test case to the ideal function'''
                    # List containing results from mapping criteria: <list>
                    self.mapping_testing.append((self.testing.iloc[i][0], self.testing.iloc[i][1], fn_ideal, dev[0]))
        return self.mapping_testing

    def testing_results_dataframe(self):
        """
        Converting testing_results <list> -> <pd.Dataframe>
        - Allows for column names be added
        """
        # Retrieve testing_function() method results
        self.testing_results_df = pd.DataFrame(self.mapping_testing, columns=['testing_x',
                                                                               'testing_Yn',
                                                                               'ideal_function',
                                                                               'deviation'])
        return self.testing_results_df

    def testing_visualization(self):
        """Visualize testing data [C] mapping to ideal function"""
        fig, axes = plt.subplots()
        axes.plot(self.test_results[['testing_Yn']], label="Testing Results", alpha=0.6)
        # Functions from ideal data that are present in ideal function: ideal_data[idea function columns]
        axes.plot(self.ideal[['y49', 'y39', 'y40',  'y4']], label='Ideal Function', alpha=0.9)
        plt.title("Test data(C) to Ideal Function Mapping")
        plt.legend()
        plt.show()

    def appending_results(self):
        """Appending test mapping results to database: idealTable"""
        test_append = db.dbCreate()
        # Method argument from dbCreate: update_tables(testing_results)
        test_append.update_tables(self.test_results)
        return "Test mapping results have been appended to database"
```

F. ClassConstruct

```
import dbModule as db
import idealModule as iM
import testModule as tM




"""
Class and method calls for idealModule.py
"""
'''Data Analysis and Visualization Class initiation'''



class funtionalDataAnalysis(object):
    def __init__(self):
        """Declaring class object as attribute"""
        # dataAnalysis class object
        self.dataAnalysis = iM.dataAnalysis()
        # IdealFunc class object
        self.idealFunc = iM.IdealFunc()
        # dbCreate class object
        self.database = db.dbCreate()
        # testFunc class object
        self.testFunc = tM.testingFunc()

    def analysis(self):
        """Exploratory Data Analysis: dataAnalysis<class>"""
        # Summary statistics
        sum_stats = self.dataAnalysis.sum_stats()
        # Checking null value: Test Driven Development
        null_values = self.dataAnalysis.test_null_values()
        # Inspecting data shapes
        data_shapes = self.dataAnalysis.data_shapes()
        # Check and conversion of data types
        data_types = self.dataAnalysis.data_types()
        return [sum_stats, null_values, data_shapes, data_types]

    def results(self):
        """Results for opertion performed in task 1 and task 2: pd.DataFrames"""
        # Ideal function results: task 1
        ideal_function_df = self.idealFunc.ideal_function()
        # Test mapping results: task 2
        testing_results_df = self.testFunc.testing_results_dataframe()
        return [ideal_function_df, testing_results_df]

    def visualization(self):
        """
        Relevant visualization on: data exploration, ideal function and test mapping
        1 - separate_plots(): dataAnalysis
        2 - linear_plots: dataAnalysis
        3 - Ideal function: IdealFunc
        4 - Test mapping: testFunc
        """
        return self.dataAnalysis.separate_plots(),\
            self.dataAnalysis.linear_plots(),\
            self.idealFunc.ideal_function_visualization_bokeh(),\
            self.testFunc.testing_visualization()


```

```
"""
Declaring class object, method selection:
- classConstructor.analysis()
- classConstructor.results()
- class Constructor.visualization()
"""
classConstructor = funtionalDataAnalysis()
```

G. Unittest idealModule

```
import unittest
import idealModule as iM
import dbModule as db



class TestdbConnection(unittest.TestCase):
    """Testing that database connection is present"""
    class testDbConnetion(unittest.TestCase):
        connection = None

        def setUp(self):
            #set up database connection from dbModule.py
            self.dBase = db.dbCreate()
            self.db_connection = self.dBase.engine.connect()
            self.analysisClass = iM.dataAnalysis()

        def tearDown(self):
            if self.db_connection is not None and self.db_connection:
                self.db_connection.close()



class TestidealFunction(unittest.TestCase):
    def setUp(self):
        self.analysisClass = iM.dataAnalysis()
        self.idealFuncClass = iM.IdealFunc()
    def test_test_null_values(self):
        if self.analysisClass.test_null_values():
            expected_message = "Null values are present"
            self.assertEqual(expected_message)

    def test_type(self):
        """Test to determine if all dataset types are as expected: <pd.DataFrame>"""
        train = type(self.analysisClass.training)
        ideal = type(self.analysisClass.ideal)
        self.assertIs(train, ideal, "This is not the same")

    def test_columns(self):
        """Test to determine if training function columns are presents in ideal
    function: y1, y2, y3, y4"""
        train_col = list(self.analysisClass.training.columns[1:])
        test_col = list(self.idealFuncClass.ideal_function().columns)
        self.assertListEqual(train_col, test_col)

    def test_ideal_function_results_columns(self):
        """Test to determine if 4 ideal functions are returned"""
        test = self.idealFuncClass.ideal_function().columns
        expected = len(test)
        self.assertEqual(expected, 4)

if __name__ == '__main__':
```

```
    unittest.main()
```

H. Unittest testModule

```
import unittest
import testModule as tM



class TestTestFunc(unittest.TestCase):
    def setUp(self):
        """Set up variables for testing of testModule.py module"""
        # Testing mapping results: <pd.DataFrame>
        self.testing_results = tM.testingFunc().testing_results_dataframe()
        # Ideal function results: <pd.DataFrame>
        self.ideal_function = tM.testingFunc().ideal_function
        # Testing data(B): <pd.DataFrame>
        self.testing = tM.testingFunc().testing
    def test_if_test_resuls(self):
        """Test for instance of testing results: <pd.DataFrame>"""
        if self.testing_results is not None:
            self.test_testResults_instance = True
            return self.test_testResults_instance
        self.assertNotIsInstance(self.test_testResults_instance, True, "There is no <pd.DataFrame>: Testing Results")
    def test_column_names(self):
        """Testing for expected column names in testing results"""
        results = list(self.testing_results.columns)
        expected = ['testing_x', 'testing_Yn', 'ideal_function', 'deviation']
        self.assertListEqual(results, expected, "columns are not as expected in testing results dataFrame")
    def testing_data_shape(self):
        """Test if testing data[B] has correct shape"""
        result = self.testing.shape
        self.assertEqual(result, (400, 5), "Testing data does not have expected shape")
    def test_for_string(self):
        """Test to ensure ideal function is represented in testing results"""
        expected = type(self.testing_results.iloc[:, 2])
        self.assertTrue(expected, "str")



if __name__ == '__main__':
    unittest.main()
```

I. Test Mapping Results

Table 3: Test data(C) mapping to Ideal Function.

Mapping Results				
	testing_x	testing_Yn	ideal_function	deviation
0	-2.000000e+01	0.470726	y49	0.442219
1	-1.990000e+01	0.901188	y49	0.033544
2	-1.980000e+01	0.421587	y49	0.392086
3	-1.970000e+01	0.788560	y49	0.036986
4	-1.960000e+01	0.655439	y49	0.026525
5	-1.950000e+01	0.991540	y49	0.386000
6	-1.940000e+01	0.376168	y49	0.146898

Continuation of Table 3				
	testing_x	testing_Yn	ideal_function	deviation
7	-1.930000e+01	0.148491	y49	0.286874
8	-1.920000e+01	0.782545	y49	0.439230
9	-1.910000e+01	0.520844	y49	0.273010
10	-1.900000e+01	0.342035	y49	0.192157
11	-1.890000e+01	-0.094569	y49	0.144992
12	-1.880000e+01	-0.039013	y49	0.010522
13	-1.870000e+01	-0.031329	y49	0.117670
14	-1.860000e+01	0.129910	y49	0.376884
15	-1.850000e+01	-0.480384	y49	0.137903
16	-1.840000e+01	-0.686957	y49	0.252391
17	-1.830000e+01	-0.872322	y49	0.350014
18	-1.820000e+01	-0.732458	y49	0.127625
19	-1.810000e+01	-0.619507	y49	0.061807
20	-1.800000e+01	-0.308173	y49	0.442814
21	-1.790000e+01	-0.610428	y49	0.202729
22	-1.780000e+01	-1.042045	y49	0.174843
23	-1.770000e+01	-0.953951	y49	0.041368
24	-1.760000e+01	-0.855656	y49	0.093189
25	-1.750000e+01	-0.834149	y49	0.141477
26	-1.740000e+01	-0.922123	y49	0.070537
27	-1.730000e+01	-0.780745	y49	0.219029
28	-1.720000e+01	-0.768909	y49	0.227991
29	-1.710000e+01	-0.725282	y49	0.258783
30	-1.700000e+01	-1.197468	y49	0.236070
31	-1.690000e+01	-0.521724	y49	0.407400
32	-1.680000e+01	-0.745946	y49	0.141621
33	-1.670000e+01	-0.384063	y49	0.453078
34	-1.660000e+01	-0.406418	y49	0.371935
35	-1.650000e+01	-0.904889	y49	0.193104
36	-1.640000e+01	-0.886621	y49	0.248515
37	-1.630000e+01	-0.569396	y49	0.011343
38	-1.620000e+01	-0.372459	y49	0.099963
39	-1.610000e+01	-0.311461	y49	0.070610
40	-1.600000e+01	-0.208638	y49	0.079265
41	-1.590000e+01	-0.097157	y49	0.093702
42	-1.580000e+01	0.278870	y49	0.370777
43	-1.570000e+01	-0.475217	y49	0.483180
44	-1.560000e+01	0.298167	y49	0.190413
45	-1.550000e+01	0.600577	y49	0.394109
46	-1.540000e+01	-0.031072	y49	0.334190
47	-1.530000e+01	0.428303	y49	0.031563
48	-1.520000e+01	0.183508	y49	0.302891
49	-1.510000e+01	0.176145	y49	0.395052
50	-1.500000e+01	0.963083	y49	0.312795
51	-1.490000e+01	0.380090	y49	0.342791
52	-1.480000e+01	0.319646	y49	0.468606
53	-1.470000e+01	0.865911	y49	0.020164
54	-1.460000e+01	0.586277	y49	0.308514

Continuation of Table 3				
	testing_x	testing_Yn	ideal_function	deviation
55	-1.450000e+01	0.622747	y49	0.312148
56	-1.440000e+01	0.929903	y49	0.035755
57	-1.430000e+01	1.136824	y49	0.150052
58	-1.420000e+01	1.265151	y49	0.267124
59	-1.410000e+01	1.152707	y49	0.153398
60	-1.400000e+01	1.166630	y49	0.176022
61	-1.390000e+01	0.667505	y49	0.304503
62	-1.380000e+01	0.719416	y49	0.224280
63	-1.370000e+01	0.653254	y49	0.252701
64	-1.360000e+01	0.976396	y49	0.117234
65	-1.350000e+01	1.005641	y49	0.201857
66	-1.340000e+01	0.776226	y49	0.035850
67	-1.330000e+01	0.651724	y49	0.017846
68	-1.320000e+01	0.131149	y49	0.460925
69	-1.310000e+01	0.269618	y49	0.239043
70	-1.300000e+01	0.897449	y49	0.477282
71	-1.290000e+01	-0.141294	y49	0.468769
72	-1.280000e+01	0.427758	y49	0.196249
73	-1.270000e+01	-0.056259	y49	0.189491
74	-1.260000e+01	-0.028128	y49	0.061751
75	-1.250000e+01	0.033890	y49	0.100211
76	-1.240000e+01	-0.403024	y49	0.237420
77	-1.230000e+01	-0.563566	y49	0.300334
78	-1.220000e+01	-0.097954	y49	0.260275
79	-1.210000e+01	-0.082825	y49	0.366823
80	-1.200000e+01	-0.321294	y49	0.215279
81	-1.190000e+01	-0.386933	y49	0.231204
82	-1.180000e+01	-1.185749	y49	0.492224
83	-1.170000e+01	-0.470908	y49	0.291075
84	-1.160000e+01	-0.452623	y49	0.370205
85	-1.150000e+01	-1.361314	y49	0.485862
86	-1.140000e+01	-0.887969	y49	0.031359
87	-1.130000e+01	-0.953454	y49	0.000565
88	-1.120000e+01	-1.229771	y49	0.250593
89	-1.110000e+01	-1.084267	y49	0.089714
90	-1.100000e+01	-0.639028	y49	0.360962
91	-1.090000e+01	-0.881391	y49	0.114045
92	-1.080000e+01	-0.632955	y49	0.347981
93	-1.070000e+01	-1.012092	y49	0.055457
94	-1.060000e+01	-0.914829	y49	0.007946
95	-1.050000e+01	-0.477487	y49	0.402209
96	-1.040000e+01	-0.999013	y49	0.171186
97	-1.030000e+01	-0.850536	y49	0.082850
98	-1.020000e+01	-1.170452	y49	0.470577
99	-1.010000e+01	-0.198818	y49	0.426253
100	-1.000000e+01	-0.355209	y49	0.188812
101	-9.900000e+00	-0.139294	y49	0.318242
102	-9.800000e+00	-0.551900	y49	0.185421

Continuation of Table 3				
	testing_x	testing_Yn	ideal_function	deviation
103	-9.700000e+00	-0.431426	y49	0.159665
104	-9.600000e+00	-0.467195	y49	0.292868
105	-9.500000e+00	0.077018	y49	0.152169
106	-9.400000e+00	-0.416249	y49	0.441024
107	-9.300000e+00	0.516175	y49	0.391721
108	-9.200000e+00	-0.002018	y49	0.224908
109	-9.100000e+00	0.633638	y49	0.314539
110	-9.000000e+00	0.846827	y49	0.434708
111	-8.900000e+00	0.096171	y49	0.404850
112	-8.800000e+00	0.286327	y49	0.298590
113	-8.700000e+00	0.433712	y49	0.229258
114	-8.600000e+00	1.091919	y49	0.357522
115	-8.500000e+00	0.401847	y49	0.396640
116	-8.400000e+00	1.130740	y49	0.276141
117	-8.300000e+00	0.960807	y49	0.058635
118	-8.200000e+00	1.088531	y49	0.147801
119	-8.100000e+00	0.755683	y49	0.214207
120	-8.000000e+00	0.750161	y49	0.239198
121	-7.900000e+00	0.506653	y49	0.492289
122	-7.800000e+00	0.759638	y49	0.238905
123	-7.700000e+00	1.122683	y49	0.134515
124	-7.600000e+00	1.153044	y49	0.185124
125	-7.500000e+00	1.317974	y49	0.379974
126	-7.400000e+00	1.250512	y49	0.351804
127	-7.300000e+00	0.376634	y49	0.473802
128	-7.200000e+00	0.491220	y49	0.302448
129	-7.100000e+00	0.777721	y49	0.048752
130	-7.000000e+00	0.794745	y49	0.137758
131	-6.900000e+00	0.871025	y49	0.292585
132	-6.800000e+00	0.159666	y49	0.334448
133	-6.700000e+00	0.584985	y49	0.180135
134	-6.600000e+00	-0.064462	y49	0.376004
135	-6.500000e+00	-0.214711	y49	0.429831
136	-6.400000e+00	-0.174698	y49	0.291247
137	-6.300000e+00	-0.291448	y49	0.308262
138	-6.200000e+00	0.070007	y49	0.153096
139	-6.100000e+00	-0.279919	y49	0.097757
140	-6.000000e+00	-0.479402	y49	0.199986
141	-5.900000e+00	-0.190103	y49	0.183773
142	-5.800000e+00	-0.929586	y49	0.464984
143	-5.700000e+00	-0.548662	y49	0.002023
144	-5.600000e+00	-0.777185	y49	0.145918
145	-5.500000e+00	-0.774965	y49	0.069425
146	-5.400000e+00	-1.260793	y49	0.488029
147	-5.300000e+00	-0.510238	y49	0.322029
148	-5.200000e+00	-0.969784	y49	0.086329
149	-5.100000e+00	-0.664293	y49	0.261521
150	-5.000000e+00	-1.073355	y49	0.114431

Continuation of Table 3				
	testing_x	testing_Yn	ideal_function	deviation
151	-4.900000e+00	-0.805357	y49	0.177096
152	-4.800000e+00	-1.450424	y49	0.454259
153	-4.700000e+00	-0.922709	y49	0.077214
154	-4.600000e+00	-1.343699	y49	0.350008
155	-4.500000e+00	-0.525256	y49	0.452274
156	-4.400000e+00	-0.748598	y49	0.203004
157	-4.300000e+00	-0.717475	y49	0.198691
158	-4.200000e+00	-1.095071	y49	0.223495
159	-4.100000e+00	-1.213766	y49	0.395489
160	-4.000000e+00	-0.851759	y49	0.094957
161	-3.900000e+00	-1.174013	y49	0.486247
162	-3.800000e+00	-1.001504	y49	0.389647
163	-3.700000e+00	-0.616137	y49	0.086300
164	-3.600000e+00	-0.369100	y49	0.073420
165	-3.500000e+00	-0.179031	y49	0.171752
166	-3.400000e+00	-0.627400	y49	0.371859
167	-3.300000e+00	-0.184814	y49	0.027069
168	-3.200000e+00	-0.386956	y49	0.328582
169	-3.100000e+00	-0.377606	y49	0.419186
170	-3.000000e+00	0.305337	y49	0.164217
171	-2.900000e+00	0.297005	y49	0.057756
172	-2.800000e+00	0.121041	y49	0.213947
173	-2.700000e+00	0.671080	y49	0.243700
174	-2.600000e+00	0.842309	y49	0.326808
175	-2.500000e+00	0.447064	y49	0.151408
176	-2.400000e+00	0.235638	y49	0.439825
177	-2.300000e+00	1.236485	y49	0.490779
178	-2.200000e+00	1.089812	y49	0.281316
179	-2.100000e+00	0.492880	y49	0.370330
180	-2.000000e+00	1.135371	y49	0.226073
181	-1.900000e+00	0.924631	y49	0.021669
182	-1.800000e+00	0.573744	y49	0.400104
183	-1.700000e+00	1.420005	y49	0.428340
184	-1.700000e+00	1.420005	y39	0.478330
185	-1.600000e+00	0.586018	y49	0.413556
186	-1.500000e+00	1.475420	y49	0.477925
187	-1.500000e+00	1.475420	y39	0.222915
188	-1.400000e+00	0.832964	y49	0.152486
189	-1.400000e+00	0.832964	y39	0.141586
190	-1.300000e+00	1.131613	y49	0.168055
191	-1.300000e+00	1.131613	y39	0.405172
192	-1.200000e+00	1.226184	y49	0.294145
193	-1.100000e+00	0.738072	y49	0.153135
194	-1.100000e+00	0.738072	y39	0.419279
195	-1.000000e+00	0.802715	y49	0.038756
196	-1.000000e+00	0.802715	y39	0.644186
197	-9.000000e-01	1.081776	y49	0.298449
198	-8.000000e-01	0.663257	y49	0.054100

Continuation of Table 3				
	testing_x	testing_Yn	ideal_function	deviation
199	-7.000000e-01	0.368971	y49	0.275246
200	-7.000000e-01	0.368971	y39	0.523189
201	-6.000000e-01	0.581721	y49	0.017079
202	-5.000000e-01	0.728226	y49	0.248801
203	-4.000000e-01	-0.026697	y49	0.416115
204	-4.000000e-01	-0.026697	y39	0.202721
205	-3.000000e-01	0.477675	y49	0.182155
206	-3.000000e-01	0.477675	y39	0.683195
207	-2.000000e-01	0.483246	y49	0.284577
208	-2.000000e-01	0.483246	y39	0.641916
209	-1.000000e-01	0.043585	y49	0.056248
210	-1.000000e-01	0.043585	y39	0.133419
211	2.840000e-13	-0.136745	y49	0.136745
212	2.840000e-13	-0.136745	y39	0.136745
213	1.000000e-01	-0.115265	y49	0.215099
214	1.000000e-01	-0.115265	y39	0.225099
215	2.000000e-01	-0.073356	y49	0.272025
216	2.000000e-01	-0.073356	y39	0.312025
217	3.000000e-01	0.149169	y49	0.146351
218	3.000000e-01	0.149169	y39	0.236351
219	4.000000e-01	0.115741	y49	0.273677
220	4.000000e-01	0.115741	y39	0.433677
221	5.000000e-01	0.964403	y49	0.484978
222	5.000000e-01	0.964403	y39	0.234978
223	6.000000e-01	0.179446	y49	0.385197
224	7.000000e-01	0.772613	y49	0.128396
225	7.000000e-01	0.772613	y39	0.361604
226	8.000000e-01	0.456331	y49	0.261025
227	9.000000e-01	0.313756	y49	0.469571
228	1.000000e+00	0.595173	y49	0.246298
229	1.100000e+00	0.589127	y49	0.302081
230	1.200000e+00	0.808063	y49	0.123976
231	1.300000e+00	0.499760	y49	0.463799
232	1.400000e+00	1.202030	y49	0.216580
233	1.500000e+00	1.253614	y49	0.256119
234	1.600000e+00	1.063556	y49	0.063982
235	1.700000e+00	0.527147	y49	0.464518
236	1.800000e+00	1.159094	y49	0.185246
237	1.900000e+00	0.616296	y49	0.330004
238	2.000000e+00	1.233090	y49	0.323792
239	2.100000e+00	1.061665	y49	0.198455
240	2.200000e+00	1.020536	y49	0.212040
241	2.300000e+00	0.736138	y49	0.009567
242	2.400000e+00	0.977980	y49	0.302517
243	2.500000e+00	0.356373	y49	0.242099
244	2.600000e+00	0.063137	y49	0.452364
245	2.700000e+00	0.586548	y49	0.159168
246	2.800000e+00	0.679065	y49	0.344077

Continuation of Table 3				
	testing_x	testing_Yn	ideal_function	deviation
247	2.900000e+00	0.121412	y49	0.117838
248	3.000000e+00	0.202768	y49	0.061648
249	3.100000e+00	0.166501	y49	0.124920
250	3.200000e+00	-0.488264	y49	0.429890
251	3.300000e+00	0.056216	y49	0.213962
252	3.400000e+00	-0.437115	y49	0.181574
253	3.500000e+00	-0.067900	y49	0.282884
254	3.600000e+00	-0.885326	y49	0.442805
255	3.700000e+00	-0.057432	y49	0.472404
256	3.800000e+00	-0.213818	y49	0.398040
257	3.900000e+00	-0.874648	y49	0.186882
258	4.000000e+00	-0.495450	y49	0.261352
259	4.100000e+00	-0.596367	y49	0.221910
260	4.200000e+00	-0.967718	y49	0.096142
261	4.300000e+00	-1.354115	y49	0.437949
262	4.400000e+00	-0.731108	y49	0.220494
263	4.500000e+00	-0.794939	y49	0.182592
264	4.600000e+00	-1.429302	y49	0.435611
265	4.700000e+00	-1.107551	y49	0.107627
266	4.800000e+00	-0.894886	y49	0.101279
267	4.900000e+00	-1.240844	y49	0.258392
268	5.000000e+00	-1.143652	y49	0.184728
269	5.100000e+00	-0.729164	y49	0.196651
270	5.200000e+00	-0.664038	y49	0.219417
271	5.300000e+00	-1.119134	y49	0.286866
272	5.400000e+00	-0.721799	y49	0.050966
273	5.500000e+00	-0.747684	y49	0.042144
274	5.600000e+00	-0.261433	y49	0.369834
275	5.700000e+00	-0.143893	y49	0.406792
276	5.800000e+00	-0.514070	y49	0.049468
277	5.900000e+00	-0.146422	y49	0.227455
278	6.000000e+00	-0.737571	y49	0.458155
279	6.100000e+00	-0.453188	y49	0.271026
280	6.200000e+00	-0.169169	y49	0.086080
281	6.300000e+00	0.425130	y49	0.408316
282	6.400000e+00	-0.021938	y49	0.138488
283	6.500000e+00	0.674392	y49	0.459272
284	6.600000e+00	0.163098	y49	0.148444
285	6.700000e+00	0.043099	y49	0.361751
286	6.800000e+00	0.554406	y49	0.060293
287	6.900000e+00	0.832357	y49	0.253917
288	7.000000e+00	0.725237	y49	0.068251
289	7.100000e+00	0.988657	y49	0.259688
290	7.200000e+00	0.783049	y49	0.010618
291	7.300000e+00	0.724913	y49	0.125523
292	7.400000e+00	0.423126	y49	0.475582
293	7.500000e+00	0.957920	y49	0.019920
294	7.600000e+00	1.114866	y49	0.146946

Continuation of Table 3				
	testing_x	testing_Yn	ideal_function	deviation
295	7.700000e+00	1.121509	y49	0.133341
296	7.800000e+00	0.944389	y49	0.054154
297	7.900000e+00	1.168658	y49	0.169717
298	8.000000e+00	1.122104	y49	0.132745
299	8.100000e+00	1.463979	y49	0.494089
300	8.200000e+00	0.792796	y49	0.147934
301	8.300000e+00	0.693673	y49	0.208498
302	8.400000e+00	1.043548	y49	0.188949
303	8.500000e+00	1.271445	y49	0.472957
304	8.600000e+00	0.922974	y49	0.188577
305	8.700000e+00	1.051320	y49	0.388350
306	8.800000e+00	0.427356	y49	0.157562
307	8.900000e+00	0.711770	y49	0.210749
308	9.000000e+00	-0.029297	y49	0.441416
309	9.100000e+00	0.136031	y49	0.183067
310	9.200000e+00	-0.230107	y49	0.452997
311	9.300000e+00	-0.368006	y49	0.492460
312	9.400000e+00	0.064808	y49	0.040033
313	9.500000e+00	0.158023	y49	0.233174
314	9.600000e+00	0.322069	y49	0.496396
315	9.700000e+00	-0.569195	y49	0.297434
316	9.800000e+00	-0.822315	y49	0.455836
317	9.900000e+00	-0.353072	y49	0.104464
318	1.000000e+01	-0.622970	y49	0.078949
319	1.010000e+01	-0.223119	y49	0.401951
320	1.020000e+01	-0.241356	y49	0.458518
321	1.030000e+01	-0.984443	y49	0.216757
322	1.040000e+01	-0.667201	y49	0.160626
323	1.050000e+01	-1.254946	y49	0.375250
324	1.060000e+01	-1.319075	y49	0.396299
325	1.070000e+01	-0.502400	y49	0.454235
326	1.080000e+01	-1.233153	y49	0.252217
327	1.090000e+01	-0.617343	y49	0.378093
328	1.100000e+01	-1.047367	y49	0.047376
329	1.110000e+01	-1.425399	y49	0.430846
330	1.120000e+01	-1.235018	y49	0.255840
331	1.130000e+01	-0.811512	y49	0.142507
332	1.140000e+01	-0.438904	y49	0.480424
333	1.150000e+01	-0.500012	y49	0.375440
334	1.160000e+01	-0.920152	y49	0.097324
335	1.170000e+01	-0.553895	y49	0.208089
336	1.180000e+01	-0.478413	y49	0.215112
337	1.190000e+01	-0.433403	y49	0.184734
338	1.200000e+01	-0.443487	y49	0.093086
339	1.210000e+01	-0.101856	y49	0.347792
340	1.220000e+01	-0.397827	y49	0.039597
341	1.230000e+01	-0.483397	y49	0.220166
342	1.240000e+01	-0.604073	y49	0.438469

Continuation of Table 3				
	testing_x	testing_Yn	ideal_function	deviation
343	1.250000e+01	-0.278983	y49	0.212661
344	1.260000e+01	0.105028	y49	0.071405
345	1.270000e+01	-0.132870	y49	0.266102
346	1.280000e+01	0.538460	y49	0.306950
347	1.290000e+01	-0.088811	y49	0.416286
348	1.300000e+01	-0.035723	y49	0.455890
349	1.310000e+01	0.159899	y49	0.348762
350	1.320000e+01	0.988631	y49	0.396557
351	1.330000e+01	0.847325	y49	0.177755
352	1.340000e+01	0.533876	y49	0.206500
353	1.350000e+01	0.984745	y49	0.180961
354	1.360000e+01	1.099316	y49	0.240154
355	1.370000e+01	0.990093	y49	0.084138
356	1.380000e+01	0.550026	y49	0.393670
357	1.390000e+01	1.439286	y49	0.467279
358	1.400000e+01	1.343419	y49	0.352811
359	1.410000e+01	1.480255	y49	0.480945
360	1.420000e+01	0.667839	y49	0.330188
361	1.430000e+01	1.163358	y49	0.176586
362	1.440000e+01	1.046789	y49	0.081131
363	1.450000e+01	0.452991	y49	0.481904
364	1.460000e+01	1.171278	y49	0.276487
365	1.470000e+01	0.479244	y49	0.366503
366	1.480000e+01	1.068143	y49	0.279891
367	1.490000e+01	0.340768	y49	0.382113
368	1.500000e+01	0.194320	y49	0.455968
369	1.510000e+01	0.237056	y49	0.334141
370	1.520000e+01	0.268649	y49	0.217750
371	1.530000e+01	0.006456	y49	0.390284
372	1.540000e+01	0.415562	y49	0.112444
373	1.550000e+01	0.593502	y49	0.387035
374	1.560000e+01	-0.117187	y49	0.224941
375	1.570000e+01	0.390894	y49	0.382931
376	1.580000e+01	0.029558	y49	0.121465
377	1.590000e+01	-0.291102	y49	0.100244
378	1.600000e+01	0.021146	y49	0.309049
379	1.610000e+01	-0.618808	y49	0.236736
380	1.620000e+01	-0.815537	y49	0.343115
381	1.630000e+01	-0.736390	y49	0.178338
382	1.640000e+01	-0.257235	y49	0.380872
383	1.650000e+01	-0.748537	y49	0.036752
384	1.660000e+01	-0.785728	y49	0.007376
385	1.670000e+01	-0.814895	y49	0.022246
386	1.680000e+01	-0.525376	y49	0.362191
387	1.690000e+01	-1.330052	y49	0.400928
388	1.700000e+01	-0.715587	y49	0.245811
389	1.710000e+01	-0.981426	y49	0.002638
390	1.720000e+01	-1.343804	y49	0.346904

Continuation of Table 3				
	testing_x	testing_Yn	ideal_function	deviation
391	1.730000e+01	-1.369427	y49	0.369653
392	1.740000e+01	-1.255938	y49	0.263279
393	1.750000e+01	-1.082445	y49	0.106819
394	1.760000e+01	-0.786369	y49	0.162476
395	1.770000e+01	-1.227639	y49	0.315056
396	1.780000e+01	-0.812831	y49	0.054372
397	1.790000e+01	-0.686996	y49	0.126162
398	1.800000e+01	-1.062251	y49	0.311264
399	1.810000e+01	-0.782260	y49	0.100946
400	1.820000e+01	-0.634409	y49	0.029576
401	1.830000e+01	-0.713163	y49	0.190854
402	1.840000e+01	-0.359941	y49	0.074625
403	1.850000e+01	-0.485357	y49	0.142876
404	1.860000e+01	-0.445456	y49	0.198482
405	1.870000e+01	-0.325448	y49	0.176449
406	1.880000e+01	0.414704	y49	0.464239
407	1.890000e+01	-0.117691	y49	0.168114
408	1.900000e+01	0.058980	y49	0.090897
409	1.910000e+01	-0.120873	y49	0.368707
410	1.920000e+01	0.243773	y49	0.099542
411	1.930000e+01	0.540223	y49	0.104857
412	1.940000e+01	0.972079	y49	0.449014
413	1.950000e+01	0.651326	y49	0.045786
414	1.960000e+01	0.777651	y49	0.095687
415	1.970000e+01	0.490434	y49	0.261139
416	1.980000e+01	0.491736	y49	0.321937
417	1.990000e+01	0.731482	y49	0.136162
End of Table				

References

About code coverage. URL https://docs.codecov.com/docs/about-code-coverage.

Balamurugan Balusamy, Naveen Chilamkurti, Beena T Lucia Agnes, and Poongodi T. *17.2.1 Importance of Data Preprocessing.* Institution of Engineering and Technology, 2021. ISBN 978-1-5231-3528-8. URL http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edsknv&AN=edsknv.kt012HJ4B3&site=eds-live&scope=site.

Padraig Cunningham and Sarah Jane Delany. Algorithmic bias and regularisation in machine learning. 2020. URL http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edsarx&AN=edsarx.2005.09052&site=eds-live&scope=site.

Armando Fandango. *Python data analysis data manipulation and complex data analysis with Python.* Birmingham Mumbai Packt Publishing March, 2017.

Carlos Vladimiro Gonzalez Zelaya. Towards explaining the effects of data preprocessing on machine learning. *2019 IEEE 35th International Conference on Data Engineering (ICDE), Data Engineering (ICDE), 2019 IEEE 35th International Conference on*, pages 2086 – 2090, 2019. ISSN 978-1-5386-7474-1. URL http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=edseee&AN=edseee.8731532&site=eds-live&scope=site.

Siddharta Govindaraj. *Test Driven Python Development.* Packt Publishing, 2015. ISBN 9781783987924.

Johannes Hemmerich, Niklas Tenhaef, Wolfgang Wiechert, and Stephan Noack. pyfoomb: Python framework for object oriented modeling of bioprocesses. *Engineering in Life Sciences*, 21, 01 2021. doi: 10.1002/elsc.202000088.

Daniel Kahneman. *Thinking, fast and slow.* Farrar, Straus and Giroux, New York, 2011. ISBN 9780374275631 0374275637. URL https://www.amazon.de/Thinking-Fast-Slow-Daniel-Kahneman/dp/0374275637/ref=wl_it_dp_o_pdT1_nS_nC?ie=UTF8&colid=151193SNGKJT9&coliid=I3OCESLZCVDFL7.

Sunil Kapil. *Clean Python [electronic resource] : Elegant Coding in Python.* Apress, 2019. ISBN 9781484248782. URL http://search.ebscohost.com.pxz.iubh.de:8080/login.aspx?direct=true&db=cat05114a&AN=ihb.45035&site=eds-live&scope=site.

Prof.Dr. Ulrich Kerzel. *Use Case and Evaluation.* IUBH Internationale Hochschule GmbH, 2020.

Martin Kleppmann. *Designing Data-Intensive Applications.* O'Reilly, Beijing, 2017. ISBN 978-1-4493-7332-0. URL https://www.safaribooksonline.com/library/view/designing-data-intensive-applications/9781491903063/.

Steven J. Miller. The method of least squares, 2006. URL https://web.williams.edu/Mathematics/sjmiller/public_html/probabilitylifesaver/MethodLeastSquares.pdf.

In Jae Myung. The importance of complexity in model selection. *Journal of Mathematical Psychology 44*. doi: 10.1006.jmps.1999.1283. URL http://www.idealibrary.com.

Micheal J. Quillin. Object orientated analysis and desing, 11 2001. URL https://www.umsl.edu/~sauterv/analysis/488_f01_papers/quillin.htm.

Seyed Mohsen Sedighi. Integration of object-oriented and relational database systems. Master's thesis, University of Wollongong, 1993. URL https://ro.uow.edu.au/theses/2804.

Hannah Stepanek. *Thinking in Pandas: How to Use the Python Data Analysis Library the Right Way*. 01 2020. ISBN 978-1-4842-5838-5. doi: 10.1007/978-1-4842-5839-2.

S. van Buuren. *Flexible Imputation of Missing Data*. Chapman & Hall/CRC Interdisciplinary Statistics. CRC Press LLC, 2018. ISBN 9781138588318. URL `https://www.crcpress.com/Flexible-Imputation-of-Missing-Data-Second-Edition/Buuren/p/book/9781138588318`.

Xingquan Zhu and Xindong Wu. Class noise vs. attribute noise: A quantitative study. *Artificial Intelligence Review*, 22:177–210, 11 2004. doi: 10.1007/s10462-004-0751-8.