Description of the Task: Python-program that uses training data to choose the four ideal functions which are the best fit out of the fifty provided (C) *. You get (A) 4 training datasets and (B) one test dataset, as well as (C) datasets for 50 ideal functions. All data respectively consists of x-y-pairs of values.

Author	Lesedi Kopeledi Matshehla
Matriculation number	3210470
Address	10696 Thatch Hill Estate, Serenade Street, Olievenhoutbosch, Centurion, 0187
Place	Realized in Pretoria, Gauteng, South Africa
Tutor	Dr. Croitoru Cosmina
Course	DLMDSPWP01 – Programming with Python
University	International University of Applied Science - Online
Study-branch	Master of Science in Data Science - 120 ECTS
Task date	15-November-2024

Abstract

Python-program is the most popular programming language known as an interpreter. Learn the fundamentals of data analytics and interpretation with python. Start from data wrangler methods, followed by least squares, mean squared errors through the whole process of selecting the four ideal functions which are the best fit based on train data. By the end of this task you will understand inter-quartile range method to identify outliers. Python code is very good in terms of data visualizations to plot types of charts. Gain confidence with the results of your analysis performed by unit test framework technique known as ”unittest.TestCase”. Python has built-in mathematical libraries. This simplifies the calculation of mathematical problems used to perform data analysis. The codes include how to import, clean, manipulate, and visualize data using some of the most popular python libraries. You need to calculate least squares method for mathematical regression used to determine best fit for a set of data that provides a visual demonstration of the relationship between the data points. This method is commonly used by the statistician to predict the dependent variables. Mean squared error is a metric used to measure an error in statistical models by using the average squared difference between observed and predicted values. The python codes are provided to answer all questions of this task. A code implies system of words, numbers, letters, or signals used to execute output results.

Table Of Contents

I List of Figure Diagrams

II List of Table Diagrams

III Table of Abbreviations

CONFIG	Configuration
CSV	Comma Separated Values
DLMDSPWP01	Programming with Python Course Code
ECTS	European Credit Transfer and Accumulation System
IQR	Inter-quartile Range
IU	International University of Applied Sciences
LS	Least Squares
MAX	Maximum
MIN	Minimum
MSE	Mean Squared Error
NUMPY	Numerical Python
Q1	First Quartile
Q2	Second Quartile
Q3	Third Quartile
SCIPY	Scientific Python
STD	Standard Deviation
SYS	System Specific

Introduction

Businesses have long relied on professionals with data science and analytical skills to understand and leverage information at their disposal (Harvard Business School Online, 2024). With the proliferation of data, due to the development of programming smart devices and other technological advancements, this need has accelerated. As a result, it is important for organizations to employ individuals who understand what clean data looks like and how to shape raw data into usable forms to gain valuable insights. In this task, python program compile data wrangling, data exploration and unit test to execute correct output. Data wrangling is also called data cleaning, data remediation, or data munging. It refers to a variety of processes designed to transform raw data into more readily used formats. The exact methods differ from project to project depending on the data an organization is leveraging and the goal of output achievement. Data wrangling can be a manual or automated process. Automated data cleaning becomes a necessity in scenarios where datasets are exceptionally large. In organizations that employ a full data team, a data scientist or other team member is typically responsible for data wrangling. In smaller organizations, non-data professionals are often responsible for cleaning their data before leveraging it (Harvard Business School Online, 2024).

A data project requires a unique approach to ensure its final dataset is reliable and accessible. Several processes of data wrangling steps or activities typically inform the approach. Discovery, Structuring, Cleaning, Enriching, Validating, and Publishing are the six steps of data wrangling. This recommends the process of familiarizing yourself with data to conceptualize how you might employ it. Raw data is typically unusable in its raw state because it is either incomplete or not formatted for its intended application. Data structuring transforms raw data to make it more readily leveraged. The form your data takes will depend on the analytical model you use to interpret it. Please remove inherent errors in data that might distort your analysis or render it less valuable. Cleaning can be done in different forms, including deleting empty cells or rows, removing outliers, and standardizing inputs. After an existing data is transformed into a more usable state, you must determine whether you have all of the data necessary for the project at hand. If data is incomplete, you may choose to enrich or augment your data by incorporating values from other datasets. Therefore, it is important to understand what other data is available for use. Should you decide that enrichment is necessary, you need to repeat the prior steps of data wrangling for any new data. Data validation can be used to verify that your data is both consistent and of a high enough quality. During validation, you may discover issues you need to resolve or conclude that your data is ready to be analyzed. Validation is typically done through various automated processes and requires programming. After your data has been validated, you can publish it. This implies making it available to others within your organization for analytical purposes. The format you use to share the written assignment or electronic file information, will depend on your data and organization's goal (Harvard Business School Online, 2024). Following the published information, data visualization can serve visual presentation of information and guide the data-exploration process. Unit test can be performed as a technique where individual components of a data application are tested in isolation to ensure they perform as expected. This method is important to identify and fix bugs early in the development process, significantly improving data quality and reliability.

1 Chapter 1 : DataWrangler Class

The tutor instruction stated that we have two types of datasets to complete programming with python written assignment. Datasets1.zip and Datasets2.zip folders are shared for students access on teams in the Course Feed Programming with Python Written Assignment Channel. Furthermore, the tutor instructions stated that a student with an odd number birth date should select Datasets1.zip folder and a student with an even number birth date should select Datasets2.zip folder. Inside the selected dataset zipped folder, you get (A) 4 training datasets and (B) one test dataset, as well as (C) datasets for 50 ideal functions. All data respectively consists of x-y-pairs of values. Structure of all CSV-files provided:

X	Y
x_1	y_1
...	...
x_n	y_n

Table 1: CSV-files structure.

Consider the definition of data wrangling, also known as data cleaning, data remediation, or data munging, in order to write a Python-program that uses training data to choose the four ideal functions which are the best fit out of the fifty provided $(C)^*$. Data wrangling refers to a variety of processes designed to transform raw data into more readily used formats. In this chapter, you will perform data exploration of data wrangler class methods. The attributes are provided by the name of the file bearing our dataset to be wrangled.

1.1 List of data wrangler methods

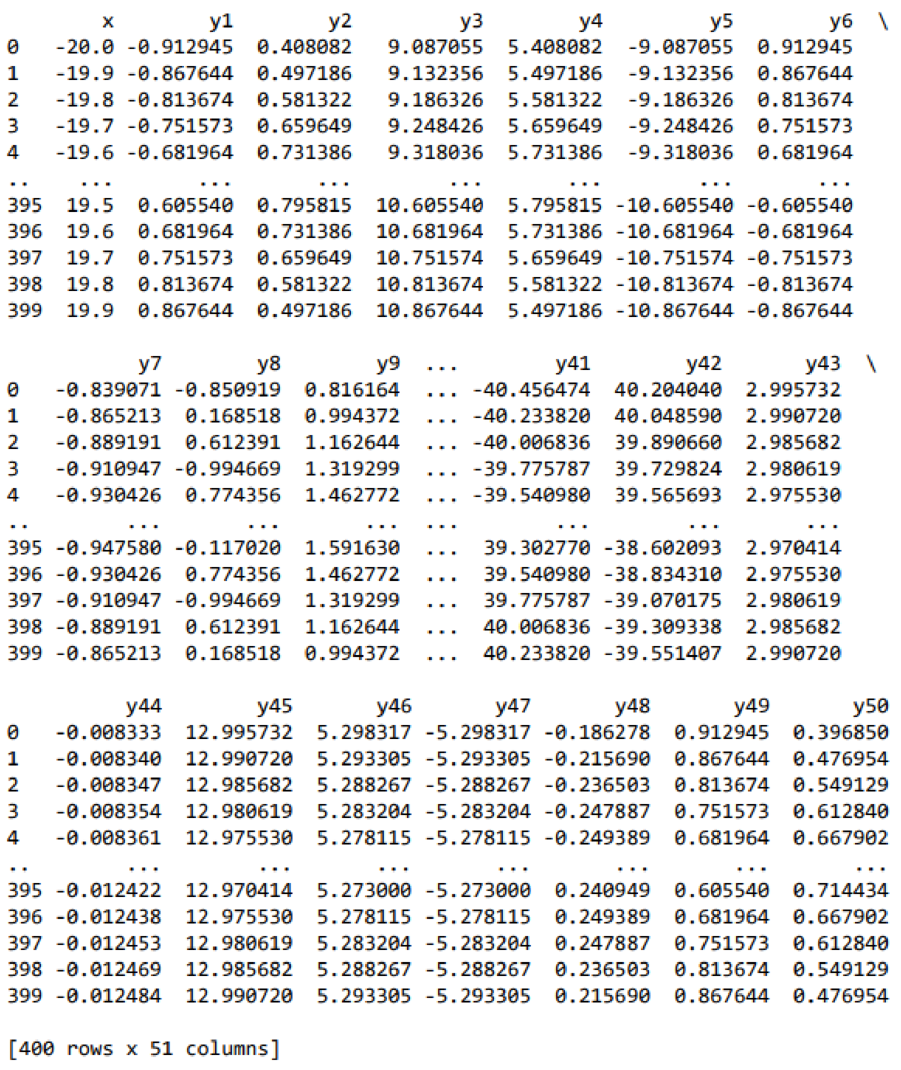
1.1.1 load data()

1. load data(): Loads the dataset from the ideal, test and train files into a pandas dataframe.
Loading data into Python is extremely important in any data science or analytics project. Python provides several libraries, such as Built-in Statistics Module, Pandas, NumPy and SciPy, that enable users to import data from various file formats such as CSV, Excel, JSON, and text files. Once the data is loaded, users can use Python data manipulation and analysis tools to explore and process the data. For example, the Pandas library provides several methods for loading data into a dataframe, such as read_csv(), read_excel(), read_json() and read_sql() (Richman, 2023).

Python-program:
print(”load_data(): Loads the dataset from the file into a pandas dataframe. Remember to change file name.”)
”file_name” = ”ideal.csv”
dataWrangler = DataWrangler(file_name)

df_data = pd.read_csv(”ideal.csv”)
#df_data = dataWrangler.load_data()
print(df_data)
print(” ”)
print(”The compressed view of dataset file to do a quick check of your DataFrame.”)
display(df_data)

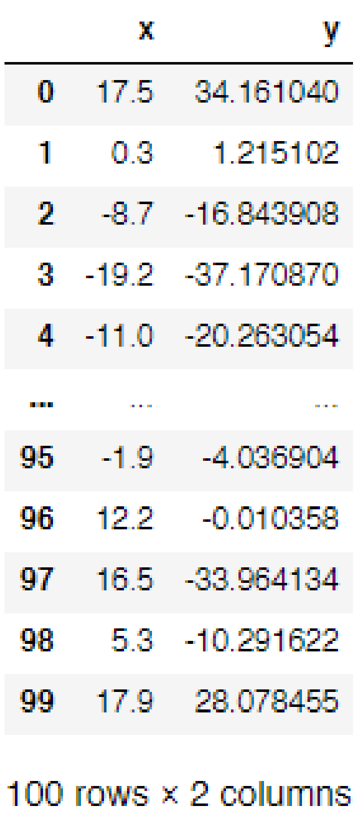
1.1. The compressed view of ideal dataset file to do a quick check of your DataFrame.



```
        x        y1        y2         y3        y4         y5        y6  \
0   -20.0 -0.912945  0.408082   9.087055  5.408082  -9.087055  0.912945
1   -19.9 -0.867644  0.497186   9.132356  5.497186  -9.132356  0.867644
2   -19.8 -0.813674  0.581322   9.186326  5.581322  -9.186326  0.813674
3   -19.7 -0.751573  0.659649   9.248426  5.659649  -9.248426  0.751573
4   -19.6 -0.681964  0.731386   9.318036  5.731386  -9.318036  0.681964
..    ...       ...       ...        ...       ...        ...       ...
395  19.5  0.605540  0.795815  10.605540  5.795815 -10.605540 -0.605540
396  19.6  0.681964  0.731386  10.681964  5.731386 -10.681964 -0.681964
397  19.7  0.751573  0.659649  10.751574  5.659649 -10.751574 -0.751573
398  19.8  0.813674  0.581322  10.813674  5.581322 -10.813674 -0.813674
399  19.9  0.867644  0.497186  10.867644  5.497186 -10.867644 -0.867644

           y7        y8        y9  ...        y41        y42       y43  \
0   -0.839071 -0.850919  0.816164  ... -40.456474  40.204040  2.995732
1   -0.865213  0.168518  0.994372  ... -40.233820  40.048590  2.990720
2   -0.889191  0.612391  1.162644  ... -40.006836  39.890660  2.985682
3   -0.910947 -0.994669  1.319299  ... -39.775787  39.729824  2.980619
4   -0.930426  0.774356  1.462772  ... -39.540980  39.565693  2.975530
..        ...       ...       ...  ...        ...        ...       ...
395 -0.947580 -0.117020  1.591630  ...  39.302770 -38.602093  2.970414
396 -0.930426  0.774356  1.462772  ...  39.540980 -38.834310  2.975530
397 -0.910947 -0.994669  1.319299  ...  39.775787 -39.070175  2.980619
398 -0.889191  0.612391  1.162644  ...  40.006836 -39.309338  2.985682
399 -0.865213  0.168518  0.994372  ...  40.233820 -39.551407  2.990720

          y44        y45       y46       y47       y48       y49       y50
0   -0.008333  12.995732  5.298317 -5.298317 -0.186278  0.912945  0.396850
1   -0.008340  12.990720  5.293305 -5.293305 -0.215690  0.867644  0.476954
2   -0.008347  12.985682  5.288267 -5.288267 -0.236503  0.813674  0.549129
3   -0.008354  12.980619  5.283204 -5.283204 -0.247887  0.751573  0.612840
4   -0.008361  12.975530  5.278115 -5.278115 -0.249389  0.681964  0.667902
..        ...        ...       ...       ...       ...       ...       ...
395 -0.012422  12.970414  5.273000 -5.273000  0.240949  0.605540  0.714434
396 -0.012438  12.975530  5.278115 -5.278115  0.249389  0.681964  0.667902
397 -0.012453  12.980619  5.283204 -5.283204  0.247887  0.751573  0.612840
398 -0.012469  12.985682  5.288267 -5.288267  0.236503  0.813674  0.549129
399 -0.012484  12.990720  5.293305 -5.293305  0.215690  0.867644  0.476954

[400 rows x 51 columns]
```

Figure 1: Ideal File DataFrame

1.2. The compressed view of test dataset file to do a quick check of your DataFrame.

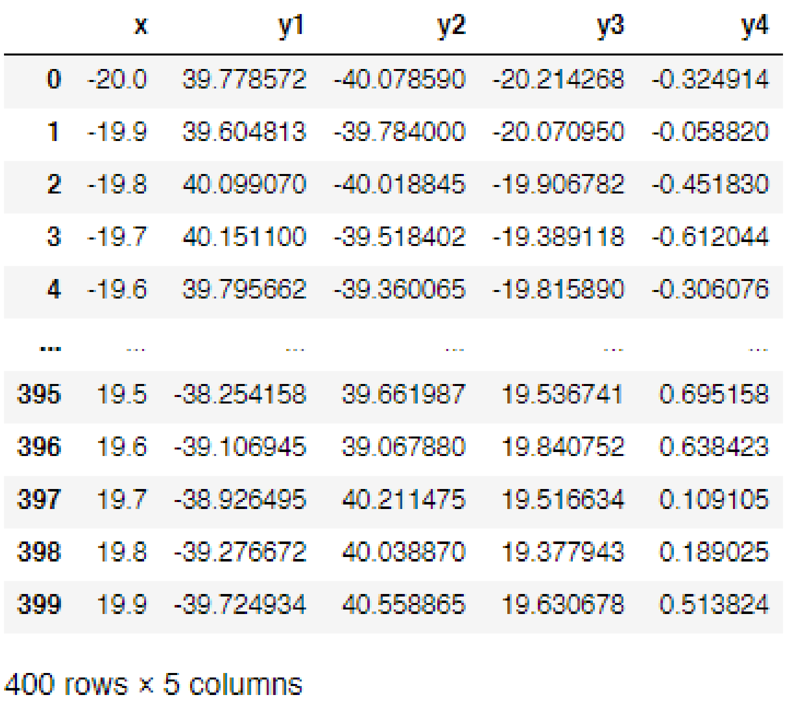


	x	y
0	17.5	34.161040
1	0.3	1.215102
2	-8.7	-16.843908
3	-19.2	-37.170870
4	-11.0	-20.263054
...	...	...
95	-1.9	-4.036904
96	12.2	-0.010358
97	16.5	-33.964134
98	5.3	-10.291622
99	17.9	28.078455

100 rows × 2 columns

Figure 2: Test File DataFrame

1.3. The compressed view of train dataset file to do a quick check of your DataFrame.



	x	y1	y2	y3	y4
0	-20.0	39.778572	-40.078590	-20.214268	-0.324914
1	-19.9	39.604813	-39.784000	-20.070950	-0.058820
2	-19.8	40.099070	-40.018845	-19.906782	-0.451830
3	-19.7	40.151100	-39.518402	-19.389118	-0.612044
4	-19.6	39.795662	-39.360065	-19.815890	-0.306076
...	...	...	...	...	...
395	19.5	-38.254158	39.661987	19.536741	0.695158
396	19.6	-39.106945	39.067880	19.840752	0.638423
397	19.7	-38.926495	40.211475	19.516634	0.109105
398	19.8	-39.276672	40.038870	19.377943	0.189025
399	19.9	-39.724934	40.558865	19.630678	0.513824

400 rows × 5 columns

Figure 3: Train File DataFrame

1.1.2 Shape of data.

1. Shape_of_data(df_data): Returns the shape of the dataFrame passed to it.
This code calculates and prints the shape of the DataFrame, which includes the number of rows and columns (Geeks for Geeks, 2023). The shape property is used to return a tuple representing the dimensionality of the Pandas DataFrame.

Python-program:
```
print("Return: A tuple, which is the shape of the dataframe.")
df_shape= dataWrangler.shape_of_data(df_data)
print(df_shape)
```

1.1 Ideal dataframe tuple.

Number of rows	Number of columns
400	51

Table 2: Shape of Ideal DataFrame

1.2 Test dataframe tuple.

Number of rows	Number of columns
100	2

Table 3: Shape of Test DataFrame

1.3 Train dataframe tuple.

Number of rows	Number of columns
400	5

Table 4: Shape of Train DataFrame

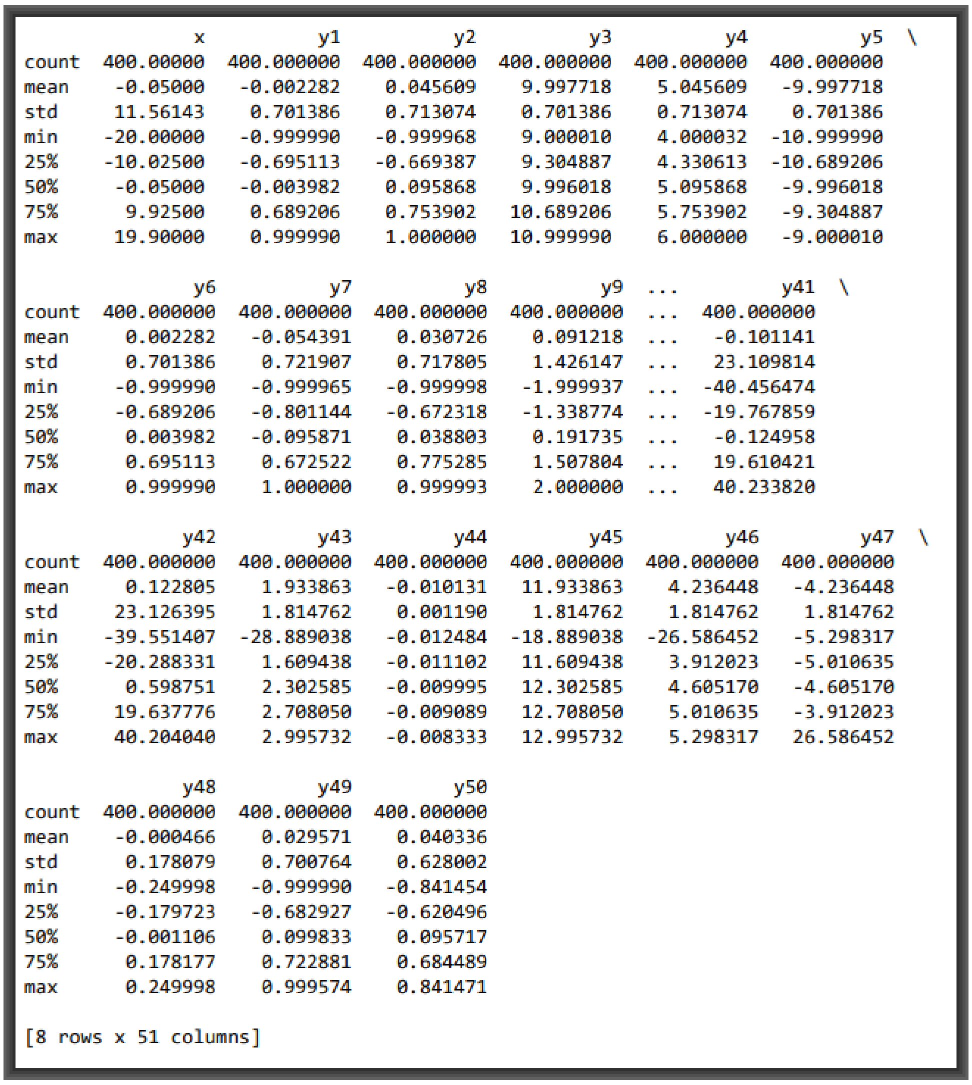
1.1.3 Summary statistics.

1. Summary_statistics(column_data): Returns a summary statistics of pandas dataframe column passed to it.
Summary statistics are used in all branches of math and science that employ statistics. These include probability, economics, biology, psychology, and astronomy. Summary statistics measures central tendency, dispersion, and correlation combined with descriptions of shape that provide a simple overview of a data set or data sets. (Story of Mathematics, 2024). Description of all three dataframes is explained by statistical values such as count, mean, standard deviation, minimum, maximum and quartiles. Additionally, other measures include mode, range, and correlation coefficient. It is advisable to interpret individual components of summary statistics taking into consideration the other components. For example, a larger range and larger standard deviation indicate a wider dispersion. A wider range with a smaller standard deviation indicates outliers. Descriptions of the overall data shape such as "normally distributed", "skewed left" or "skewed right" are also part of summary statistics.

Measures of central tendency with a mean that is higher than the median indicates a skew to the right. On the other hand, a mean that is less than the median indicates a skew to the left. If the mean and median are the same, the data set is normally distributed. Summary statistics give a small "snapshot" of a data set that is simply approachable than large quantities of data and more easily generalized than random data points. Similar to the summary of a story, they analyze and describe even large data sets in just a few numbers and words (Story of Mathematics, 2024).

Python-program:
```
print(”Computes the summary statistic of each column of the dataframe.”)
print(”Return: A dataframe, which gives a summary statistic of each column in the dataframe.”)
column_data = dataWrangler.summary_statistics(df_data)
print(column_data)
```

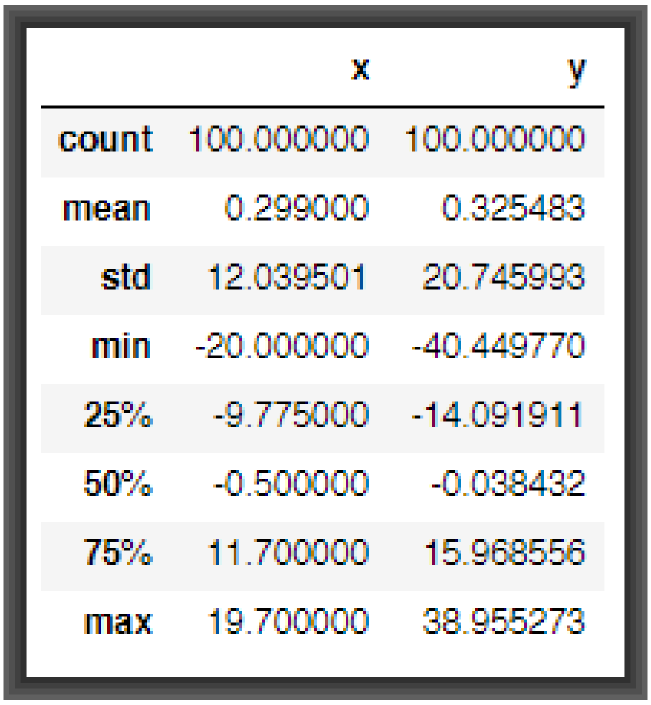
1.1. Summary statistics of each column in the Ideal dataframe.



```
               x          y1          y2          y3          y4          y5  \
count  400.00000  400.000000  400.000000  400.000000  400.000000  400.000000
mean    -0.05000   -0.002282    0.045609    9.997718    5.045609   -9.997718
std     11.56143    0.701386    0.713074    0.701386    0.713074    0.701386
min    -20.00000   -0.999990   -0.999968    9.000010    4.000032  -10.999990
25%    -10.02500   -0.695113   -0.669387    9.304887    4.330613  -10.689206
50%     -0.05000   -0.003982    0.095868    9.996018    5.095868   -9.996018
75%      9.92500    0.689206    0.753902   10.689206    5.753902   -9.304887
max     19.90000    0.999990    1.000000   10.999990    6.000000   -9.000010

               y6          y7          y8          y9  ...         y41  \
count  400.000000  400.000000  400.000000  400.000000  ...  400.000000
mean     0.002282   -0.054391    0.030726    0.091218  ...   -0.101141
std      0.701386    0.721907    0.717805    1.426147  ...   23.109814
min     -0.999990   -0.999965   -0.999998   -1.999937  ...  -40.456474
25%     -0.689206   -0.801144   -0.672318   -1.338774  ...  -19.767859
50%      0.003982   -0.095871    0.038803    0.191735  ...   -0.124958
75%      0.695113    0.672522    0.775285    1.507804  ...   19.610421
max      0.999990    1.000000    0.999993    2.000000  ...   40.233820

              y42         y43         y44         y45         y46         y47  \
count  400.000000  400.000000  400.000000  400.000000  400.000000  400.000000
mean     0.122805    1.933863   -0.010131   11.933863    4.236448   -4.236448
std     23.126395    1.814762    0.001190    1.814762    1.814762    1.814762
min    -39.551407  -28.889038   -0.012484  -18.889038  -26.586452   -5.298317
25%    -20.288331    1.609438   -0.011102   11.609438    3.912023   -5.010635
50%      0.598751    2.302585   -0.009995   12.302585    4.605170   -4.605170
75%     19.637776    2.708050   -0.009089   12.708050    5.010635   -3.912023
max     40.204040    2.995732   -0.008333   12.995732    5.298317   26.586452

              y48         y49         y50
count  400.000000  400.000000  400.000000
mean    -0.000466    0.029571    0.040336
std      0.178079    0.700764    0.628002
min     -0.249998   -0.999990   -0.841454
25%     -0.179723   -0.682927   -0.620496
50%     -0.001106    0.099833    0.095717
75%      0.178177    0.722881    0.684489
max      0.249998    0.999574    0.841471

[8 rows x 51 columns]
```

Figure 4: Summary Statistics of Ideal DataFrame Columns.

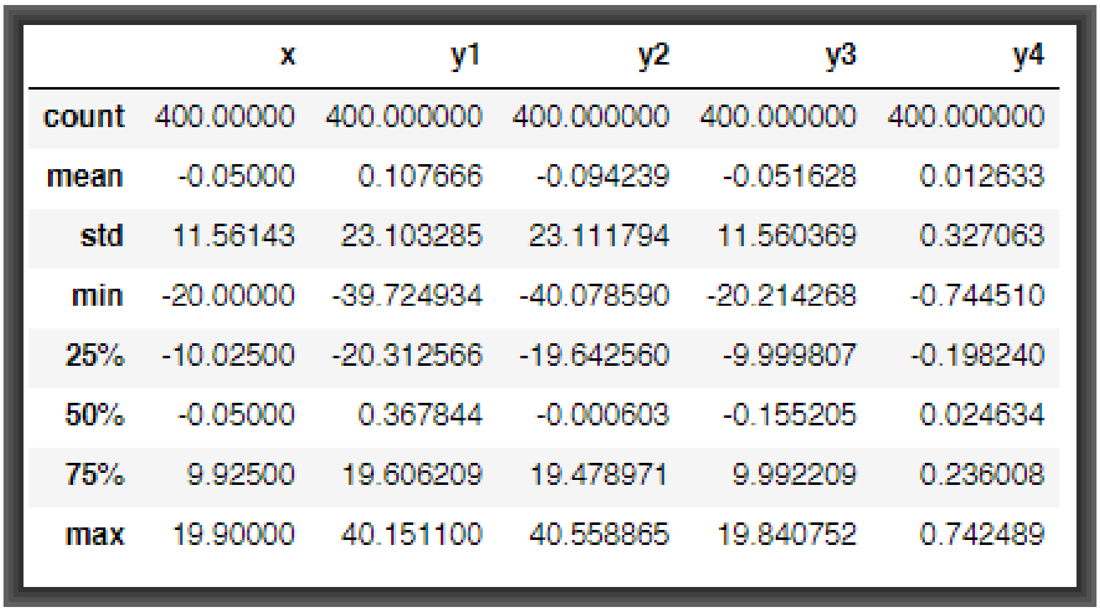
1.2. Summary statistics of each column in the Test dataframe.



	x	y
count	100.000000	100.000000
mean	0.299000	0.325483
std	12.039501	20.745993
min	-20.000000	-40.449770
25%	-9.775000	-14.091911
50%	-0.500000	-0.038432
75%	11.700000	15.968556
max	19.700000	38.955273

Figure 5: Summary Statistics of Test DataFrame Columns.

1.3. Summary statistics of each column in the Train dataframe.



	x	y1	y2	y3	y4
count	400.00000	400.000000	400.000000	400.000000	400.000000
mean	-0.05000	0.107666	-0.094239	-0.051628	0.012633
std	11.56143	23.103285	23.111794	11.560369	0.327063
min	-20.00000	-39.724934	-40.078590	-20.214268	-0.744510
25%	-10.02500	-20.312566	-19.642560	-9.999807	-0.198240
50%	-0.05000	0.367844	-0.000603	-0.155205	0.024634
75%	9.92500	19.606209	19.478971	9.992209	0.236008
max	19.90000	40.151100	40.558865	19.840752	0.742489

Figure 6: Summary Statistics of Train DataFrame Columns.

1.1.4 Find missing values.

i. Find_missing_values(df_data): Returns a list of missing values (nan) by columns.
Explore various top techniques to handle missing values and their implementations in Python. Investigating missing data is a common and inherent issue in data collection that result in working with large datasets (Keita, 2023). Statistical data have various reasons for missing values because participants can provide incomplete information or decline to share information, poor designed survey is not easy to answer, and/ or removal of data for confidentiality reasons. Missing data can bias the conclusions of all the statistical analyses on the data if it is not appropriately handled. This can lead to wrong business decision-making. Therefore, we will illustrate the benefits and drawbacks of each technique used to handle missing values to help you choose the right one for a given situation. There are different formats in which missing data occurs. This section explains the different types of missing data and how to identify them (Keita, 2023).

1. **MCAR - Missing completely at random.**
   This occurs if all the variables and observations have the same probability of being missing.

2. **MAR - Missing at random.**
   The probability of the value being missing is related to the value of the variable or other variables in the dataset. All of the observations and variables do not have the same chance of being missing.

3. **MNAR- Missing not at random**
   MNAR is considered to be the most difficult scenario among the three types of missing data. It is used when neither MAR nor MCAR apply. For this reason, the probability of being missing is completely different for different values of the same variable, and these reasons can be unknown to us.

We have several methods for identifying missing data in python (Keita, 2023).
There are multiple methods that can be used to identify missing data in pandas:

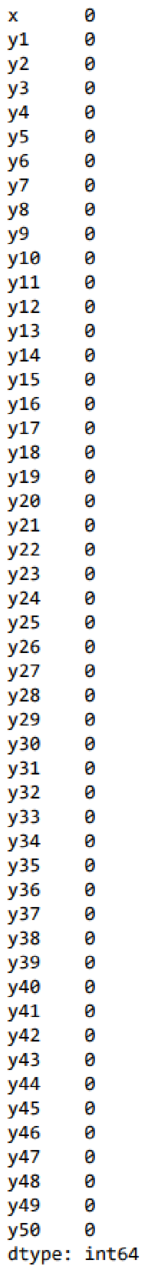
Functions	**Descriptions**
.isnull()	This function returns a pandas dataframe, where each value is a boolean value True if the value is missing, False otherwise.
.notnull()	Similarly to the previous function, the values for this one are False if either NaN or None value is detected.
.info()	This function generates three main columns, including the “Non-Null Count” which shows the number of non-missing values for each column.
.isna()	This one is similar to isnull and notnull. However it shows True only when the missing value is NaN type.

Table 5: Methods for identifying missing data.

Python-program:
print(”Computes the count of missing values in each column of the dataframe.”)
print(”Return: Returns a pandas series bearing the details on the count of missing values in each column of the dataframe.”)
df_result= dataWrangler.find_missing_values(df_data)
print(df_result)
print(” ”)

display(df_result)

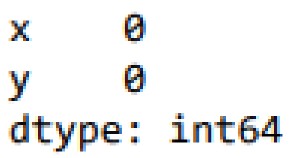
ii. Count missing values in each column of the Ideal dataframe.



```
x      0
y1     0
y2     0
y3     0
y4     0
y5     0
y6     0
y7     0
y8     0
y9     0
y10    0
y11    0
y12    0
y13    0
y14    0
y15    0
y16    0
y17    0
y18    0
y19    0
y20    0
y21    0
y22    0
y23    0
y24    0
y25    0
y26    0
y27    0
y28    0
y29    0
y30    0
y31    0
y32    0
y33    0
y34    0
y35    0
y36    0
y37    0
y38    0
y39    0
y40    0
y41    0
y42    0
y43    0
y44    0
y45    0
y46    0
y47    0
y48    0
y49    0
y50    0
dtype: int64
```

Figure 7: Count of Missing Values in Ideal DataFrame Columns.

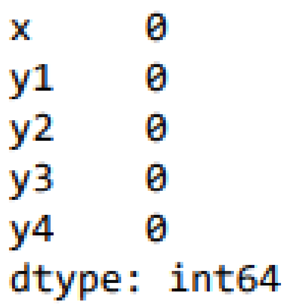
iii. Count missing values in each column of the Test dataframe.



```
x    0
y    0
dtype: int64
```

Figure 8: Count of Missing Values in Test DataFrame Columns.

iv. Count missing values in each column of the Train dataframe.



```
x     0
y1    0
y2    0
y3    0
y4    0
dtype: int64
```

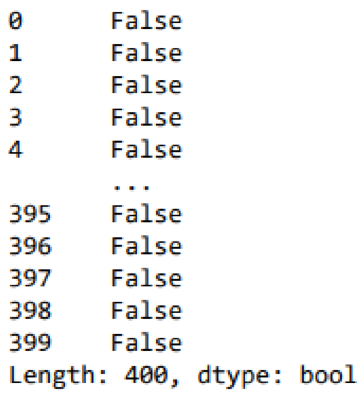
Figure 9: Count of Missing Values in Train DataFrame Columns.

1.1.5 Find duplicate rows.

1. Duplicated_rows(df_data): Boolean data type is used to return duplicated rows. If the result is True, it means you have duplicated rows. A False return means you do not have duplicated rows.
The duplicated rows found in a dataset have an impact on the results of data pre-processing and analysis. Therefore, you will often need to figure out whether you have duplicated data and how to deal with them (Chen, 2021). You should be able to find duplicated rows by using pandas method duplicated ().

Python-program:
```
print(”Finds all duplicated rows in the dataframe.”)
print(”Return: Returns a dataframe bearing duplicated rows in the dataframe.”)
#df_duplicated= dataWrangler.duplicated_rows()
df_duplicated = df_data.duplicated()
print(” ”)
print(df_duplicated)
print(” ”)
display(df_result)
```

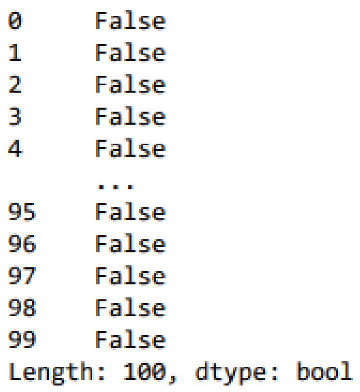
1.1. Find all duplicated rows in the Ideal dataframe.



```
0      False
1      False
2      False
3      False
4      False
       ...
395    False
396    False
397    False
398    False
399    False
Length: 400, dtype: bool
```

Figure 10: Status of Duplicated Rows in Ideal DataFrame.

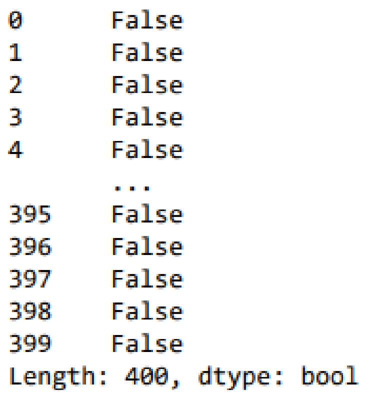
1.2. Find all duplicated rows in the Test dataframe.



```
0     False
1     False
2     False
3     False
4     False
      ...
95    False
96    False
97    False
98    False
99    False
Length: 100, dtype: bool
```

Figure 11: Status of Duplicated Rows in Test DataFrame.

1.3. Find all duplicated rows in the Train dataframe.



```
0      False
1      False
2      False
3      False
4      False
       ...
395    False
396    False
397    False
398    False
399    False
Length: 400, dtype: bool
```

Figure 12: Status of Duplicated Rows in Train DataFrame.

1.1.6 Drop duplicate rows.

1. Drop_duplicates(df_data): Returns a pandas dataframe with duplicated rows dropped.
After finding duplicate rows, you should remove them using pandas method drop_duplicates (). It is important to determine which duplicates to mark with keep (Chen, 2021). Please note, the code dropped duplicate rows and keep is equal to the first row.

Python-program:
```
print(”Drops all duplicated rows in the dataframe and keeps the first occurrence.”)
print(”Return: Returns a dataframe bearing no duplicated rows”)
print(” ”)
#df_data = dataWrangler.drop_duplicated()
df_data = df_data.drop_duplicates(keep=”first”)
print(df_data)
```

1.1. Drop all of the duplicated rows in the Test dataframe and keep the first occurrence. This returns a Test dataframe without duplicated rows.

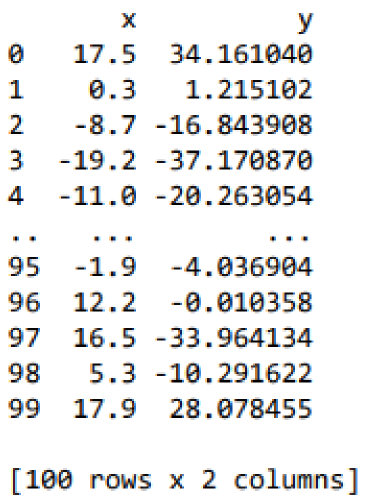
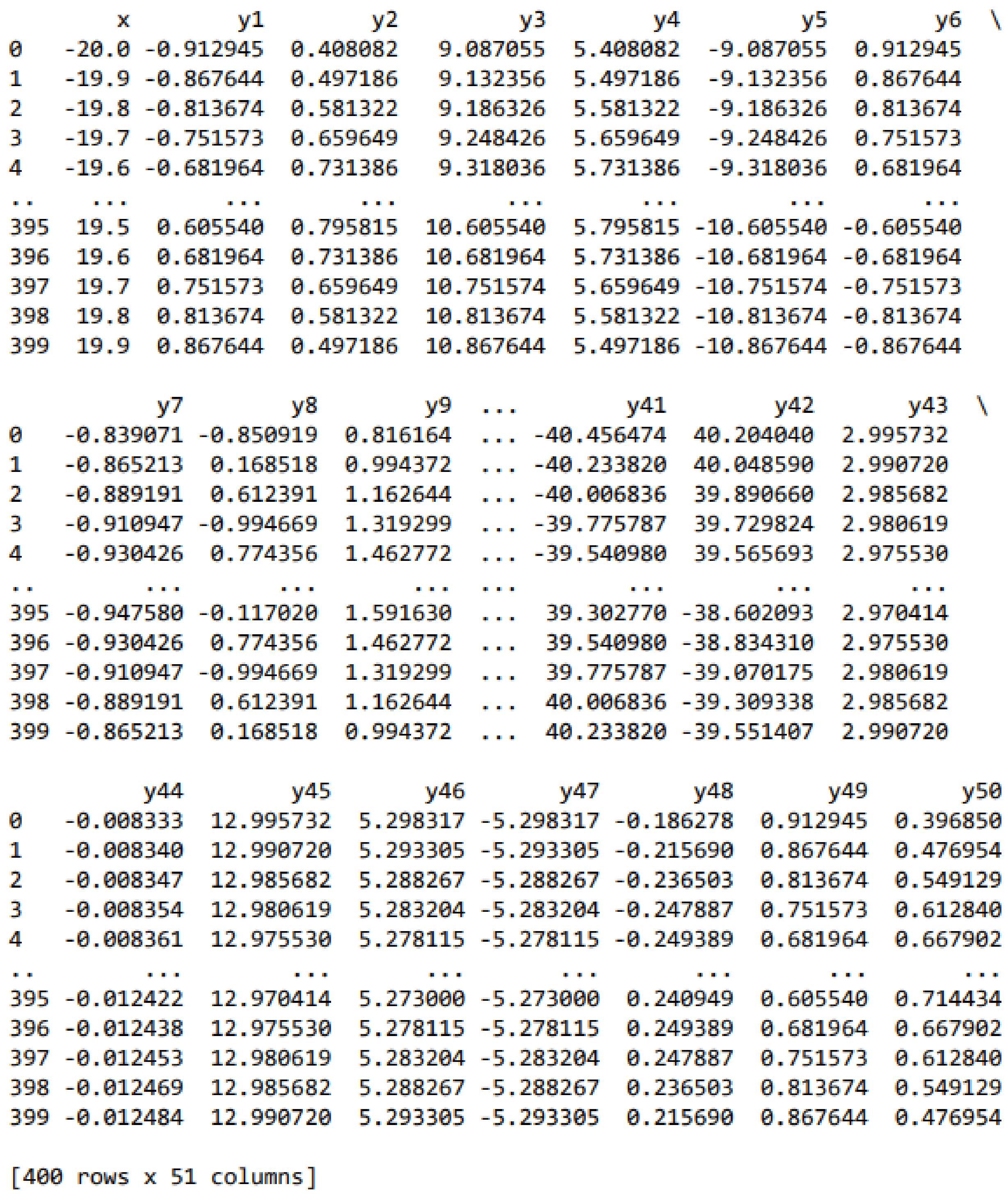

```
       x          y
0   17.5  34.161040
1    0.3   1.215102
2   -8.7 -16.843908
3  -19.2 -37.170870
4  -11.0 -20.263054
..   ...        ...
95  -1.9  -4.036904
96  12.2  -0.010358
97  16.5 -33.964134
98   5.3 -10.291622
99  17.9  28.078455

[100 rows x 2 columns]
```

Figure 13: Test DataFrame without Duplicated Rows.

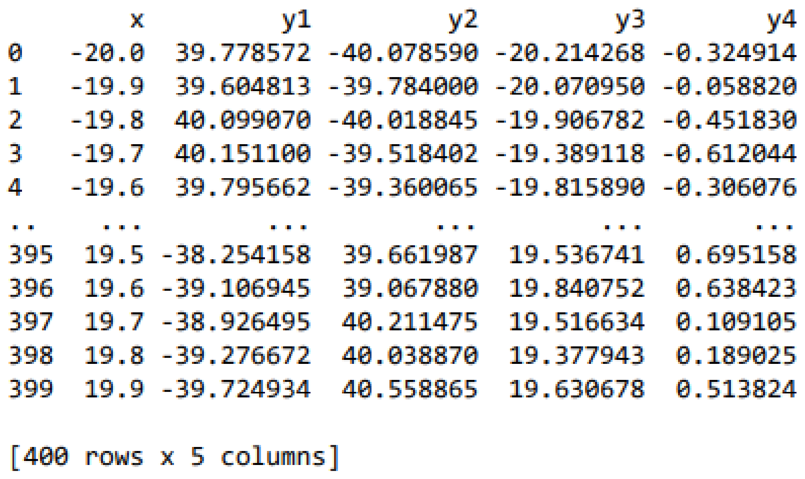
1.2. Drop all of the duplicated rows in the Ideal dataframe and keep the first occurrence. This returns an Ideal dataframe without duplicated rows.



```
         x        y1        y2         y3        y4         y5        y6  \
0    -20.0 -0.912945  0.408082   9.087055  5.408082  -9.087055  0.912945
1    -19.9 -0.867644  0.497186   9.132356  5.497186  -9.132356  0.867644
2    -19.8 -0.813674  0.581322   9.186326  5.581322  -9.186326  0.813674
3    -19.7 -0.751573  0.659649   9.248426  5.659649  -9.248426  0.751573
4    -19.6 -0.681964  0.731386   9.318036  5.731386  -9.318036  0.681964
..     ...       ...       ...        ...       ...        ...       ...
395   19.5  0.605540  0.795815  10.605540  5.795815 -10.605540 -0.605540
396   19.6  0.681964  0.731386  10.681964  5.731386 -10.681964 -0.681964
397   19.7  0.751573  0.659649  10.751574  5.659649 -10.751574 -0.751573
398   19.8  0.813674  0.581322  10.813674  5.581322 -10.813674 -0.813674
399   19.9  0.867644  0.497186  10.867644  5.497186 -10.867644 -0.867644

           y7        y8        y9  ...        y41        y42       y43  \
0   -0.839071 -0.850919  0.816164  ... -40.456474  40.204040  2.995732
1   -0.865213  0.168518  0.994372  ... -40.233820  40.048590  2.990720
2   -0.889191  0.612391  1.162644  ... -40.006836  39.890660  2.985682
3   -0.910947 -0.994669  1.319299  ... -39.775787  39.729824  2.980619
4   -0.930426  0.774356  1.462772  ... -39.540980  39.565693  2.975530
..        ...       ...       ...  ...        ...        ...       ...
395 -0.947580 -0.117020  1.591630  ...  39.302770 -38.602093  2.970414
396 -0.930426  0.774356  1.462772  ...  39.540980 -38.834310  2.975530
397 -0.910947 -0.994669  1.319299  ...  39.775787 -39.070175  2.980619
398 -0.889191  0.612391  1.162644  ...  40.006836 -39.309338  2.985682
399 -0.865213  0.168518  0.994372  ...  40.233820 -39.551407  2.990720

          y44        y45       y46       y47       y48       y49       y50
0   -0.008333  12.995732  5.298317 -5.298317 -0.186278  0.912945  0.396850
1   -0.008340  12.990720  5.293305 -5.293305 -0.215690  0.867644  0.476954
2   -0.008347  12.985682  5.288267 -5.288267 -0.236503  0.813674  0.549129
3   -0.008354  12.980619  5.283204 -5.283204 -0.247887  0.751573  0.612840
4   -0.008361  12.975530  5.278115 -5.278115 -0.249389  0.681964  0.667902
..        ...        ...       ...       ...       ...       ...       ...
395 -0.012422  12.970414  5.273000 -5.273000  0.240949  0.605540  0.714434
396 -0.012438  12.975530  5.278115 -5.278115  0.249389  0.681964  0.667902
397 -0.012453  12.980619  5.283204 -5.283204  0.247887  0.751573  0.612840
398 -0.012469  12.985682  5.288267 -5.288267  0.236503  0.813674  0.549129
399 -0.012484  12.990720  5.293305 -5.293305  0.215690  0.867644  0.476954

[400 rows x 51 columns]
```

Figure 14: Ideal DataFrame without Duplicated Rows.

1.3. Drop all of the duplicated rows in the Train dataframe and keep the first occurrence. This returns a Train dataframe without duplicated rows.



```
        x         y1         y2         y3        y4
0   -20.0  39.778572 -40.078590 -20.214268 -0.324914
1   -19.9  39.604813 -39.784000 -20.070950 -0.058820
2   -19.8  40.099070 -40.018845 -19.906782 -0.451830
3   -19.7  40.151100 -39.518402 -19.389118 -0.612044
4   -19.6  39.795662 -39.360065 -19.815890 -0.306076
..    ...        ...        ...        ...       ...
395  19.5 -38.254158  39.661987  19.536741  0.695158
396  19.6 -39.106945  39.067880  19.840752  0.638423
397  19.7 -38.926495  40.211475  19.516634  0.109105
398  19.8 -39.276672  40.038870  19.377943  0.189025
399  19.9 -39.724934  40.558865  19.630678  0.513824

[400 rows x 5 columns]
```

Figure 15: Train DataFrame without Duplicated Rows.

1.1.7 Fill missing values.

1. Fill_missing_values(df_data): Fills missing values (nan) in the dataframe with the mean value of each column as indicated by summary statistics. All of the Ideal, Test and Train dataframes does not have missing values. This means that you cannot replace the missing values (nan) with the calculated mean values.

1.1.8 Find outliers.

Find_outliers(df_column): Returns all values that are outliers in the numeric column passed to it. An outlier is a data point on a graph or in a set of results that is very much bigger or smaller than the next nearest data point. Outliers are values at the extreme ends of a dataset (Bhandari, 2024). These extreme values differ from most other data points in a dataset. They can have a very big impact on your statistical analysis and skew the results of any hypothesis tests. It is recommended to identify and deal with potential outliers in an appropriate manner for accurate results.

Inter-quartile range (IQR) method is used to identify outliers in this chapter. It tells you the range of the middle half of your dataset. The function of IQR is to create lower and upper bounds around your dataset. This method define outliers as any values that are less than lower bound or greater than upper bound. The figure below shows a box plot of IQR with two lower fence outliers and two upper fence outliers.

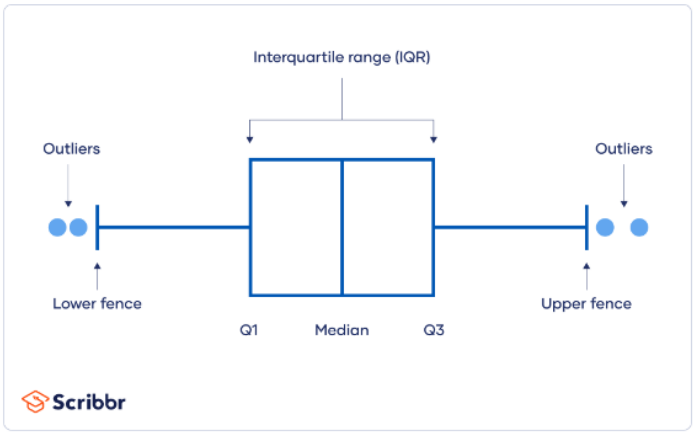
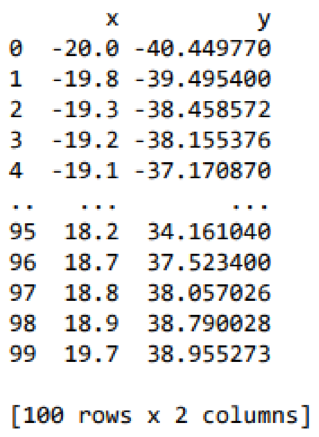


Figure 16: Inter-quartile Range (IQR) with Outliers (Bhandari, 2024).

Inter-quartile range method for Ideal, Test and Train datasets.
1. Sort your data from low to high.

Python-program:
#Sort values in each column.
sorted_df_data = df_data.apply(lambda s: s.sort_values().values)
print("DataFrame with sorted values in each column:")
print(sorted_df_data)

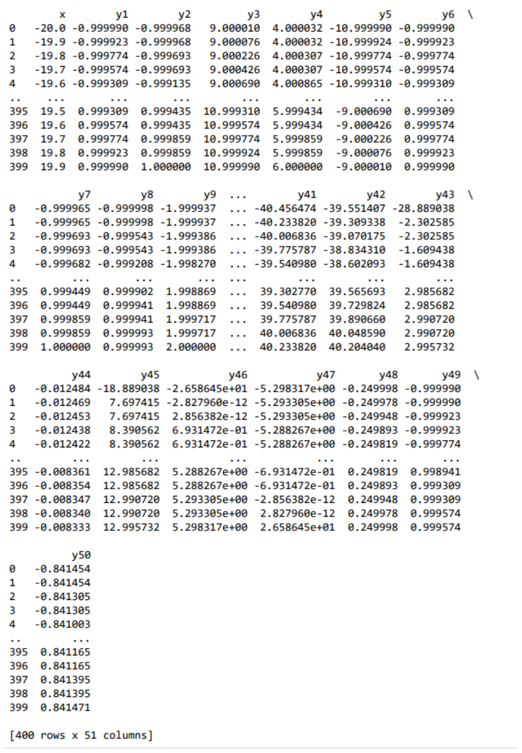
1.1. Test dataframe with sorted values in each column.



```
       x          y
0  -20.0 -40.449770
1  -19.8 -39.495400
2  -19.3 -38.458572
3  -19.2 -38.155376
4  -19.1 -37.170870
..   ...        ...
95  18.2  34.161040
96  18.7  37.523400
97  18.8  38.057026
98  18.9  38.790028
99  19.7  38.955273

[100 rows x 2 columns]
```

Figure 17: Low to high sorted column values in Test dataframe.

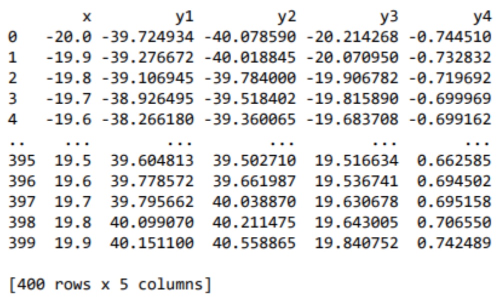
1.2. Ideal dataframe with sorted values in each column.



```
         x        y1        y2         y3        y4         y5        y6  \
0    -20.0 -0.999990 -0.999968   9.000010  4.000032 -10.999990 -0.999990
1    -19.9 -0.999923 -0.999968   9.000076  4.000032 -10.999924 -0.999923
2    -19.8 -0.999774 -0.999693   9.000226  4.000307 -10.999774 -0.999774
3    -19.7 -0.999574 -0.999693   9.000426  4.000307 -10.999574 -0.999574
4    -19.6 -0.999309 -0.999135   9.000690  4.000865 -10.999310 -0.999309
..     ...       ...       ...        ...       ...        ...       ...
395   19.5  0.999309  0.999435  10.999310  5.999434  -9.000690  0.999309
396   19.6  0.999574  0.999435  10.999574  5.999434  -9.000426  0.999574
397   19.7  0.999774  0.999859  10.999774  5.999859  -9.000226  0.999774
398   19.8  0.999923  0.999859  10.999924  5.999859  -9.000076  0.999923
399   19.9  0.999990  1.000000  10.999990  6.000000  -9.000010  0.999990

           y7        y8        y9  ...        y41        y42        y43  \
0   -0.999965 -0.999998 -1.999937  ... -40.456474 -39.551407 -28.889038
1   -0.999965 -0.999998 -1.999937  ... -40.233820 -39.309338  -2.302585
2   -0.999693 -0.999543 -1.999386  ... -40.006836 -39.070175  -2.302585
3   -0.999693 -0.999543 -1.999386  ... -39.775787 -38.834310  -1.609438
4   -0.999682 -0.999208 -1.998270  ... -39.540980 -38.602093  -1.609438
..        ...       ...       ...  ...        ...        ...        ...
395  0.999449  0.999902  1.998869  ...  39.302770  39.565693   2.985682
396  0.999449  0.999941  1.998869  ...  39.540980  39.729824   2.985682
397  0.999859  0.999941  1.999717  ...  39.775787  39.890660   2.990720
398  0.999859  0.999993  1.999717  ...  40.006836  40.048590   2.990720
399  1.000000  0.999993  2.000000  ...  40.233820  40.204040   2.995732

          y44        y45           y46           y47       y48       y49  \
0   -0.012484 -18.889038 -2.658645e+01 -5.298317e+00 -0.249998 -0.999990
1   -0.012469   7.697415 -2.827960e-12 -5.293305e+00 -0.249978 -0.999990
2   -0.012453   7.697415  2.856382e-12 -5.293305e+00 -0.249948 -0.999923
3   -0.012438   8.390562  6.931472e-01 -5.288267e+00 -0.249893 -0.999923
4   -0.012422   8.390562  6.931472e-01 -5.288267e+00 -0.249819 -0.999774
..        ...        ...           ...           ...       ...       ...
395 -0.008361  12.985682  5.288267e+00 -6.931472e-01  0.249819  0.998941
396 -0.008354  12.985682  5.288267e+00 -6.931472e-01  0.249893  0.999309
397 -0.008347  12.990720  5.293305e+00 -2.856382e-12  0.249948  0.999309
398 -0.008340  12.990720  5.293305e+00  2.827960e-12  0.249978  0.999574
399 -0.008333  12.995732  5.298317e+00  2.658645e+01  0.249998  0.999574

          y50
0   -0.841454
1   -0.841454
2   -0.841305
3   -0.841305
4   -0.841003
..        ...
395  0.841165
396  0.841165
397  0.841395
398  0.841395
399  0.841471

[400 rows x 51 columns]
```

Figure 18: Low to high sorted column values in Ideal dataframe.

1.3. Train dataframe with sorted values in each column.



```
        x         y1         y2         y3        y4
0   -20.0 -39.724934 -40.078590 -20.214268 -0.744510
1   -19.9 -39.276672 -40.018845 -20.070950 -0.732832
2   -19.8 -39.106945 -39.784000 -19.906782 -0.719692
3   -19.7 -38.926495 -39.518402 -19.815890 -0.699969
4   -19.6 -38.266180 -39.360065 -19.683708 -0.699162
..    ...        ...        ...        ...       ...
395  19.5  39.604813  39.502710  19.516634  0.662585
396  19.6  39.778572  39.661987  19.536741  0.694502
397  19.7  39.795662  40.038870  19.630678  0.695158
398  19.8  40.099070  40.211475  19.643005  0.706550
399  19.9  40.151100  40.558865  19.840752  0.742489

[400 rows x 5 columns]
```

Figure 19: Low to high sorted column values in Train dataframe.

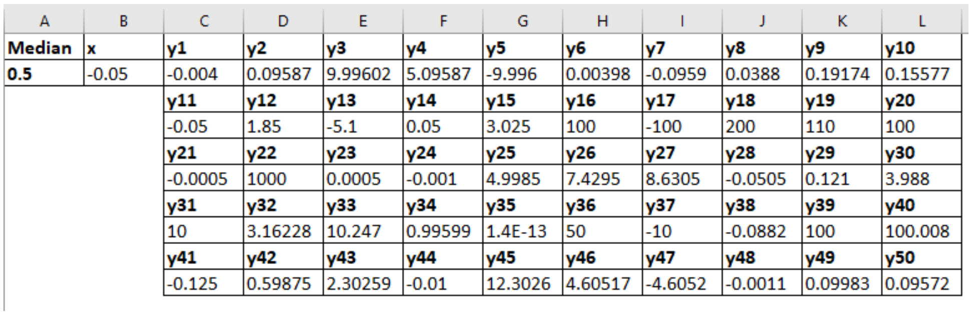
2. Identify the first quartile (Q1), the median (Q2), and the third quartile (Q3). Median is the value in the middle of your dataset when all values are ordered from low to high.(Bhandari, 2024). First quartile (Q1) is the value in the middle of the first half of your dataset. Third quartile (Q3) value is in the middle of the second half of your dataset. The median (Q2) is the value that divides an ordered list of data values in half.

Python-program:
print(”Median”)
q2 = df_data.quantile([0.5])
print(q2)
print(” ”)
display(q2)
# Determining the name of the excel file. Remember to change file_name.

```
dfq2_excel = 'Q2_Ideal.xlsx'
# Saving to excel file.
q2.to_excel(dfq2_excel)
print('DataFrame is written to Excel File successfully.')

print( "First Quartile.")
q1 = df_data.quantile([0.25])
print(q1)
print(" ")
display(q1)
# Determining the name of the excel file. Remember to change file_name.
dfq1_excel = 'Q1_Test.xlsx'
# Saving to excel file.
q1.to_excel(dfq1_excel)
print('DataFrame is written to Excel File successfully.')

print("Third Quartile.")
q3 = df_data.quantile([0.75])
print(q3)
print(" ")
display(q3)
# Determining the name of the excel file. Remember to change file_name.
dfq3_excel = 'Q3_Train.xlsx'
# Saving to excel file.
q3.to_excel(dfq3_excel)
print('DataFrame is written to Excel File successfully.')
```

2.1. Median, first quartile (Q1), and third quartile (Q3) for Ideal dataset.

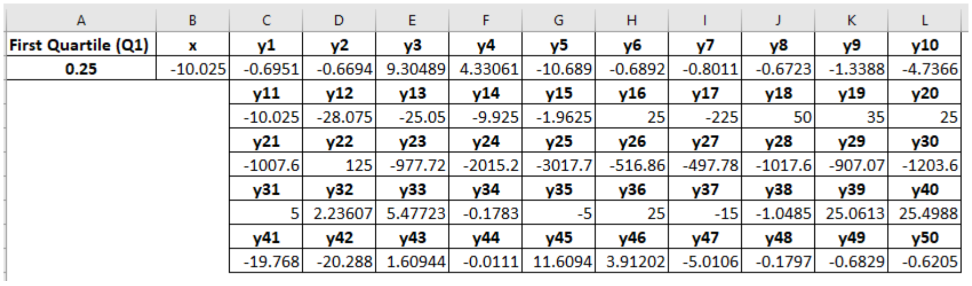
2.1.1. Median



A	B	C	D	E	F	G	H	I	J	K	L
Median	**x**	**y1**	**y2**	**y3**	**y4**	**y5**	**y6**	**y7**	**y8**	**y9**	**y10**
0.5	-0.05	-0.004	0.09587	9.99602	5.09587	-9.996	0.00398	-0.0959	0.0388	0.19174	0.15577
		y11	**y12**	**y13**	**y14**	**y15**	**y16**	**y17**	**y18**	**y19**	**y20**
		-0.05	1.85	-5.1	0.05	3.025	100	-100	200	110	100
		y21	**y22**	**y23**	**y24**	**y25**	**y26**	**y27**	**y28**	**y29**	**y30**
		-0.0005	1000	0.0005	-0.001	4.9985	7.4295	8.6305	-0.0505	0.121	3.988
		y31	**y32**	**y33**	**y34**	**y35**	**y36**	**y37**	**y38**	**y39**	**y40**
		10	3.16228	10.247	0.99599	1.4E-13	50	-10	-0.0882	100	100.008
		y41	**y42**	**y43**	**y44**	**y45**	**y46**	**y47**	**y48**	**y49**	**y50**
		-0.125	0.59875	2.30259	-0.01	12.3026	4.60517	-4.6052	-0.0011	0.09983	0.09572

Figure 20: Median of each column in the Ideal dataframe.

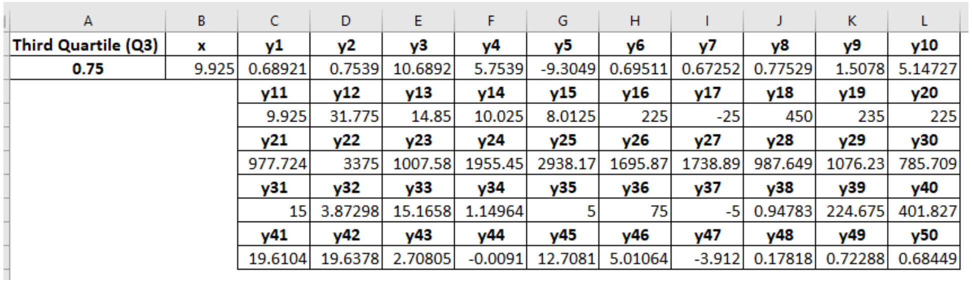
2.1.2. First Quartile (Q1).



A	B	C	D	E	F	G	H	I	J	K	L
First Quartile (Q1)	**x**	**y1**	**y2**	**y3**	**y4**	**y5**	**y6**	**y7**	**y8**	**y9**	**y10**
0.25	-10.025	-0.6951	-0.6694	9.30489	4.33061	-10.689	-0.6892	-0.8011	-0.6723	-1.3388	-4.7366
		y11	**y12**	**y13**	**y14**	**y15**	**y16**	**y17**	**y18**	**y19**	**y20**
		-10.025	-28.075	-25.05	-9.925	-1.9625	25	-225	50	35	25
		y21	**y22**	**y23**	**y24**	**y25**	**y26**	**y27**	**y28**	**y29**	**y30**
		-1007.6	125	-977.72	-2015.2	-3017.7	-516.86	-497.78	-1017.6	-907.07	-1203.6
		y31	**y32**	**y33**	**y34**	**y35**	**y36**	**y37**	**y38**	**y39**	**y40**
		5	2.23607	5.47723	-0.1783	-5	25	-15	-1.0485	25.0613	25.4988
		y41	**y42**	**y43**	**y44**	**y45**	**y46**	**y47**	**y48**	**y49**	**y50**
		-19.768	-20.288	1.60944	-0.0111	11.6094	3.91202	-5.0106	-0.1797	-0.6829	-0.6205

Figure 21: First Quartile of each column in the Ideal dataframe.

2.1.3. Third Quartile (Q3).

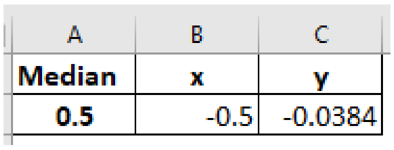


A	B	C	D	E	F	G	H	I	J	K	L
Third Quartile (Q3)	**x**	**y1**	**y2**	**y3**	**y4**	**y5**	**y6**	**y7**	**y8**	**y9**	**y10**
0.75	9.925	0.68921	0.7539	10.6892	5.7539	-9.3049	0.69511	0.67252	0.77529	1.5078	5.14727
		y11	**y12**	**y13**	**y14**	**y15**	**y16**	**y17**	**y18**	**y19**	**y20**
		9.925	31.775	14.85	10.025	8.0125	225	-25	450	235	225
		y21	**y22**	**y23**	**y24**	**y25**	**y26**	**y27**	**y28**	**y29**	**y30**
		977.724	3375	1007.58	1955.45	2938.17	1695.87	1738.89	987.649	1076.23	785.709
		y31	**y32**	**y33**	**y34**	**y35**	**y36**	**y37**	**y38**	**y39**	**y40**
		15	3.87298	15.1658	1.14964	5	75	-5	0.94783	224.675	401.827
		y41	**y42**	**y43**	**y44**	**y45**	**y46**	**y47**	**y48**	**y49**	**y50**
		19.6104	19.6378	2.70805	-0.0091	12.7081	5.01064	-3.912	0.17818	0.72288	0.68449

Figure 22: Third Quartile of each column in the Ideal dataframe.

2.2. Median, first quartile (Q1), and third quartile (Q3) for Test dataset.

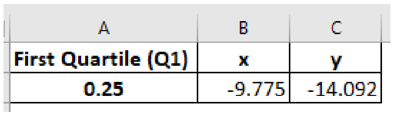
2.2.1. Median



A	B	C
Median	**x**	**y**
0.5	-0.5	-0.0384

Figure 23: Median of each column in the Test dataframe.

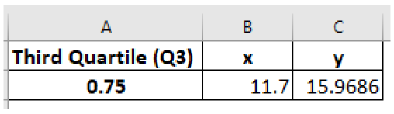
2.2.2. First Quartile (Q1).



A	B	C
First Quartile (Q1)	**x**	**y**
0.25	-9.775	-14.092

Figure 24: First Quartile of each column in the Test dataframe.

2.2.3. Third Quartile (Q3).

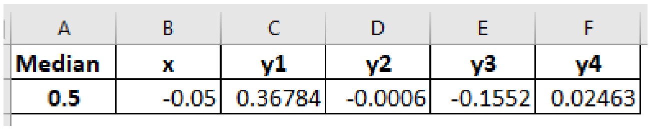


A	B	C
Third Quartile (Q3)	x	y
0.75	11.7	15.9686

Figure 25: Third Quartile of each column in the Test dataframe.

2.3. Median, first quartile (Q1), and third quartile (Q3) for Train dataset.

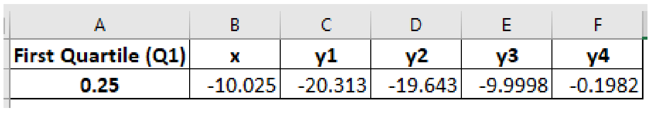
2.3.1. Median



A	B	C	D	E	F
Median	x	y1	y2	y3	y4
0.5	-0.05	0.36784	-0.0006	-0.1552	0.02463

Figure 26: Median of each column in the Train dataframe.

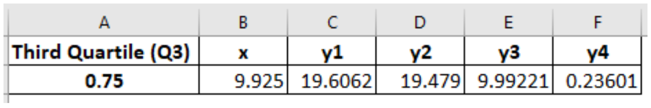
2.3.2. First Quartile (Q1).



A	B	C	D	E	F
First Quartile (Q1)	x	y1	y2	y3	y4
0.25	-10.025	-20.313	-19.643	-9.9998	-0.1982

Figure 27: First Quartile of each column in the Train dataframe.

2.3.3. Third Quartile (Q3).

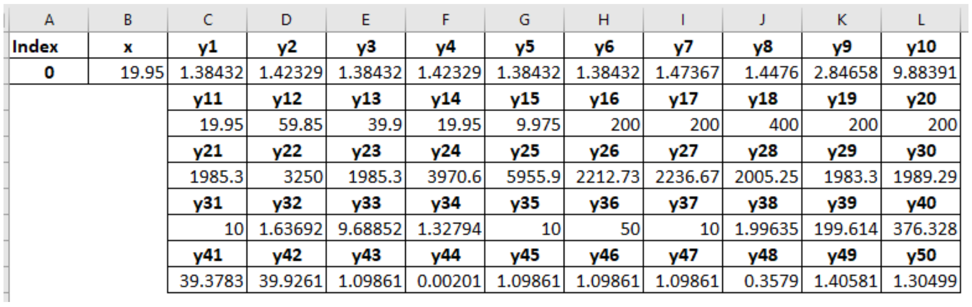


A	B	C	D	E	F
Third Quartile (Q3)	x	y1	y2	y3	y4
0.75	9.925	19.6062	19.479	9.99221	0.23601

Figure 28: Third Quartile of each column in the Train dataframe.

3. Calculate your IQR = Q3 – Q1.

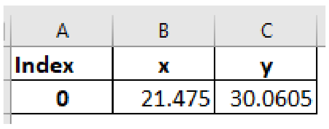
3.1. Inter-quartile range (IQR) for Ideal dataframe.



A	B	C	D	E	F	G	H	I	J	K	L
Index	x	y1	y2	y3	y4	y5	y6	y7	y8	y9	y10
0	19.95	1.38432	1.42329	1.38432	1.42329	1.38432	1.38432	1.47367	1.4476	2.84658	9.88391
		y11	y12	y13	y14	y15	y16	y17	y18	y19	y20
		19.95	59.85	39.9	19.95	9.975	200	200	400	200	200
		y21	y22	y23	y24	y25	y26	y27	y28	y29	y30
		1985.3	3250	1985.3	3970.6	5955.9	2212.73	2236.67	2005.25	1983.3	1989.29
		y31	y32	y33	y34	y35	y36	y37	y38	y39	y40
		10	1.63692	9.68852	1.32794	10	50	10	1.99635	199.614	376.328
		y41	y42	y43	y44	y45	y46	y47	y48	y49	y50
		39.3783	39.9261	1.09861	0.00201	1.09861	1.09861	1.09861	0.3579	1.40581	1.30499

Figure 29: IQR of each Column in Ideal DataFrame.

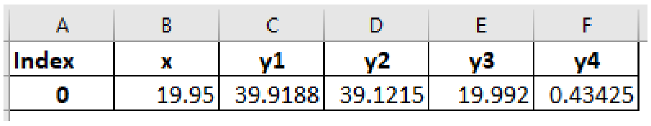
3.2. Inter-quartile range (IQR) for Test dataframe.



A	B	C
Index	x	y
0	21.475	30.0605

Figure 30: IQR of each Column in Test DataFrame.

3.3. Inter-quartile range (IQR) for Train dataframe.

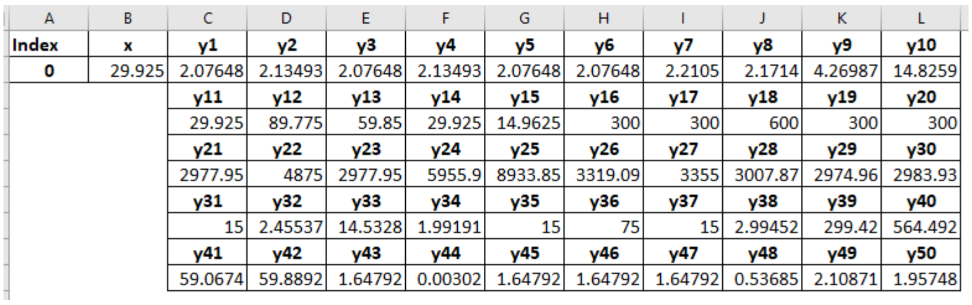


A	B	C	D	E	F
Index	x	y1	y2	y3	y4
0	19.95	39.9188	39.1215	19.992	0.43425

Figure 31: IQR of each Column in Train DataFrame.

4. Multiply your inter-quartile range (IQR) with a coeficient of 1.5.

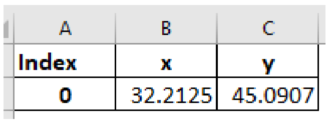
4.1. Ideal dataframe with IQR times 1.5 coeficient.



A	B	C	D	E	F	G	H	I	J	K	L
Index	x	y1	y2	y3	y4	y5	y6	y7	y8	y9	y10
0	29.925	2.07648	2.13493	2.07648	2.13493	2.07648	2.07648	2.2105	2.1714	4.26987	14.8259
		y11	y12	y13	y14	y15	y16	y17	y18	y19	y20
		29.925	89.775	59.85	29.925	14.9625	300	300	600	300	300
		y21	y22	y23	y24	y25	y26	y27	y28	y29	y30
		2977.95	4875	2977.95	5955.9	8933.85	3319.09	3355	3007.87	2974.96	2983.93
		y31	y32	y33	y34	y35	y36	y37	y38	y39	y40
		15	2.45537	14.5328	1.99191	15	75	15	2.99452	299.42	564.492
		y41	y42	y43	y44	y45	y46	y47	y48	y49	y50
		59.0674	59.8892	1.64792	0.00302	1.64792	1.64792	1.64792	0.53685	2.10871	1.95748

Figure 32: Ideal IQR Multiplied by 1.5 Coeficient in each Column.

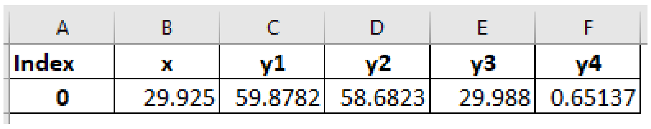
4.2. Test dataframe with IQR times 1.5 coeficient.



A	B	C
Index	x	y
0	32.2125	45.0907

Figure 33: Test IQR Multiplied by 1.5 Coeficient in each Column.

4.3. Train dataframe with IQR times 1.5 coeficient.

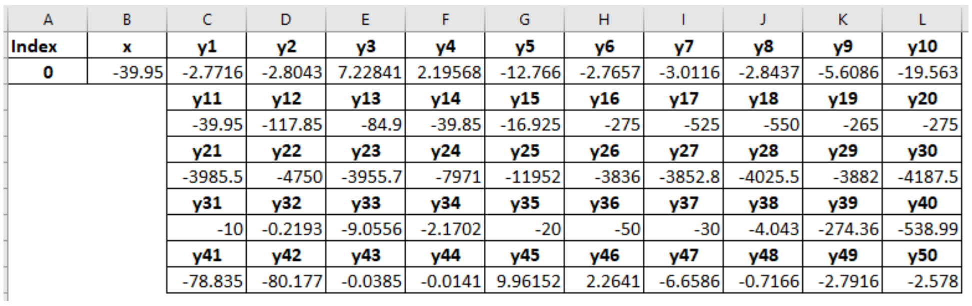


A	B	C	D	E	F
Index	x	y1	y2	y3	y4
0	29.925	59.8782	58.6823	29.988	0.65137

Figure 34: Train IQR Multiplied by 1.5 Coeficient in each Column.

5. Calculate your lower bound = Q1 - (1.5 * IQR).

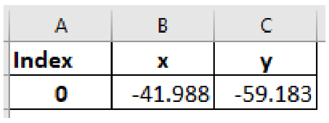
5.1. Lower bound for each column in Ideal dataframe.



A	B	C	D	E	F	G	H	I	J	K	L
Index	x	y1	y2	y3	y4	y5	y6	y7	y8	y9	y10
0	-39.95	-2.7716	-2.8043	7.22841	2.19568	-12.766	-2.7657	-3.0116	-2.8437	-5.6086	-19.563
		y11	y12	y13	y14	y15	y16	y17	y18	y19	y20
		-39.95	-117.85	-84.9	-39.85	-16.925	-275	-525	-550	-265	-275
		y21	y22	y23	y24	y25	y26	y27	y28	y29	y30
		-3985.5	-4750	-3955.7	-7971	-11952	-3836	-3852.8	-4025.5	-3882	-4187.5
		y31	y32	y33	y34	y35	y36	y37	y38	y39	y40
		-10	-0.2193	-9.0556	-2.1702	-20	-50	-30	-4.043	-274.36	-538.99
		y41	y42	y43	y44	y45	y46	y47	y48	y49	y50
		-78.835	-80.177	-0.0385	-0.0141	9.96152	2.2641	-6.6586	-0.7166	-2.7916	-2.578

Figure 35: Ideal Lower Bound in each Column.

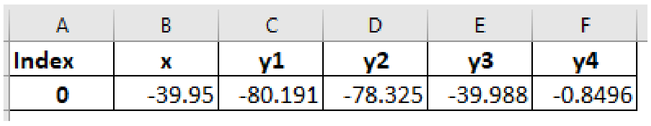
5.2. Lower bound for each column in Test dataframe.



A	B	C
Index	x	y
0	-41.988	-59.183

Figure 36: Test Lower Bound in each Column.

5.3. Lower bound for each column in Train dataframe.

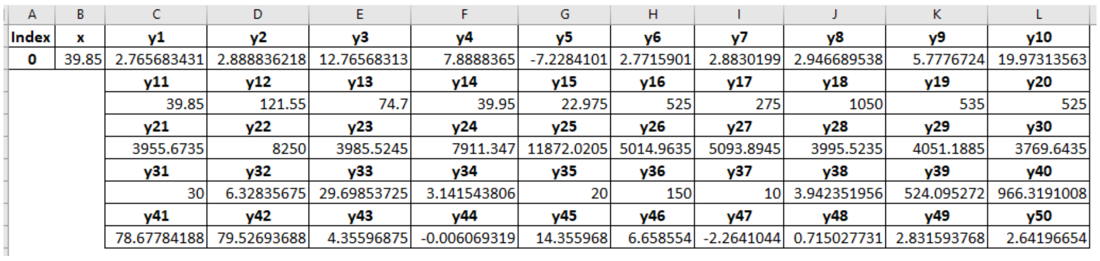


A	B	C	D	E	F
Index	x	y1	y2	y3	y4
0	-39.95	-80.191	-78.325	-39.988	-0.8496

Figure 37: Train Lower Bound in each Column.

6. Calculate your upper bound = Q3 + (1.5 * IQR).

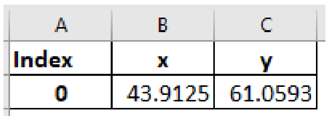
6.1. Upper bound for each column in Ideal dataframe.



A	B	C	D	E	F	G	H	I	J	K	L
Index	x	y1	y2	y3	y4	y5	y6	y7	y8	y9	y10
0	39.85	2.765683431	2.888836218	12.76568313	7.8888365	-7.2284101	2.7715901	2.8830199	2.946689538	5.7776724	19.97313563
		y11	y12	y13	y14	y15	y16	y17	y18	y19	y20
		39.85	121.55	74.7	39.95	22.975	525	275	1050	535	525
		y21	y22	y23	y24	y25	y26	y27	y28	y29	y30
		3955.6735	8250	3985.5245	7911.347	11872.0205	5014.9635	5093.8945	3995.5235	4051.1885	3769.6435
		y31	y32	y33	y34	y35	y36	y37	y38	y39	y40
		30	6.32835675	29.69853725	3.141543806	20	150	10	3.942351956	524.095272	966.3191008
		y41	y42	y43	y44	y45	y46	y47	y48	y49	y50
		78.67784188	79.52693688	4.35596875	-0.006069319	14.355968	6.658554	-2.2641044	0.715027731	2.831593768	2.64196654

Figure 38: Ideal Upper Bound in each Column.

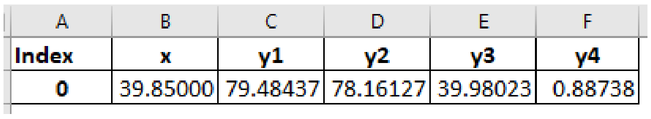
6.2. Upper bound for each column in Test dataframe.



A	B	C
Index	x	y
0	43.9125	61.0593

Figure 39: Test Upper Bound in each Column.

6.3. Upper bound for each column in Train dataframe.



A	B	C	D	E	F
Index	**x**	**y1**	**y2**	**y3**	**y4**
0	39.85000	79.48437	78.16127	39.98023	0.88738

Figure 40: Train Upper Bound in each Column.

7. Use your bounds to investigate any outliers, all values that fall outside your bounds.

7.1. List of Ideal Columns Box Plots.

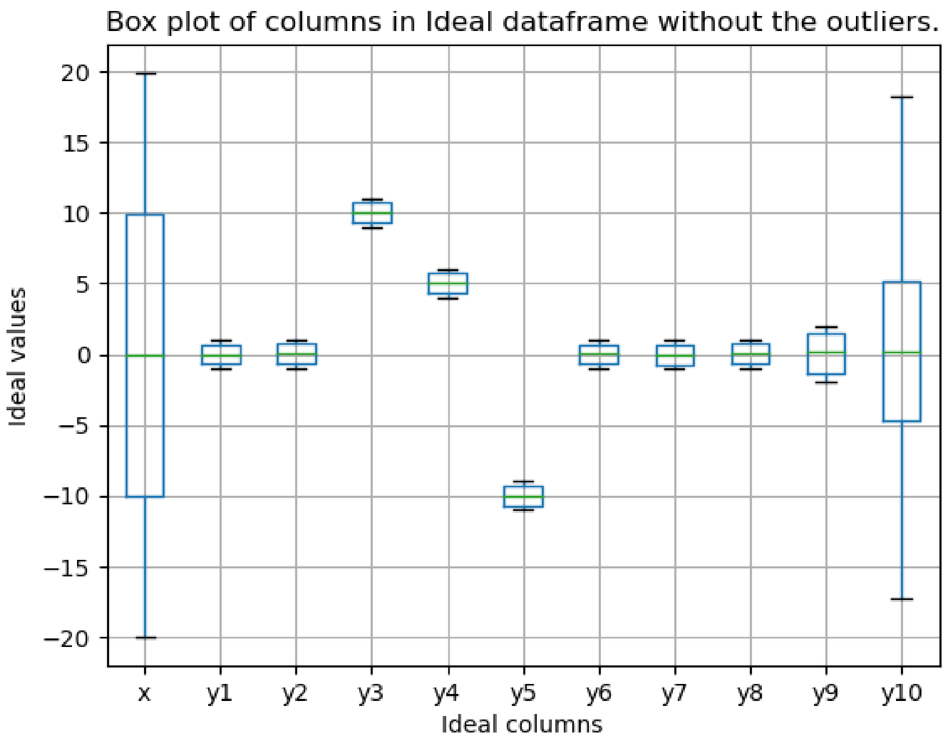


Figure 41: Box Plots of x - y10 Ideal Columns.

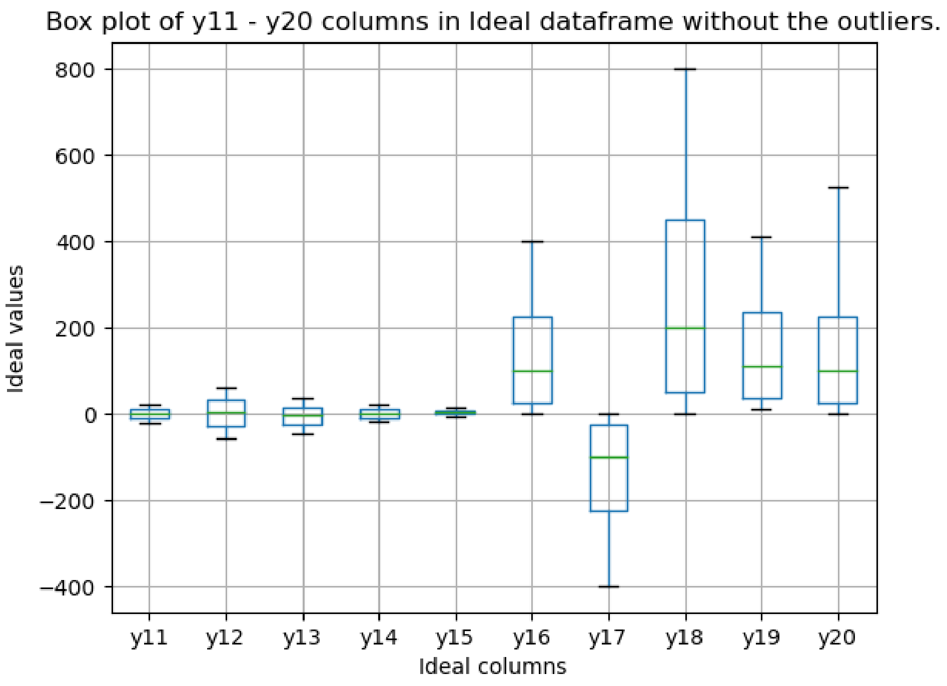


Figure 42: Box Plots of y11 - y20 Ideal Columns.

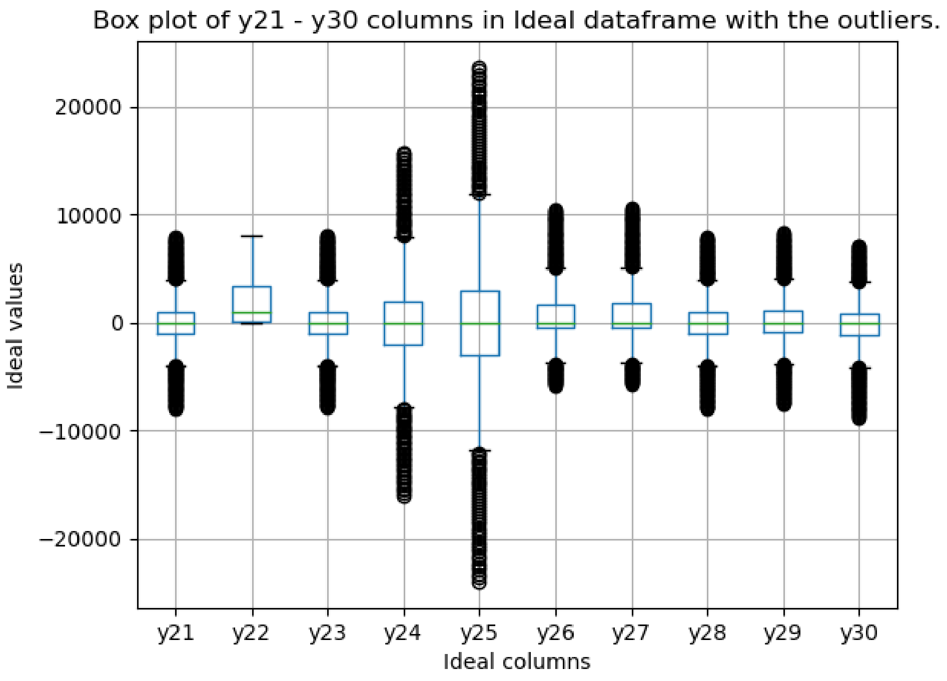


Figure 43: Box Plots of y21 - y30 Ideal Columns.

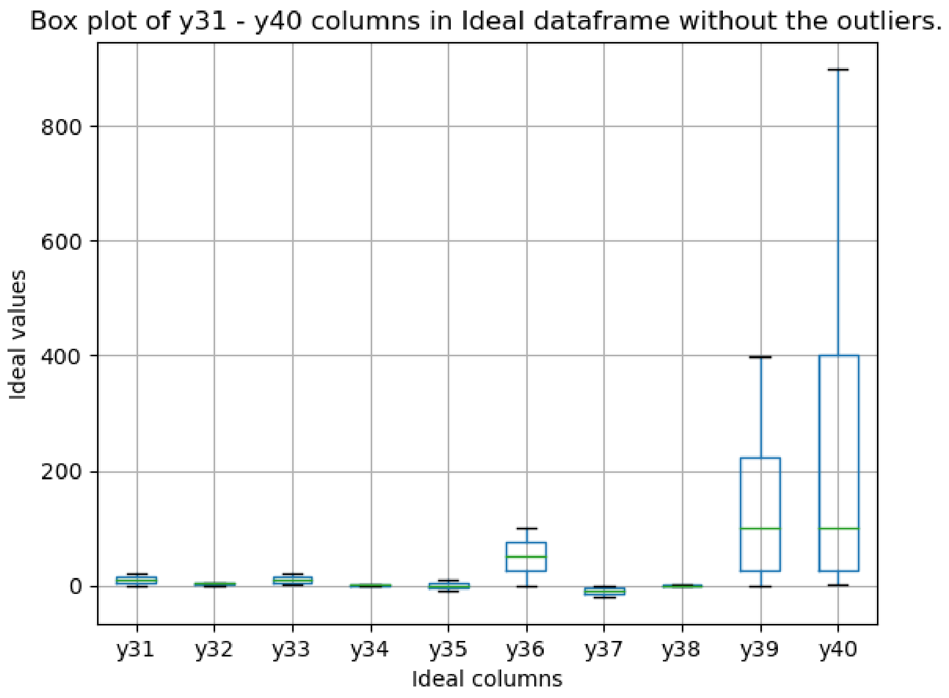


Figure 44: Box Plots of y31 - y40 Ideal Columns.

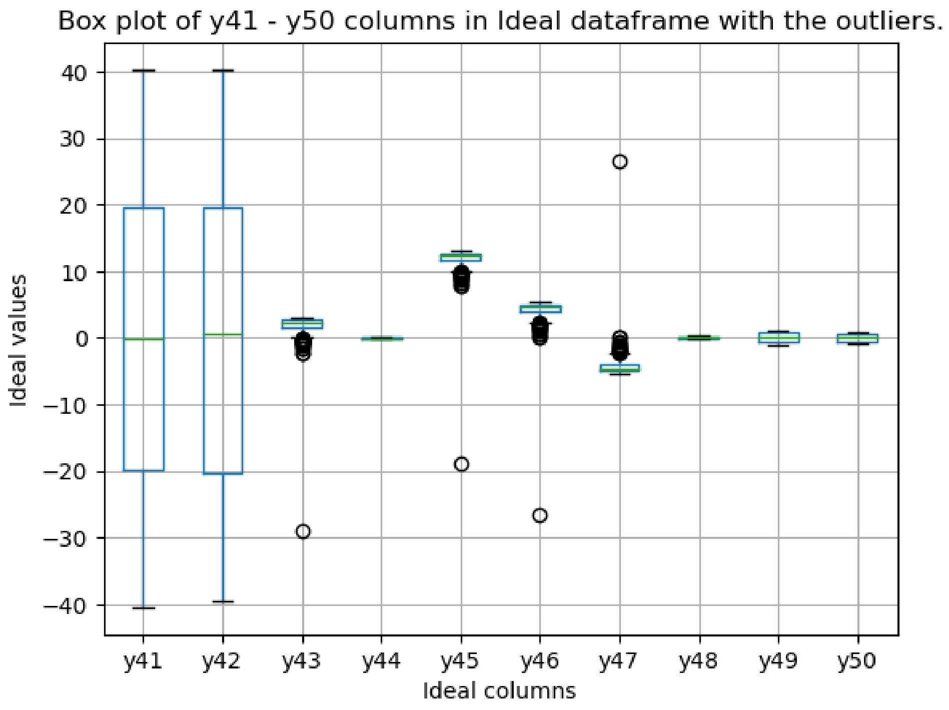


Figure 45: Box Plots of y41 - y50 Ideal Columns.

7.2. Test Columns Box Plots.

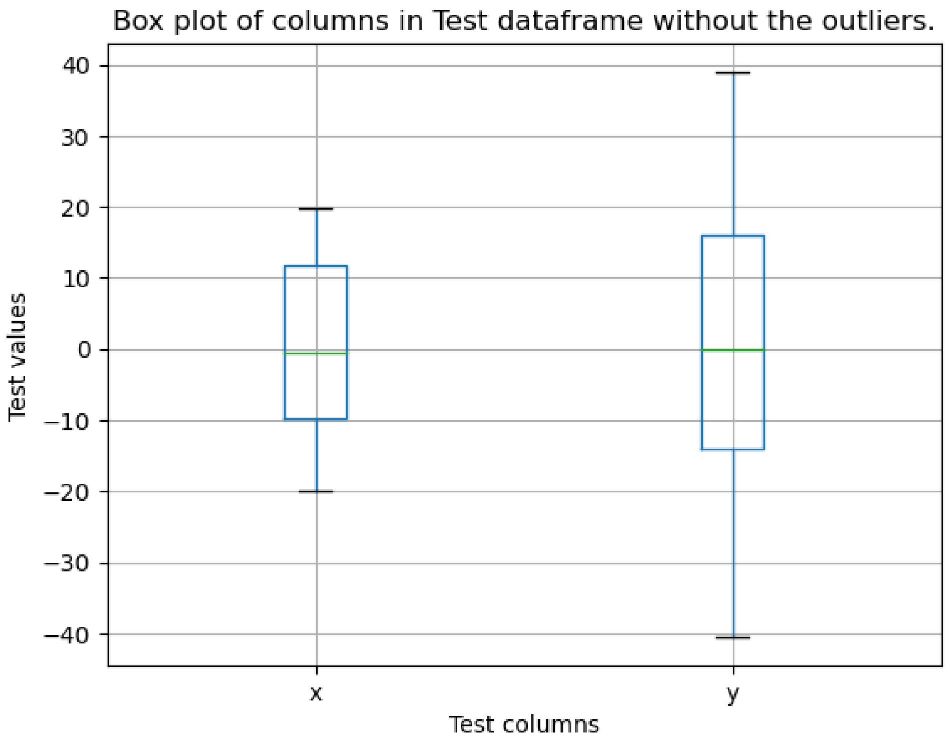


Figure 46: Box Plots of Test Columns.

7.3. List of Train Columns Box Plots.

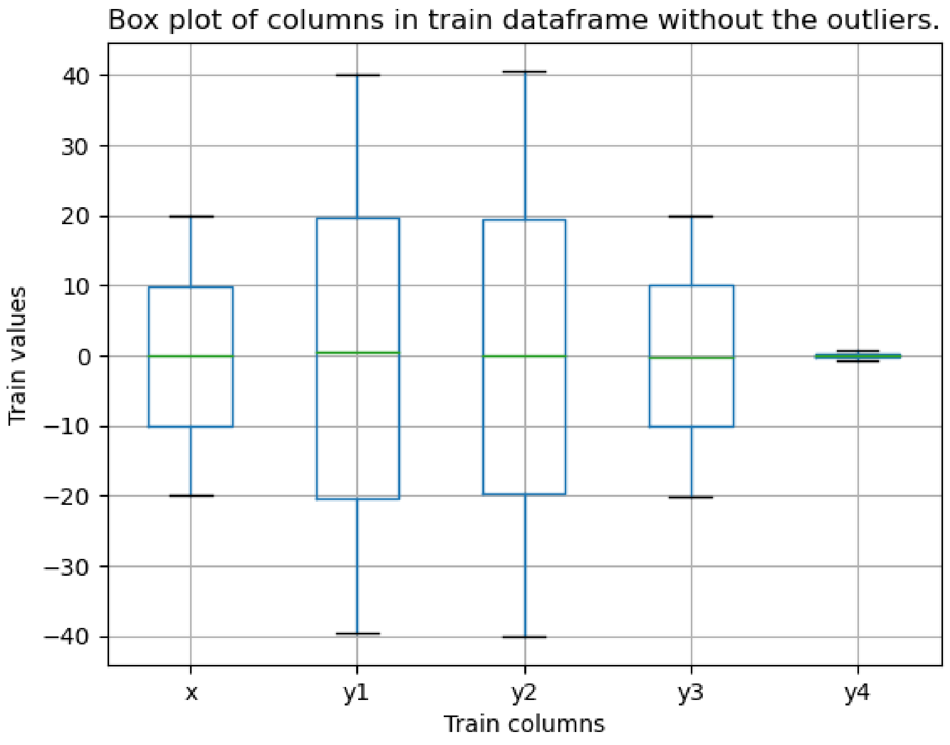


Figure 47: Box Plots of Train Columns.

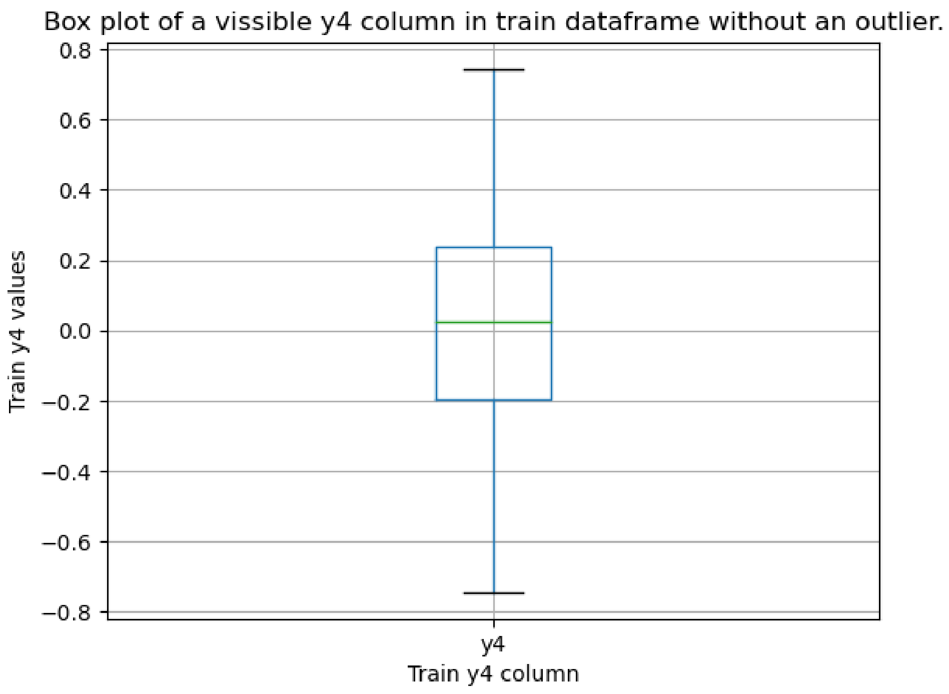


Figure 48: Box Plot of Y4 Train Column.

Line plots visualization of Test, Ideal and Train functions.

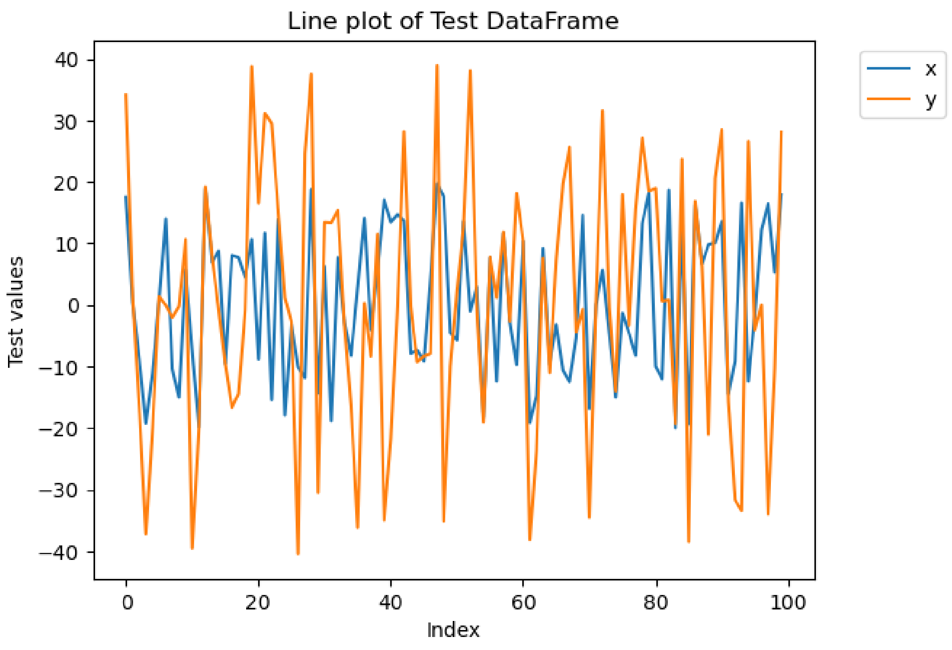


Figure 49: Line Plot of Test DataFrame.

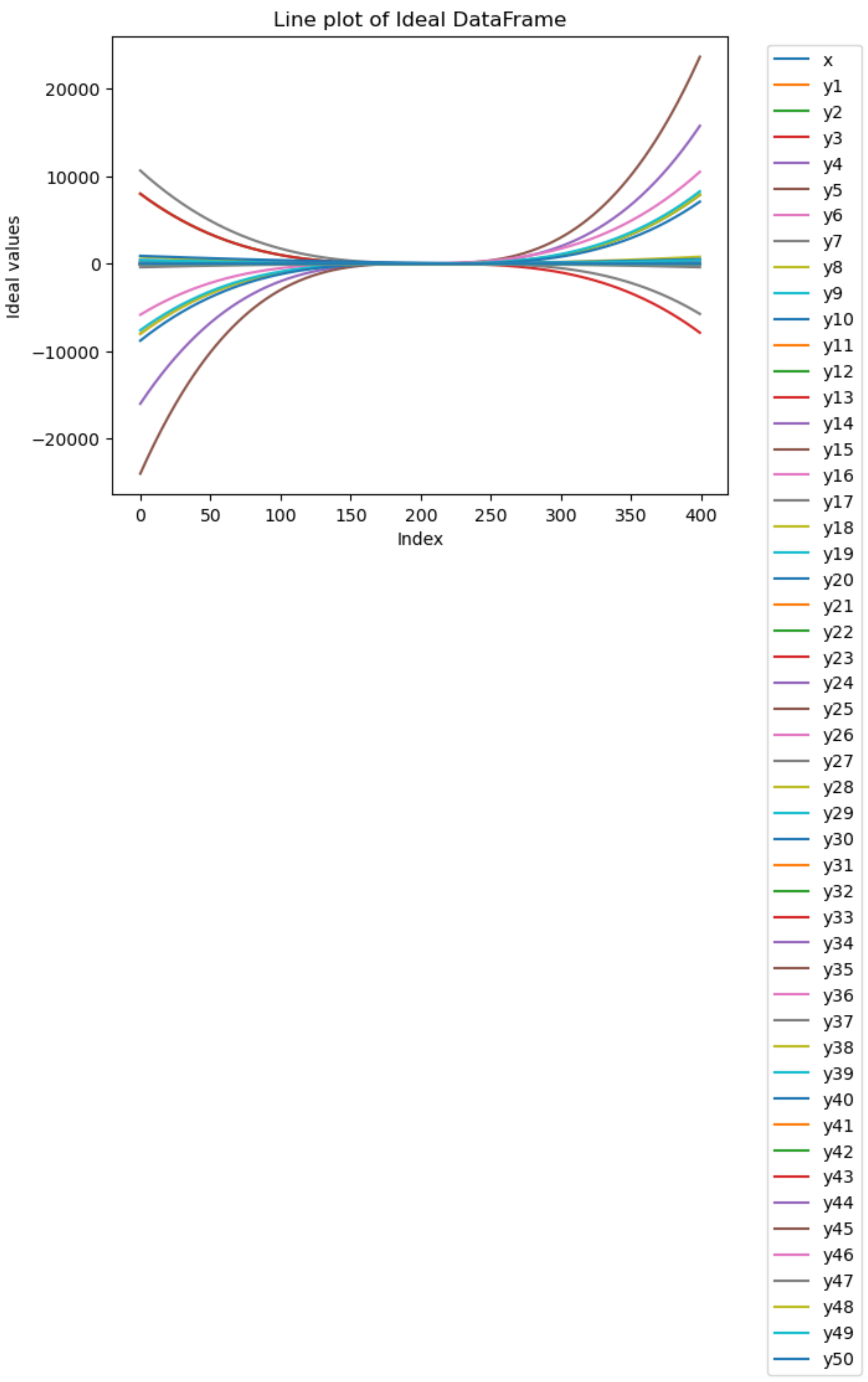


Figure 50: Line Plot of Ideal DataFrame.

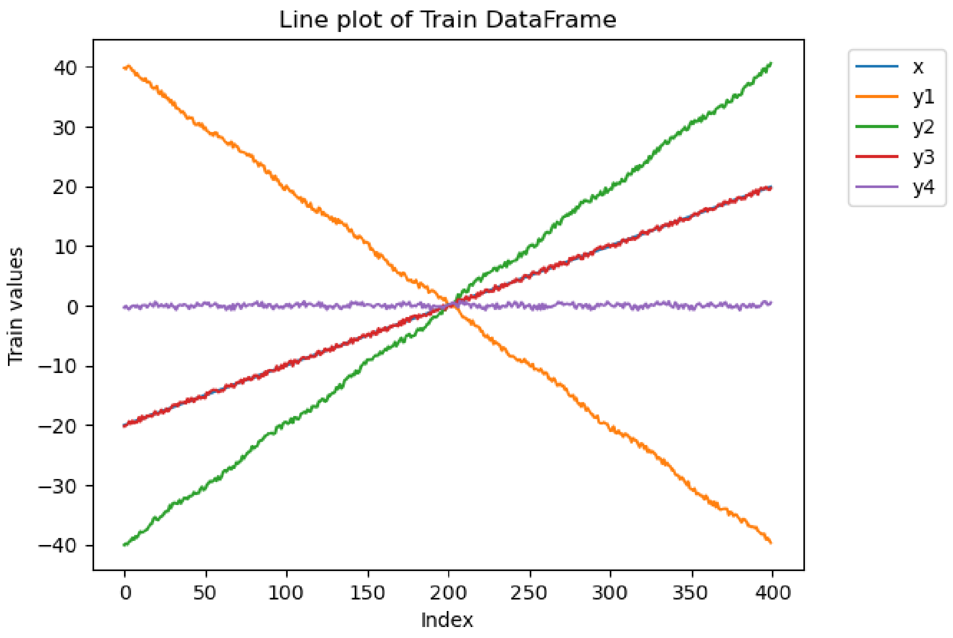


Figure 51: Line Plot of Train DataFrame.

2 Chapter 2 : Choose the Four Ideal Functions which are the Best Fit.

In this chapter, you will go through some parts of python-program that uses training data to choose the four ideal functions which are the best fit out of the fifty provided (C) *. The full program will be provided in the apendix.

Definition of Least Square Method.
Least Square method is a fundamental mathematical technique used in data analysis, statistics, and regression modeling to identify the best-fitting function for a given set of data points. The method ensures that the overall error is reduced by providing a highly accurate model for predicting future data trends (Geeks for Geeks, 2024). In addition, Least Square Method is used to derive a generalized linear equation between two variables. This is a statistical technique used to find the equation of best-fitting function to a set of data points by minimizing the sum of the squared differences between the original observed values and the values predicted by the model (Geeks for Geeks, 2024). A best fit function is defined as a function providing the best approximation of a given set of data. It is used to study the relationship between two variables. The best fit function is studied at two different levels (Varsity Tutors, 2024). In this task, four Ideal functions that matches with train functions are chosen using the definition of python function.
There are five steps to find best fit function:

1. Denote the independent variable values as x_i and the dependent ones as y_i.
2. Calculate the average values of x_i and y_i as X and Y.
3. Find the equation of the best fit function as $y = mx + c$, where m is the slope of the function and c represents the intercept of the function on the Y-axis.
4. The formula to calculate slope of m is equal: $m = \frac{\sum(X - x_i)*(Y - y_i)}{\sum(X - x_i)^2}$.
5. The intercept c is calculated as: $c = Y - mX$.

Therefore, you obtain best fit function as $y = mx + c$. Python program use built in methods to find least squares of the four best fit ideal functions based on train dataset. Mean Squared Error (MSE) is also calculated in the python code to help with a best forecast of choosing best fit ideal functions.

Mean Squared Error Definition.
The mean squared error (MSE) investigates how close a regression line is to a set of points. It performs this by taking the distances called errors from the points to the regression line and squaring them (Statistics How To, 2024). The square of errors is important to remove any negative signs. It also gives more weight to larger differences. Mean Squared Error refers to the average of a set of errors (Statistics How To, 2024).

Mean Squared Error Formula:

$MSE = \frac{\sum(Y_i - \hat{Y})^2}{n}$
Where:

- n = number of items.
- $\sum$ = summation notation.
- Y_i = actual original or observed y-value.
- $\hat{Y}$ = forecast y-value from regression.

General steps to calculate the MSE from a set of X and Y values:

1. Find the regression line.
2. Insert your X values into the linear regression equation to find the new Y values ($\hat{Y}$).
3. Subtract the new Y value ($\hat{Y}$) from the original y-value Y_i to get the error.
4. Square the errors.
5. Calculate sum of the errors.
6. Calculate the mean of the errors.

2.1 Choose ideal functions based on train data.

Python-program:

```
import sys # Standard library imports
import pandas as pd # related third party imports
import numpy as np
from matplotlib import pyplot as plt
from sqlalchemy import create_engine

class FindFunctions:
    def __init__(self):
        pass

    def find_ideal_matches(self, train_fun, ideal_fun):
        ”””
        function finds matches between training functions and ideal functions based on min(MSE)
        :param train_fun: define training functions
        :param ideal_fun: define ideal functions set
        :return: ideal functions dataframe and their deviations
        ”””
        # find last parameters of both fucntions

         if isinstance(train_fun, pd.DataFrame) and isinstance(ideal_fun, pd.DataFrame):
```

```
        ideal_lcol = len(ideal_fun.columns)
        train_lrow = train_fun.index[-1] + 1
        train_col = len(train_fun.columns)
        # Loop and find perfect four functions
        index_list = [] # here 4 ideal indexes will be strored
        least_square = [] # here 4 ideal MSEs will be stored
        for j in range(1, train_col): # loop through 4 train functions
            least_square1 = []
            for k in range(1, ideal_lcol): # loop through 50 ideal functions
                MSE_sum = 0 # Sum MSE
                for i in range(train_lrow): # calculate MSE Y value of train and Y value of ideal function
                    z1 = train_fun.iloc[i, j] # Train y value
                    z2 = ideal_fun.iloc[i, k] # Ideal y value
                    MSE_sum = MSE_sum + ((z1 - z2) ** 2)
                least_square1.append(MSE_sum / train_lrow)
            min_least = min(least_square1)
            index = least_square1.index(min_least) # find index of the ideal function
            index_list.append(index + 1)
            least_square.append(min_least)
        per_frame = pd.DataFrame(list(zip(index_list, least_square)), columns=["Index", "least_square_value"])
        return per_frame
    else:
        raise TypeError("Given arguments are not of Dataframe type.")
# Choose ideal functions based on train data
print("Choose ideal functions based on train data.")
df = FindFunctions().find_ideal_matches(train, ideal)
print(df)
```

Index	Selected Ideal Function	Least Square Value
0	Y42	0.085616
1	Y41	0.089005
2	Y11	0.074655
3	Y48	0.079909

Table 6: Select Ideal Functions Based on Train Data.

2.2 Plot a graph of all four pairs of Ideal and Train functions.

Python-program:

```
import sys # Standard library imports
import pandas as pd # related third party imports
import numpy as np
from matplotlib import pyplot as plt
from sqlalchemy import create_engine
```

```
class FindFunctions:
    def __init__(self):
        pass
    def prepare_graphs(self, x_fun, x_par, y1_fun, y1_par, y2_fun, y2_par, show_plots=True):
        """
        function prepares a plot based on given paramaters
        :param x_fun: x function
        :param x_par: x position
        :param y1_fun: y1 function
        :param y1_par: y1 position
        :param y2_fun: y2 function
        :param y2_par: y2 position
        :param show_plots: True/False to display plot
        :return: graph of x and y
        """
        x = x_fun.iloc[:, x_par] # x
        y1 = y1_fun.iloc[:, y1_par] # y1 (training function)
        y2 = y2_fun.iloc[:, y2_par] # y2 (ideal function)
        # print(y1, y2)
        plt.plot(x, y1, c="r", label="Train function") # plot both axis
        plt.plot(x, y2, c="b", label="Ideal function")
        plt.xlabel("x")
        plt.ylabel("y")
        plt.title("Four Best Fit of the Selected Ideal Functions Based on Train Functions.")
        plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
        if show_plots is True:
            plt.show() # show current plot
            plt.clf() # clear plots
        elif show_plots is False:
            pass
        else:
            pass # no paramater show_plots or wrong paramater show_plots was given
```

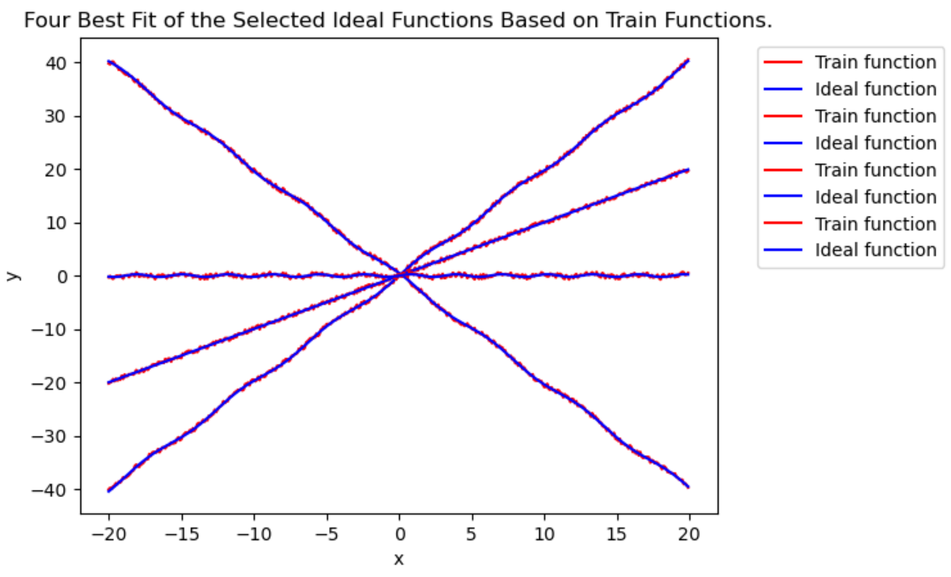


Figure 52: Plot of Four Best Fit Selected Ideal Functions Based on Train Functions.

2.3 i) Determine pair of values assigned to the selected four ideal functions. Afterwards execute the mapping and save it together with the deviation.

Python-program:

```
import sys # Standard library imports
import pandas as pd # related third party imports
import numpy as np
from matplotlib import pyplot as plt
from sqlalchemy import create_engine

class FindFunctions:
    def __init__(self):
        pass
    def find_ideal_via_row(self, test_fun):
        """
        determine for each and every x-y-pair of values whether they can be assigned to the four chosen ideal
functions
        :param test_fun: Dataframe with x and y values
        :return: test function paired with values from the four ideal functions
        """
        if isinstance(test_fun, pd.DataFrame):
            test_lrow = test_fun.index[-1] + 1 # last row of the test df (used for loop)
            test_lcol = len(test_fun.columns) # last columns of the test df (used for loop)
            # print(test)
            ideal_index = [] # list to store index of ideal function
            deviation = [] # list to store Deviation
            for j in range(test_lrow): # loop through rows
```

```
                MSE_l = [] # list to store all four deviations
                for i in range(2, test_lcol): # loop through columns 2, 3, 4, 5
                    z1 = test_fun.iloc[j, 1]
                    z2 = test_fun.iloc[j, i]
                    MSE_sum = ((z2 - z1) ** 2) # calculate MSE
                    MSE_l.append(MSE_sum) # append MSE to the MSE_l list
                min_least = min(MSE_l) # select min deviation in MSE_l
                if min_least < (np.sqrt(2)):
                    deviation.append(min_least) # append min_least to the deviation list
                    index = MSE_l.index(min_least) # select index of the min_least to find ideal function
                    ideal_index.append(index) # append index to the ideal_index list
                else:
                    deviation.append(min_least)
                    ideal_index.append("Miss") # no criteria match
            # Add two new columns to the test
            test["Deviation"] = deviation
            test["Ideal index"] = ideal_index
            return test
        else:
            raise TypeError("Given argument is not of Dataframe type.")
# read CSV files and load them into Dataframes
train = pd.read_csv("train.csv")
ideal = pd.read_csv("ideal.csv")
test = pd.read_csv("test.csv")
# Clean test df
test = test.sort_values(by=["x"], ascending=True) # sort by x
test = test.reset_index() # reset index
test = test.drop(columns=["index"]) # drop old index column
print("")
print("Clean test df")
# Get x, y values of each of the 4 ideal functions
print("")
print("Get x, y values of each of the 4 ideal functions.")
ideals = []
for i in range(0, 4):
    ideals.append(ideal[["x", f"ystr(df.iloc[i, 0])"]])
# merge test and 4 ideal functions
print("")
print("Merge test and 4 ideal functions.")
for i in ideals:
    test = test.merge(i, on="x", how="left")
# determine whether or not each and every x-y-pair of values can be assigned to the four chosen ideal functions.
print("")
print(”Determine for each and every x-y-pair of values whether or not they can be assigned to the four chosen
ideal functions.”)
test = FindFunctions().find_ideal_via_row(test)
# Replace values with ideal function names
print(””)
print(”Replace values with ideal function names.”)
for i in range(0, 4):
    test[”Ideal index”] = test[”Ideal index”].replace([i], str(f”ydf.iloc[i, 0]”))
# add y values to another test_fun (used later for scatter plot)
print(””)
print(”Add y values to another test_fun (used later for scatter plot).”)
test_scat = test
test_scat[”ideal y value”] = ””
for i in range(0, 100):
    k = test_scat.iloc[i, 7]
    if k == ”y42”:
        test_scat.iloc[i, 8] = test_scat.iloc[i, 2]
    elif k == ”y41”:
        test_scat.iloc[i, 8] = test_scat.iloc[i, 3]
    elif k == ”y11”:
        test_scat.iloc[i, 8] = test_scat.iloc[i, 4]
    elif k == ”y48”:
        test_scat.iloc[i, 8] = test_scat.iloc[i, 5]
    elif k == ”Miss”:
        test_scat.iloc[i, 8] = test_scat.iloc[i, 1]
print(test_scat)
# Drop other columns that are not used
print(””)
print(”Drop other columns that are not used.”)
test = test.drop(columns=[”y42”, ”y41”, ”y11”, ”y48”, ”ideal y value”])
print(test)
```

Determine whether x-y-pair of values can be assigned to the four chosen ideal functions.

x	y	y42	y41	y11	y48	Deviation	Ideal index	Ideal y value
-20.0	-19.284970	40.204040	-40.456474	-20.0	-0.186278	0.511268	y11	-20.0
-19.8	-19.915014	39.890660	-40.006836	-19.8	-0.236503	0.013228	y11	-19.8
-19.3	-38.458572	39.050125	-38.817684	-19.3	-0.195970	0.128961	y41	-38.817684
-19.2	-37.170870	38.869610	-38.571660	-19.2	-0.161224	1.962213	Miss	-37.17087
-19.1	-38.155376	38.684402	-38.323917	-19.1	-0.120051	0.028406	y41	-38.323917
...	...	...	...	...	...	...	...	...
18.2	18.535152	-36.001823	36.097584	18.2	-0.240830	0.112327	y11	18.2
18.7	0.832272	-36.905582	37.325500	18.7	-0.073668	0.820727	y48	-0.073668
18.8	37.523400	-37.100613	37.575233	18.8	-0.024737	0.002687	y41	37.575233
18.9	19.193245	-37.300636	37.825210	18.9	0.025179	0.085993	y11	18.9
19.7	38.955273	-39.070175	39.775787	19.7	0.247887	0.673243	y41	39.775787

Table 7: X and Y pair of values which can be assigned to the selected four ideal functions.

The executed mapping and saved deviation.

x	y	Deviation	Ideal index
-20.0	-19.284970	0.511268	y11
-19.8	-19.915014	0.013228	y11
-19.3	-38.458572	0.128961	y41
-19.2	-37.170870	1.962213	Miss
-19.1	-38.155376	0.028406	y41
...	...	...	...
18.2	18.535152	0.112327	y11
18.7	0.832272	0.820727	y48
18.8	37.523400	0.002687	y41
18.9	19.193245	0.085993	y11
19.7	38.955273	0.673243	y41

Table 8: Test dataset with additional deviation and ideal index.

2.4 Criteria of Database Tables.

The python-program to execute the required structure of database tables continues from all of the above defined programs in this chapter. Please note that it renames columns of the defined and loaded dataframes. Consult appendices and materials chapter for a full python-program.

# rename columns for the train table
print("")
print("Rename columns for the train table.")
train = train.rename(columns="y1": "Y1 (training func)", "y2": "Y2 (training func)", "y3": "Y3 (training func)", "y4": "Y4 (training func)")
print(train)
# rename columns for the ideal table
print("")
print("Rename columns for the ideal table.")

```
for col in ideal.columns: # rename columns in ideal to fit criteria
    if len(col) > 1: # if column name is not x, therefore > 1
        ideal = ideal.rename(columns=col: f”col (ideal func)”)
print(ideal)
ideal_excel = ’Rename_Ideal_Columns.xlsx’
# saving to excel
ideal.to_excel(ideal_excel)
print(’DataFrame is written to Excel File successfully.’)
# rename columns for the test table
print(””)
print(”Rename columns for the test table.”)
test = test.rename(columns=”x”: ”X (test func)”, ”y”: ”Y (test func)”, ”Deviation”: ”Delta Y (test func)”, ”Ideal
index”: ”No. of ideal func”)
print(test)
# Load data to sqlite
#assignment_database
print(””)
print(”Load data to sqlite.”)
dbs = SqliteDb()
dataframes = [train, ideal, test]
table_names = [”train_table”, ”ideal_table”, ”test_table”]
dbs.db_and_table_creation(dataframes, ”assignment_database”, table_names)
```

X (test func)	Y (test func)	Delta Y (test func)	No. of ideal func
-20	-19.28497	0.511268	y11
-19.8	-19.915014	0.013228	y11
-19.3	-38.458572	0.128961	y41
-19.2	-37.17087	1.962213	Miss
-19.1	-38.155376	0.028406	y41
...	...	...	...
18.2	18.535152	0.112327	y11
18.7	0.832272	0.820727	y48
18.8	37.5234	0.002687	y41
18.9	19.193245	0.085993	y11
19.7	38.955273	0.673243	y41

Table 9: The database table of the test-data, with mapping and y-deviation.

X	Y1 (training func)	Y2 (training func)	Y3 (training func)	Y4 (training func)
-20	39.778572	-40.07859	-20.214268	-0.324914
-19.9	39.604813	-39.784	-20.07095	-0.05882
-19.8	40.09907	-40.018845	-19.906782	-0.45183
-19.7	40.1511	-39.518402	-19.389118	-0.612044
-19.6	39.795662	-39.360065	-19.81589	-0.306076
...	...	...	...	...
19.5	-38.254158	39.661987	19.536741	0.695158
19.6	-39.106945	39.06788	19.840752	0.638423
19.7	-38.926495	40.211475	19.516634	0.109105
19.8	-39.276672	40.03887	19.377943	0.189025
19.9	-39.724934	40.558865	19.630678	0.513824

Table 10: The training data’s database table.

X	Y1 (ideal func)	Y2 (ideal func)	...	Y11 (ideal func)	Y12 (ideal func)	...	Y41 (ideal func)	Y42 (ideal func)	...	Y48 (ideal func)	Y49 (ideal func)	Y50 (ideal func)
-20	-0.9129453	0.40808207	...	-20	-58	...	-40.456474	40.20404	...	-0.18627828	0.9129453	0.3968496
-19.9	-0.8676441	0.4971858	...	-19.9	-57.7	...	-40.23382	40.04859	...	-0.21569017	0.8676441	0.47695395
-19.8	-0.81367373	0.58132184	...	-19.8	-57.4	...	-40.006836	39.89066	...	-0.23650314	0.81367373	0.5491291
-19.7	-0.75157344	0.65964943	...	-19.7	-57.1	...	-39.775787	39.729824	...	-0.24788749	0.75157344	0.6128399
-19.6	-0.6819636	0.7313861	...	-19.6	-56.8	...	-39.54098	39.565693	...	-0.24938935	0.6819636	0.6679019
...	...	...	...	...	...	...	...	...	...	...	...	...
19.5	0.60553986	0.795815	...	19.5	60.5	...	39.30277	-38.602093	...	0.24094884	0.60553986	0.7144341
19.6	0.6819636	0.7313861	...	19.6	60.8	...	39.54098	-38.83431	...	0.24938935	0.6819636	0.6679019
19.7	0.75157344	0.65964943	...	19.7	61.1	...	39.775787	-39.070175	...	0.24788749	0.75157344	0.6128399
19.8	0.81367373	0.58132184	...	19.8	61.4	...	40.006836	-39.309338	...	0.23650314	0.81367373	0.5491291
19.9	0.8676441	0.4971858	...	19.9	61.7	...	40.23382	-39.551407	...	0.21569017	0.8676441	0.47695395

Table 11: The ideal functions’ database table.

2.5 ii) Logical visualization of all data.

Please consult appendices and materials chapter for a full program.

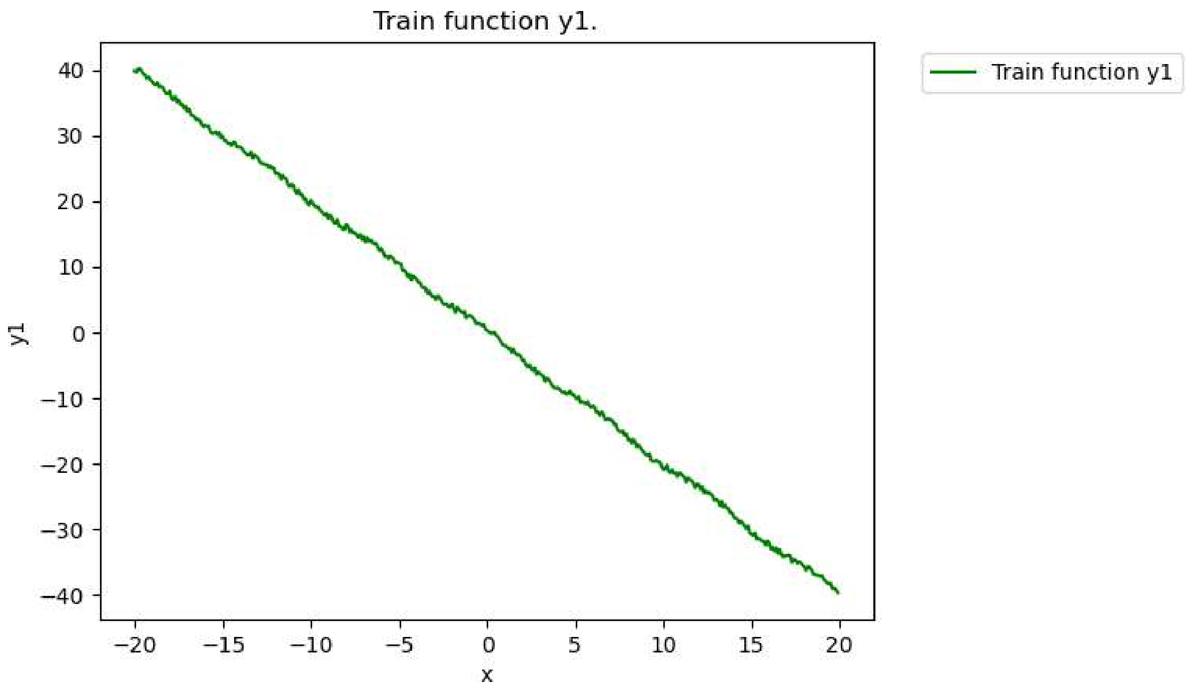
Python-program:

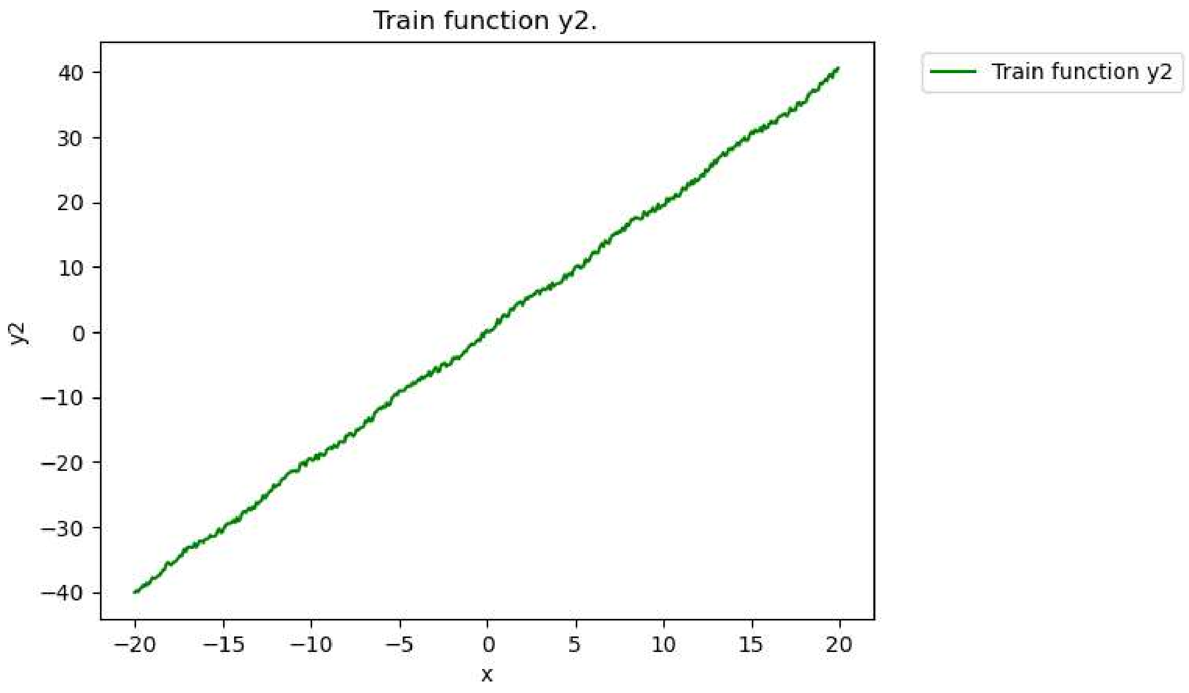
```
# Load data to sqlite
#assignment_database
print(””)
print(”Load data to sqlite.”)
dbs = SqliteDb()
dataframes = [train, ideal, test]
table_names = [”train_table”, ”ideal_table”, ”test_table”]
dbs.db_and_table_creation(dataframes, ”assignment_database”, table_names)
# Visualization
# train functions
print(””)
print(”Visualization.”)
print(”Train functions.”)
plt.clf()
x = train.iloc[:, 0]
for i in range(1, len(train.columns)):
    plt.plot(x, train.iloc[:, i], c=”g”, label=f”Train function y{i}”)
    plt.title(f"Train function y{i}.")
    plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
    plt.xlabel("x")
    plt.ylabel(f"y{i}")
    plt.show()
    plt.clf()
# ideal functions (4 chosen)
print("")
print("Ideal functions (4 chosen).")
plt.clf()
x = train.iloc[:, 0]
for i in range(0, df.index[-1] + 1):
    y = df.iloc[i, 0] # get ideal y column number (42, 41, 11, 48)
    plt.plot(x, ideal.iloc[:, y], c="#FF4500", label=f"Ideal function y{y}")
    plt.title(f"The Chosen Ideal Function y{y}.")
    plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
    plt.xlabel("x")
    plt.ylabel(f"y{y}")
    plt.show()
    plt.clf()
# test scatter (show points of test.csv)
print("")
print("Test scatter (show points of test.csv).")
plt.clf() # clear previous plots
plt.scatter(test.iloc[:, 0], test.iloc[:, 1]) # select x and y values
plt.title("Test scatter (show points of test.csv).")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
plt.clf() # clear previous plots
# create lists to visualize test_scat dataframe
print("")
print("Create lists to visualize test_scat dataframe.")
x1 = []
x2 = []
x3 = []
x4 = []
xm = []
y1 = []
y2 = []
y3 = []
y4 = []
ym = []
```

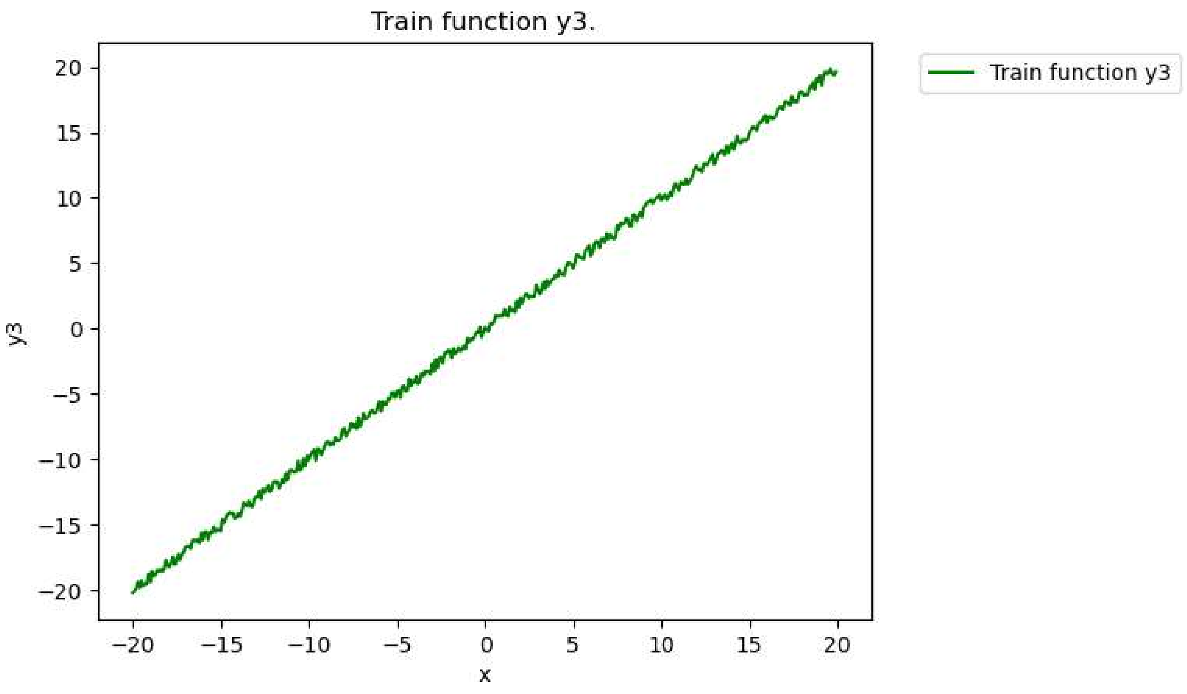
```
# append x and y values to the upper lists
print(””)
print(”Append x and y values to the upper lists.”)
for i in range(0, 100):
    k = test_scat.iloc[i, 7]
    if k == ”y42”:
        x1.append(test_scat.iloc[i, 0]) # append x value of y42 to the x1 list
        y1.append(test_scat.iloc[i, 8]) # append y value of y42 to the y1 list
    elif k == ”y41”:
        x2.append(test_scat.iloc[i, 0]) # append x value of y41 to the x2 list
        y2.append(test_scat.iloc[i, 8]) # append y value of y41 to the y2 list
    elif k == ”y11”:
        x3.append(test_scat.iloc[i, 0]) # append x value of y11 to the x3 list
        y3.append(test_scat.iloc[i, 8]) # append y value of y11 to the y3 list
    elif k == ”y48”:
        x4.append(test_scat.iloc[i, 0]) # append x value of y48 to the x4 list
        y4.append(test_scat.iloc[i, 8]) # append y value of y48 to the y4 list
    elif k == ”Miss”:
        xm.append(test_scat.iloc[i, 0]) # append x value of ”Miss” values to the xm list
        ym.append(test_scat.iloc[i, 8]) # append y value of ”Miss” values to the ym list
# plot ideal functions and test y-values on the same scatter plot
print(””)
print(”Plot ideal functions and test y-values on the same scatter plot.”)
plt.scatter(x1, y1, marker=”o”, label=”Test - y42”, color=”r”)
plt.scatter(x2, y2, marker=”s”, label=”Test - y41”, color=”b”)
plt.scatter(x3, y3, marker=”ˆ”, label=”Test - y11”, color=”g”)
plt.scatter(x4, y4, marker=”d”, label=”Test - y48”, color=”#FFD700”)
plt.scatter(xm, ym, marker=”x”, label=”Test - Miss”, color=”#000000”)
plt.plot(ideal.iloc[:, 0], ideal.iloc[:, 42], label=”Ideal - Y42”, color=”#FA8072”)
plt.plot(ideal.iloc[:, 0], ideal.iloc[:, 41], label=”Ideal - Y41”, color=”#1E90FF”)
plt.plot(ideal.iloc[:, 0], ideal.iloc[:, 11], label=”Ideal - Y11”, color=”#7CFC00”)
plt.plot(ideal.iloc[:, 0], ideal.iloc[:, 48], label=”Ideal - Y48”, color=”#FFA500”)
plt.title(”Scatter plot of ideal functions and test y-values.”)
plt.xlabel(”x”)
plt.ylabel(”y”)
plt.legend(bbox_to_anchor=(1.05, 1.0), loc=’upper left’)
plt.show()
# ideal functions (all 50)
print(””)
print(”Ideal functions (all 50).”)
plt.clf()
x = ideal.iloc[:, 0]
for i in range(1, len(ideal.columns)):
```

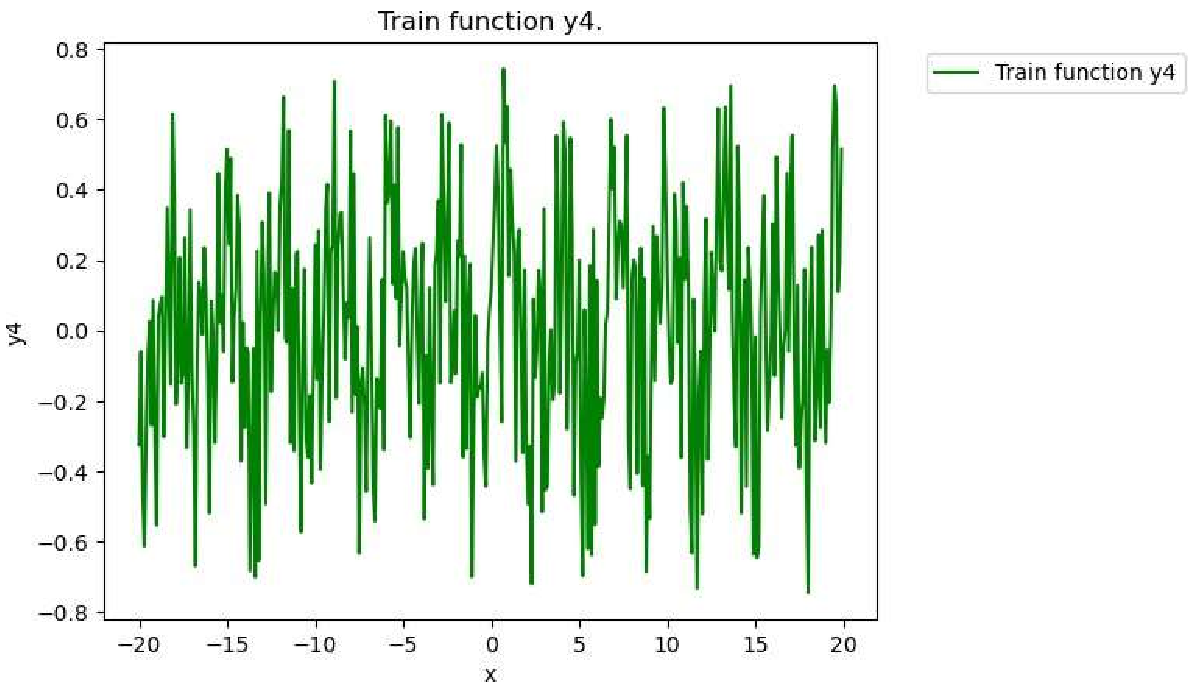
```
plt.plot(x, ideal.iloc[:, i], c=”#FF4500”, label=f”Ideal function y{i}”)
plt.title(f”Ideal Function y{i}.”)
plt.legend(bbox_to_anchor=(1.05, 1.0), loc=’upper left’)
plt.xlabel(”x”)
plt.ylabel(f”y{i}”)
plt.show()
plt.clf()
```

Visualization.
Train functions.

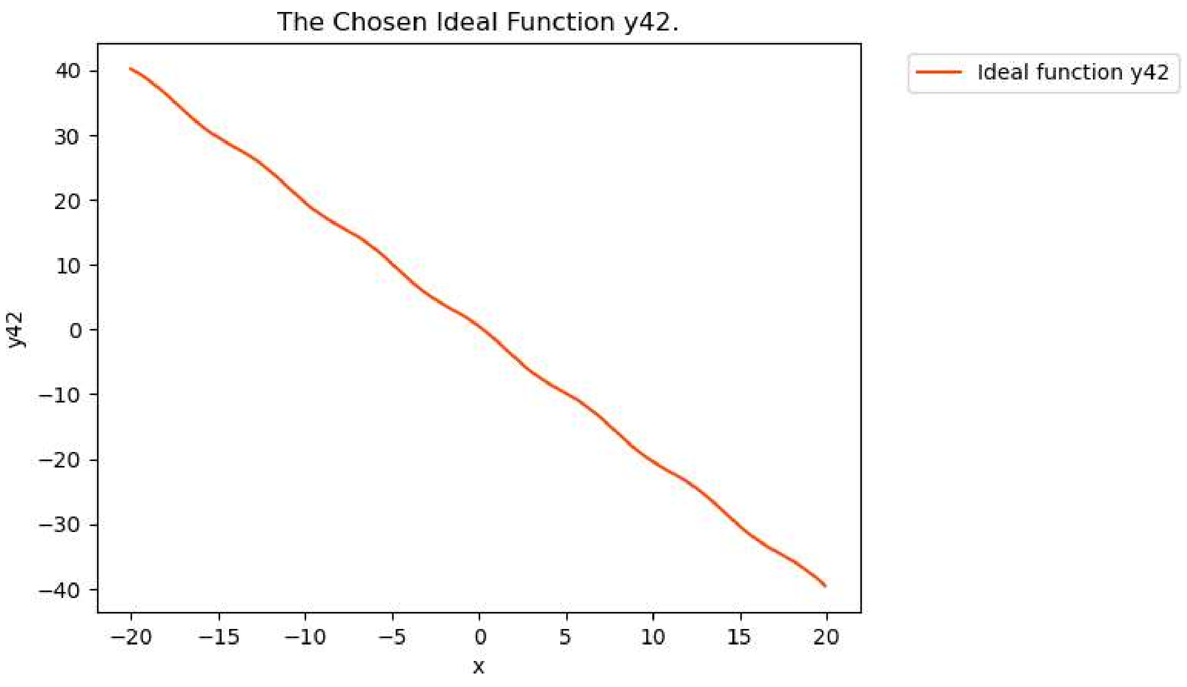


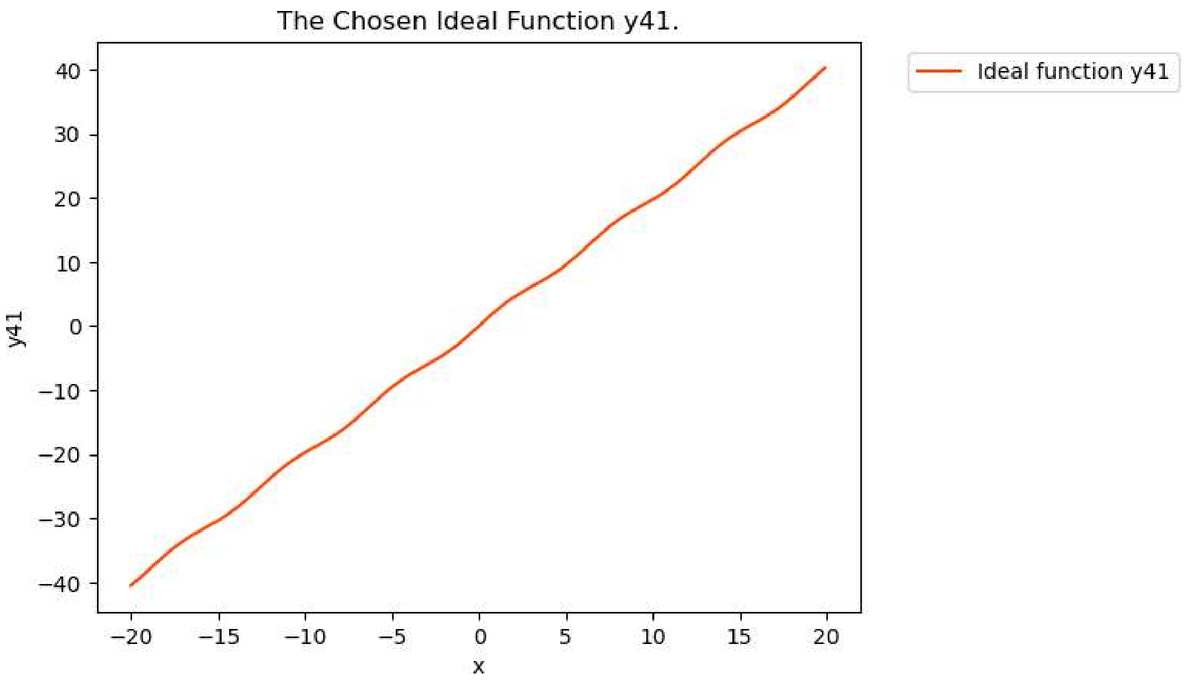


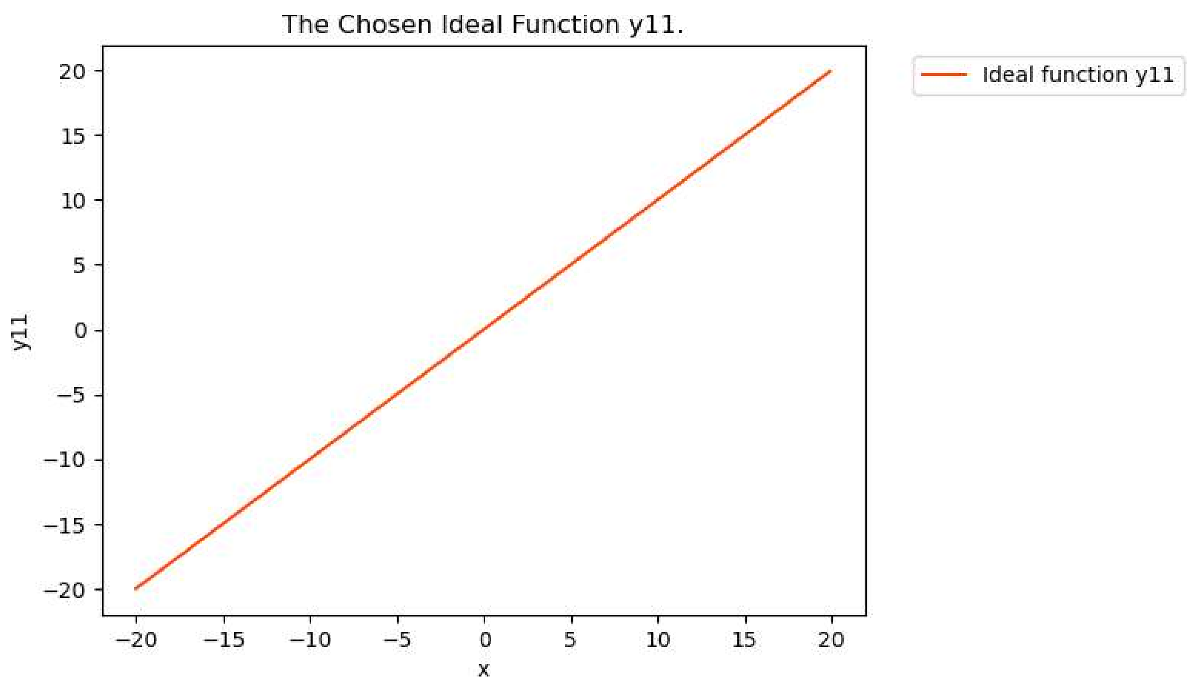

Train function y3.
Train function y3
y3
x
20
15
10
5
0
−5
−10
−15
−20

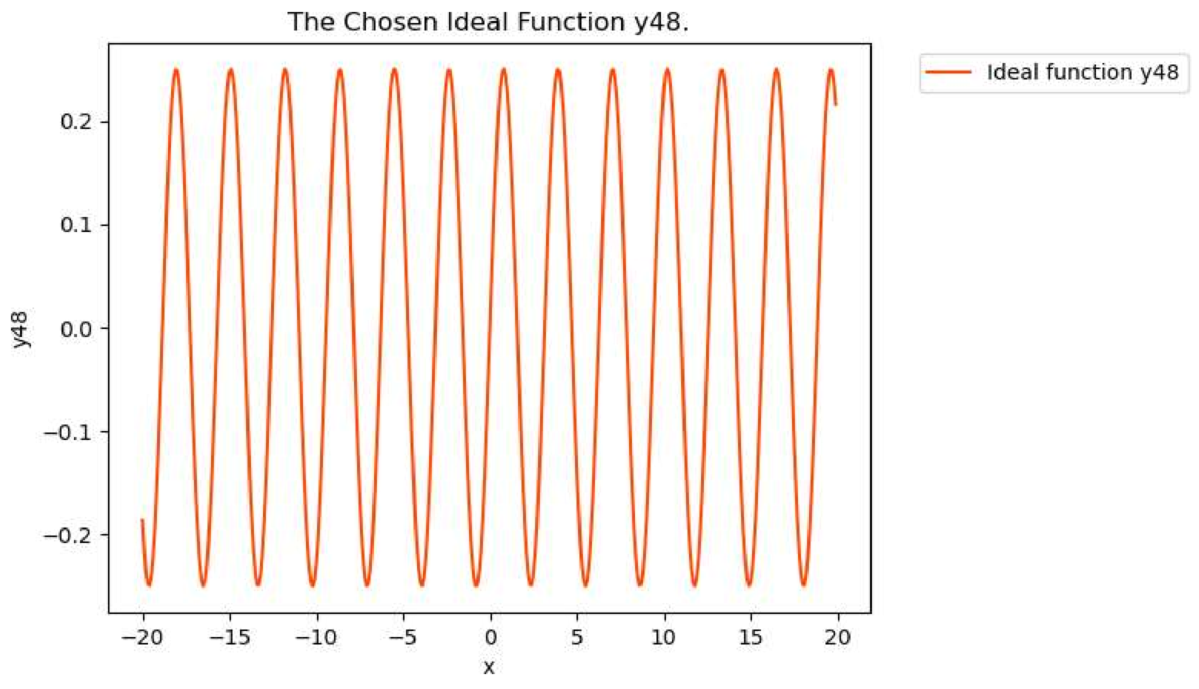

Train function y4.
Train function y4
y4
x
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−20
−15
−10
−5
0
5
10
15
20

Ideal functions (4 chosen).

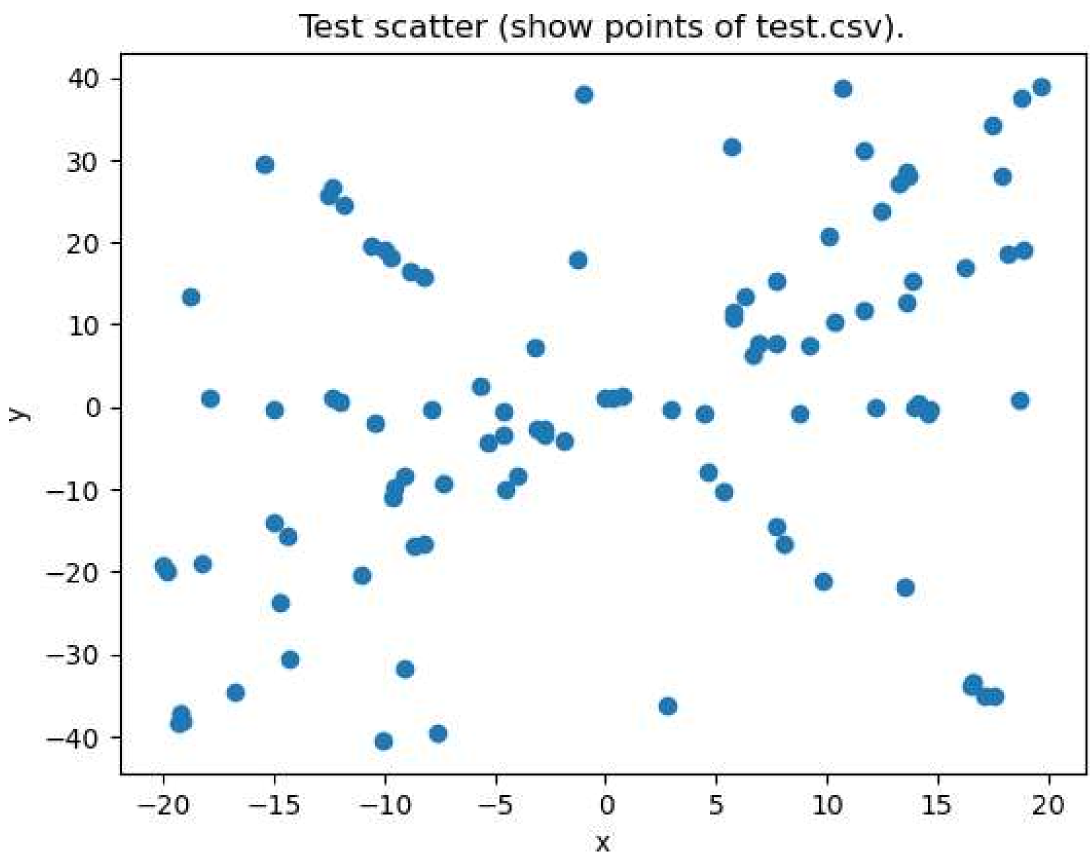




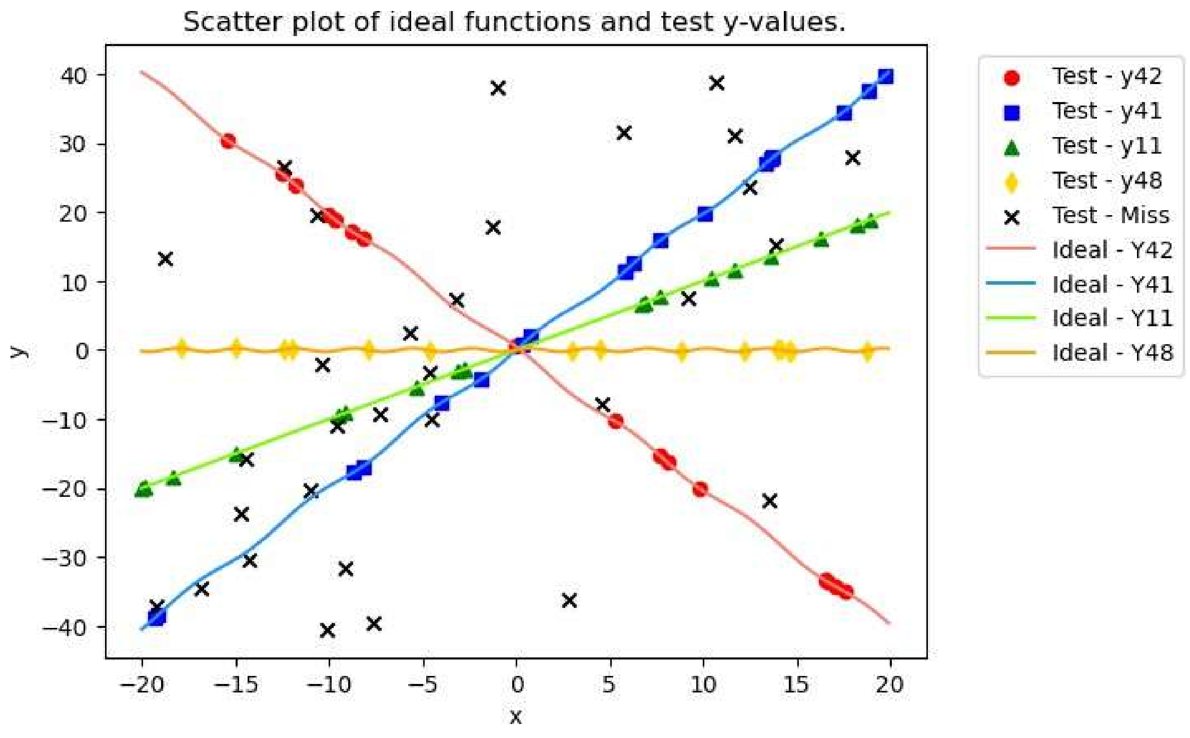




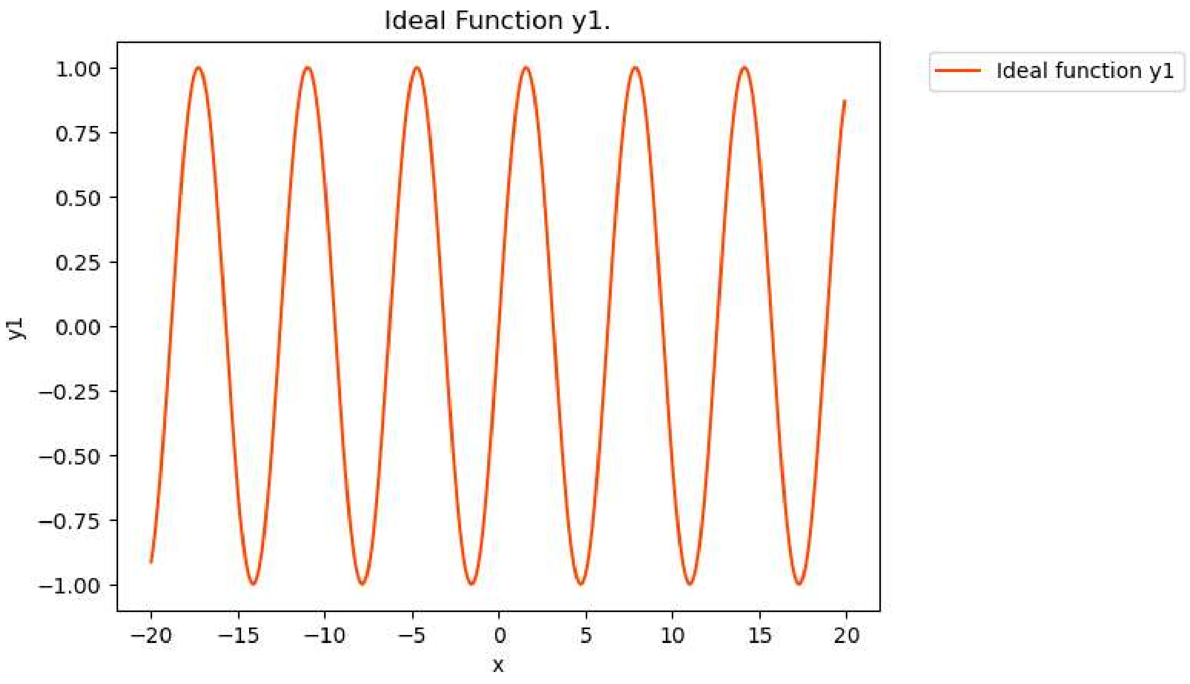
Scatter plot of test data including the points of test.csv.

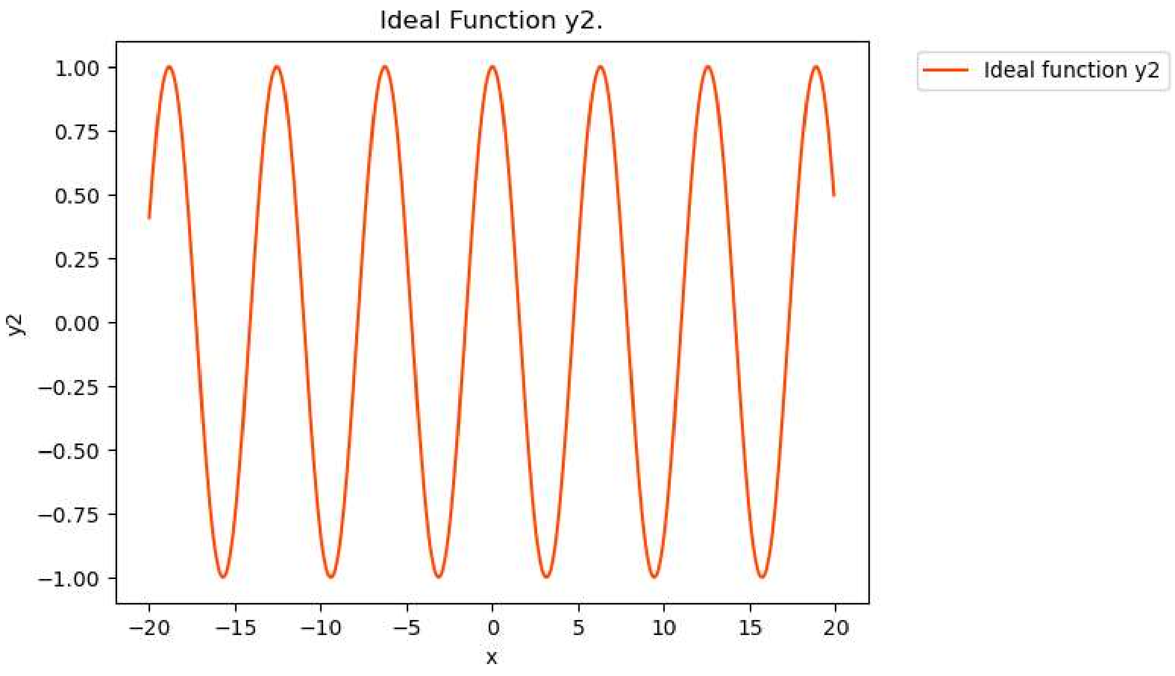


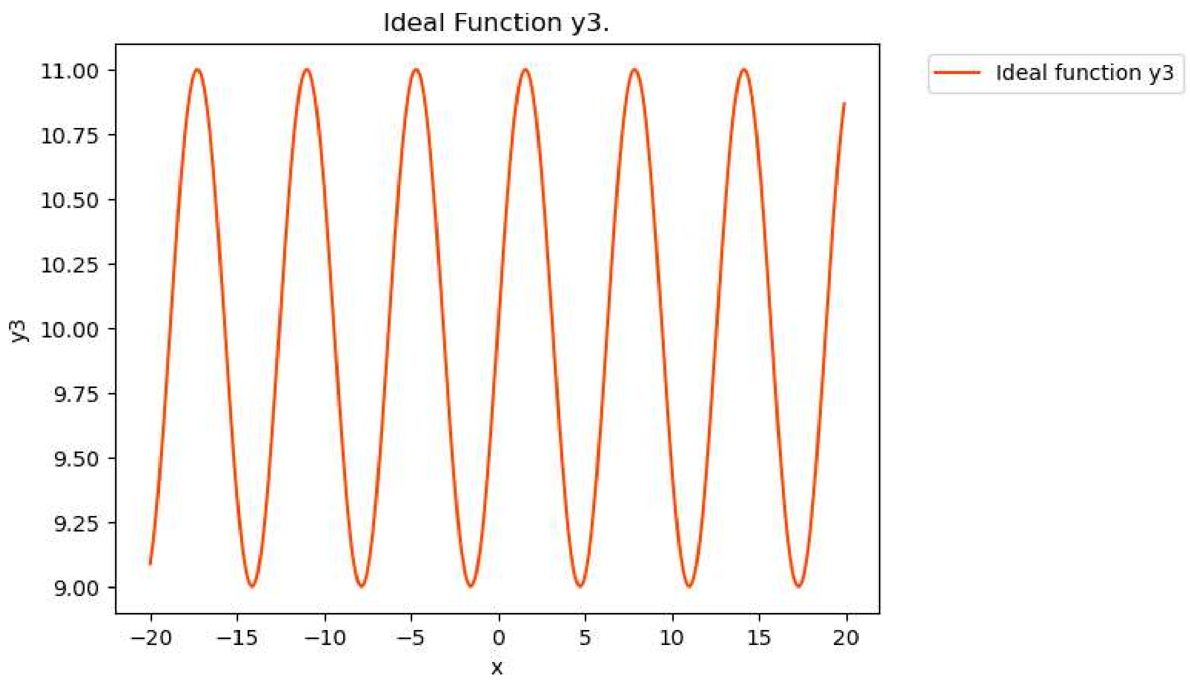
Plot ideal functions and test y-values on the same scatter plot.

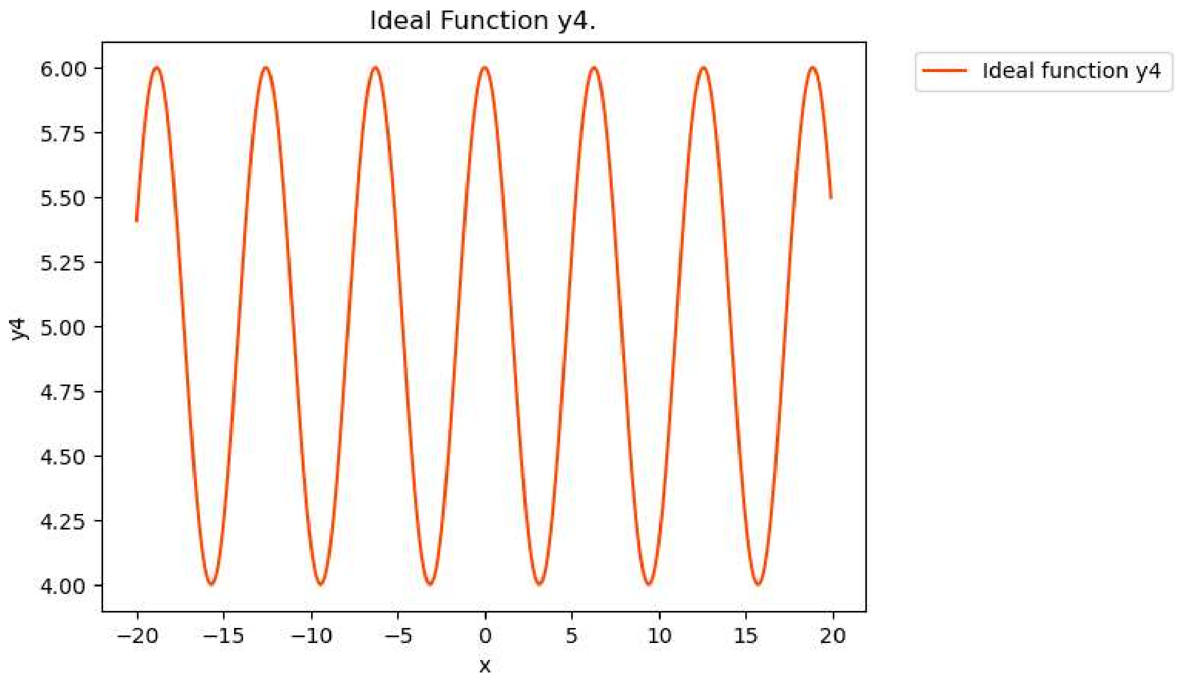


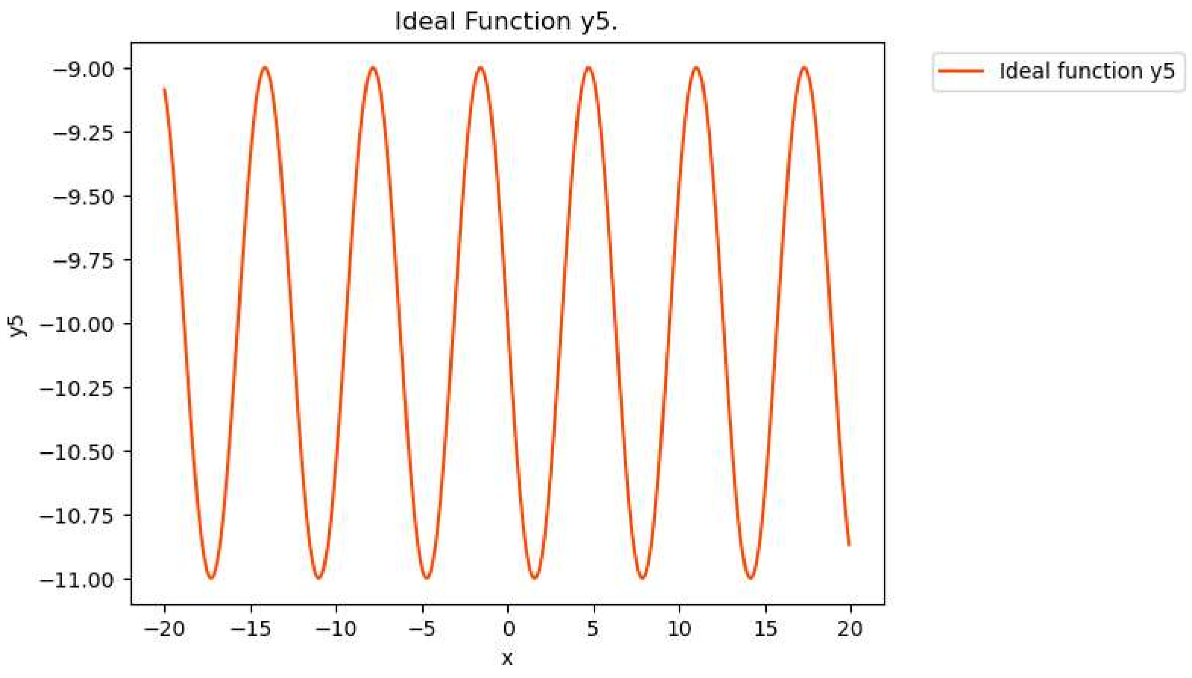
Ideal functions (all 50).

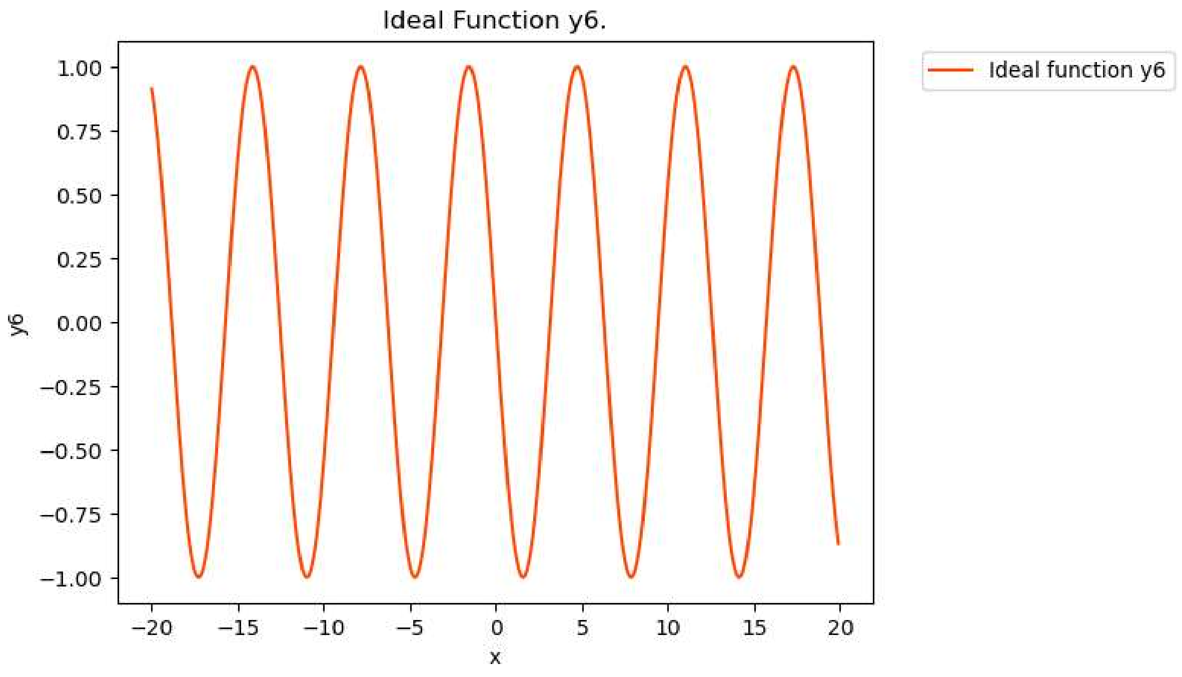


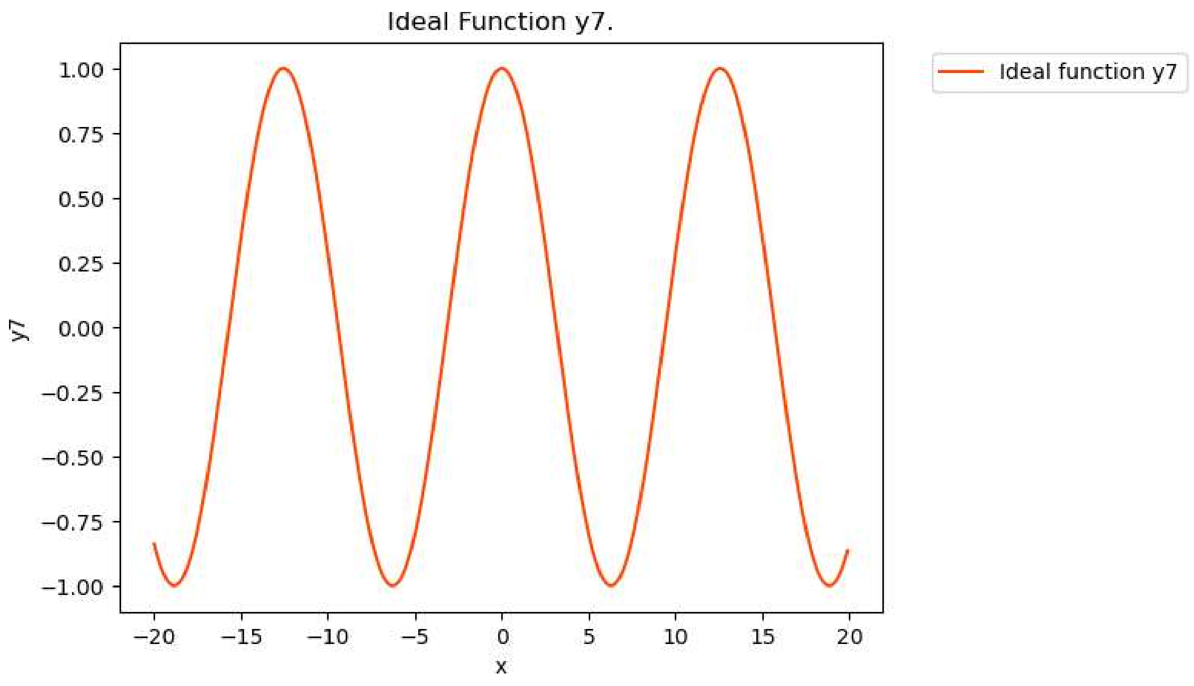


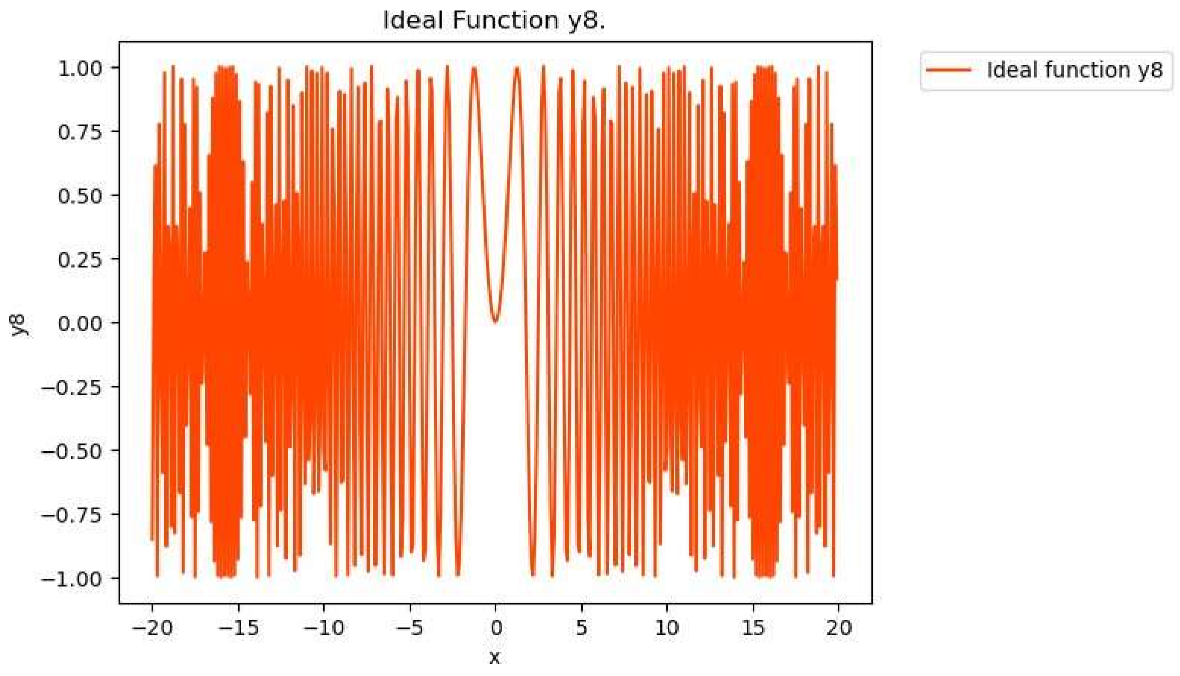

Ideal Function y3.
Ideal function y3
y3
x
11.00
10.75
10.50
10.25
10.00
9.75
9.50
9.25
9.00
−20
−15
−10
−5
0
5
10
15
20

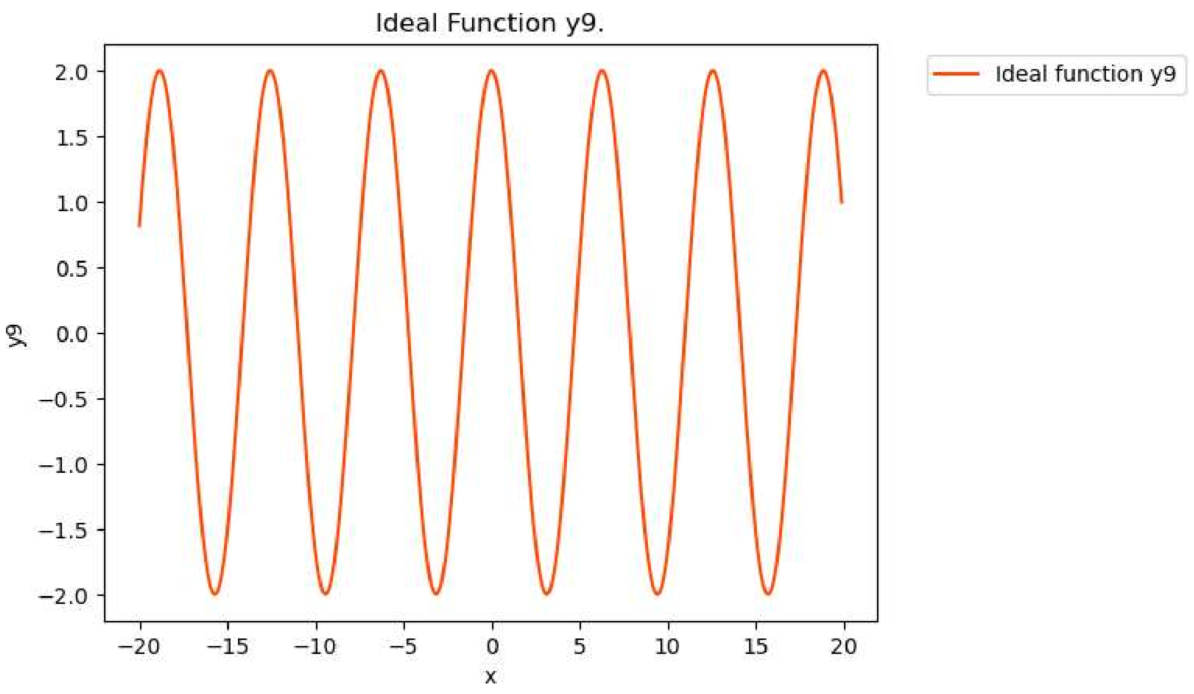

Ideal Function y4.
Ideal function y4
y4
x
6.00
5.75
5.50
5.25
5.00
4.75
4.50
4.25
4.00
−20
−15
−10
−5
0
5
10
15
20

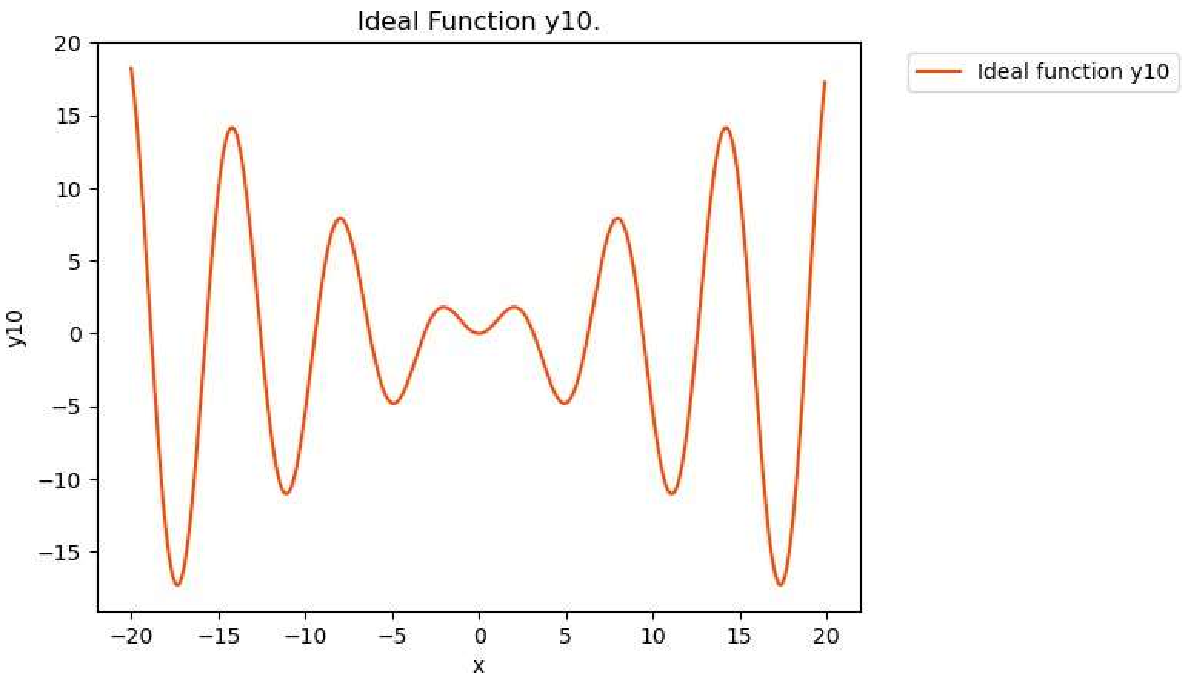

Ideal Function y5.
Ideal function y5
y5
x
−9.00
−9.25
−9.50
−9.75
−10.00
−10.25
−10.50
−10.75
−11.00
−20
−15
−10
−5
0
5
10
15
20

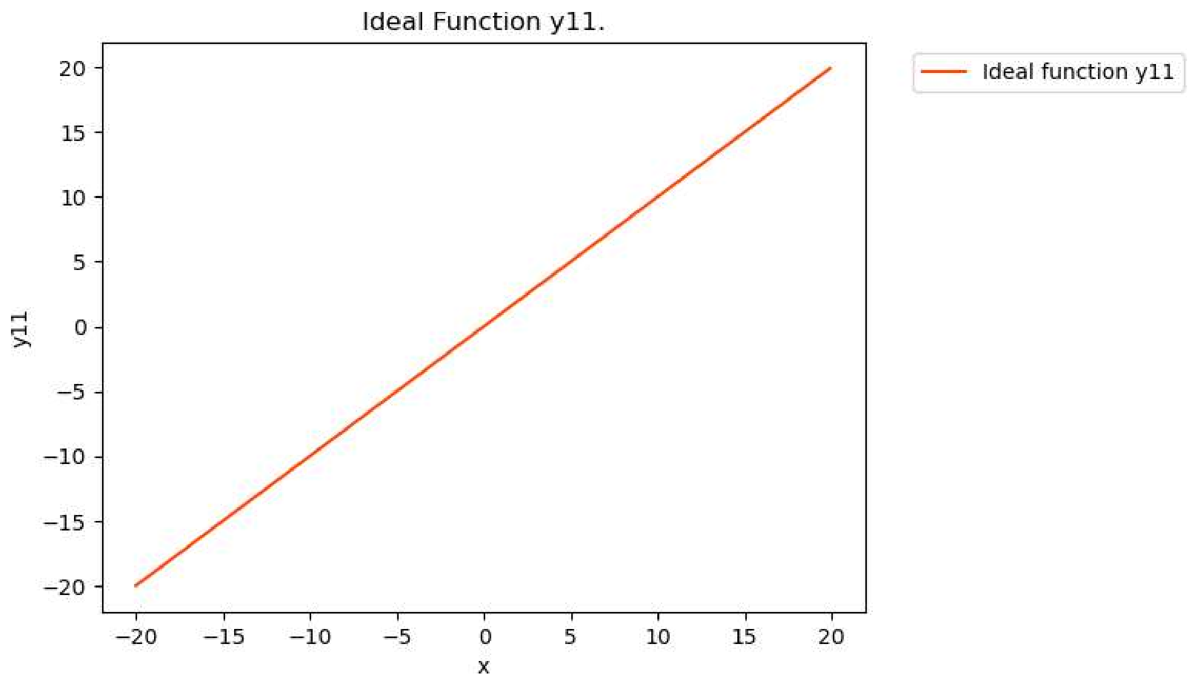

Ideal Function y6.
Ideal function y6
y6
x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−20
−15
−10
−5
0
5
10
15
20

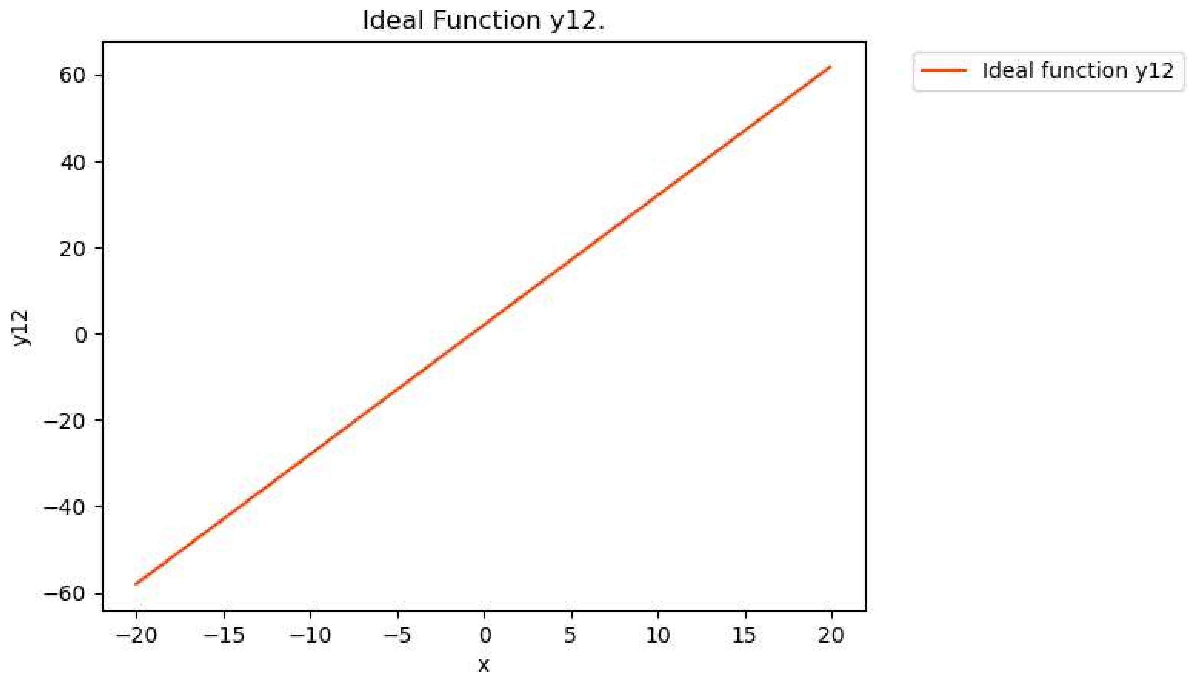

Ideal Function y7.
Ideal function y7
y7
x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−20
−15
−10
−5
0
5
10
15
20

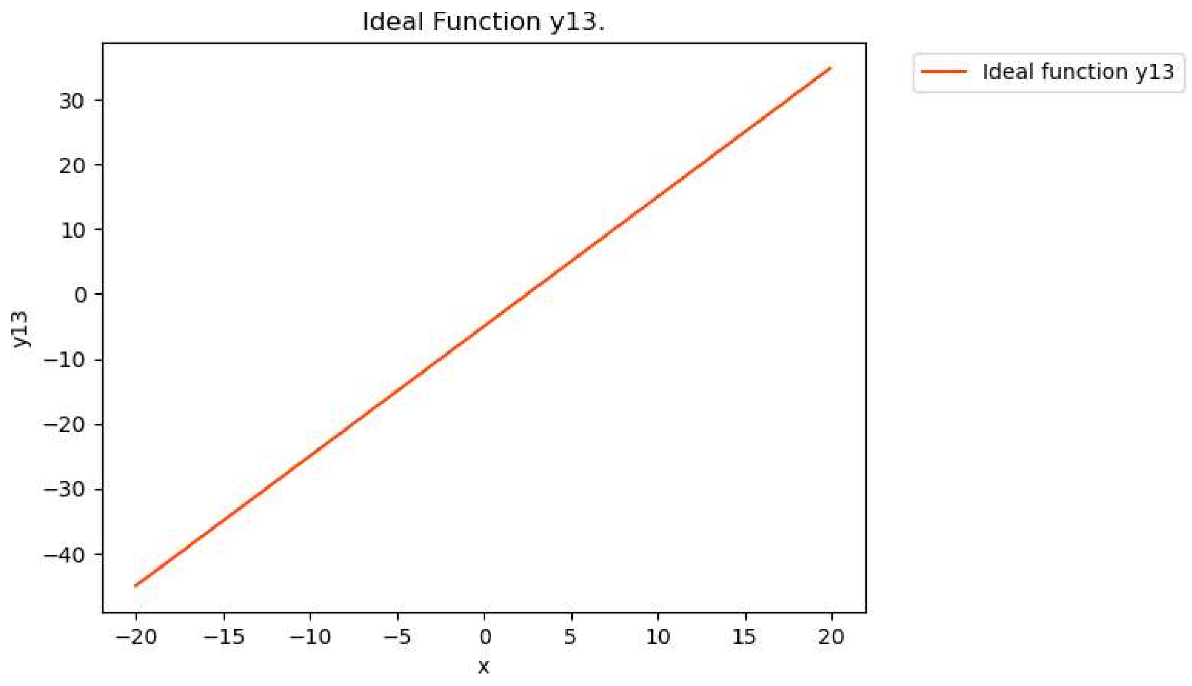

Ideal Function y8.
Ideal function y8
y8
x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−20
−15
−10
−5
0
5
10
15
20

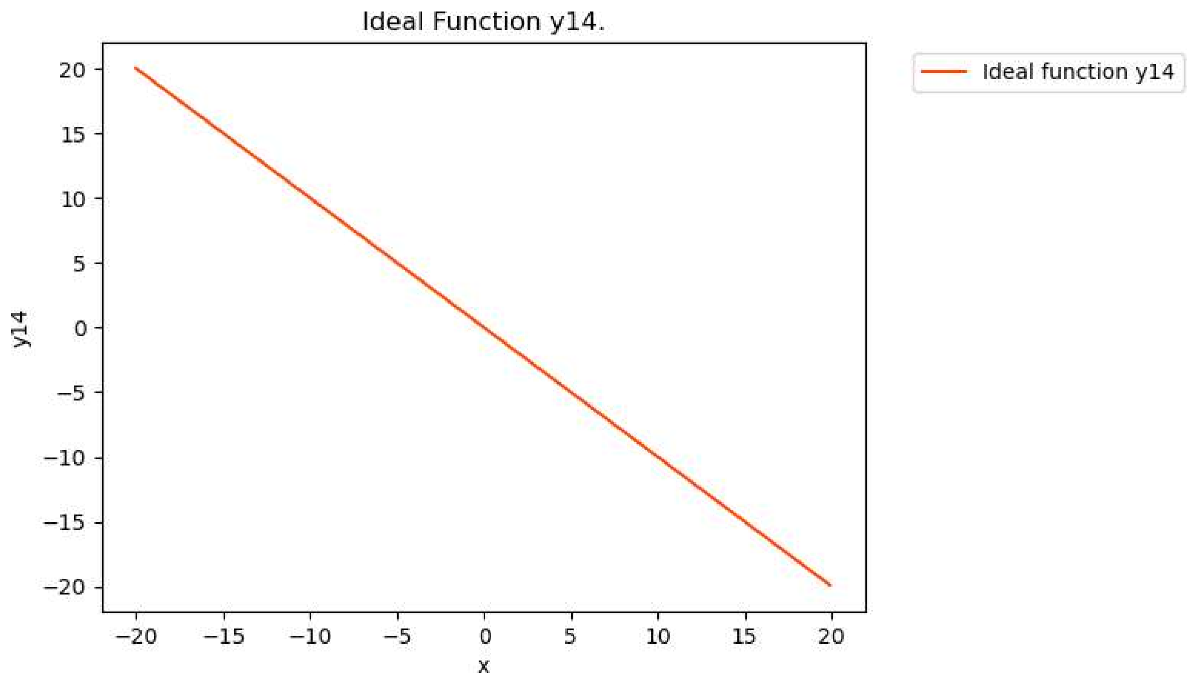

Ideal Function y9.
Ideal function y9
y9
x
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−20
−15
−10
−5
0
5
10
15
20

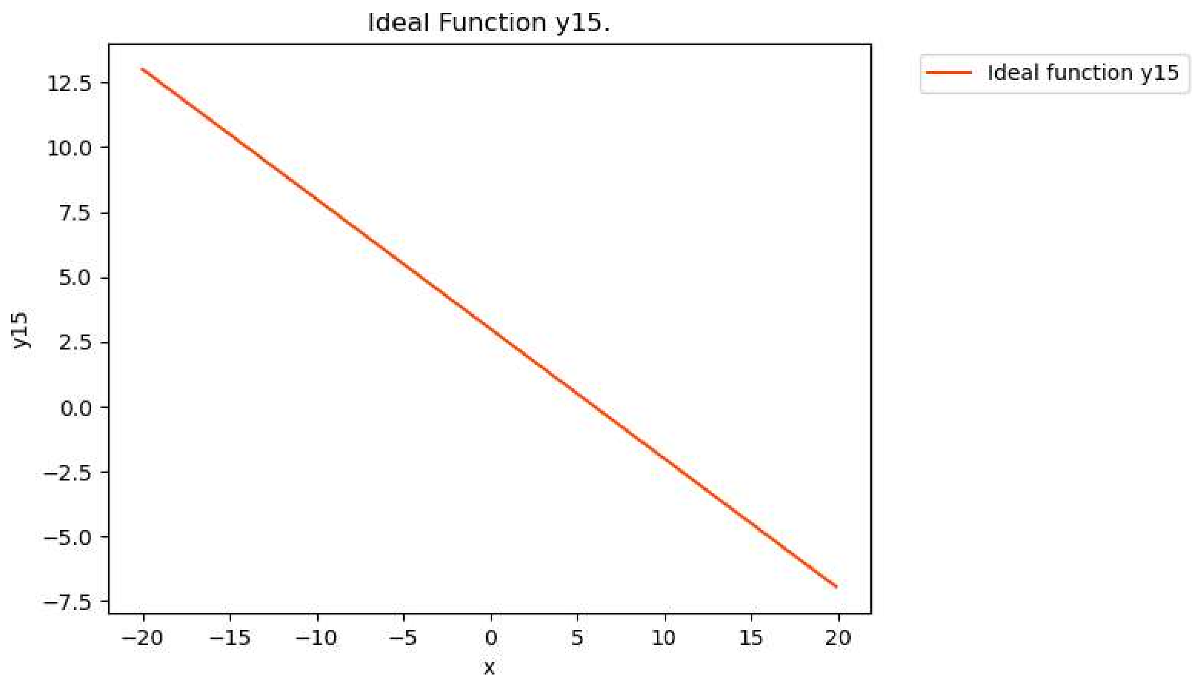

Ideal Function y10.
Ideal function y10
y10
x
20
15
10
5
0
−5
−10
−15
−20
−15
−10
−5
0
5
10
15
20

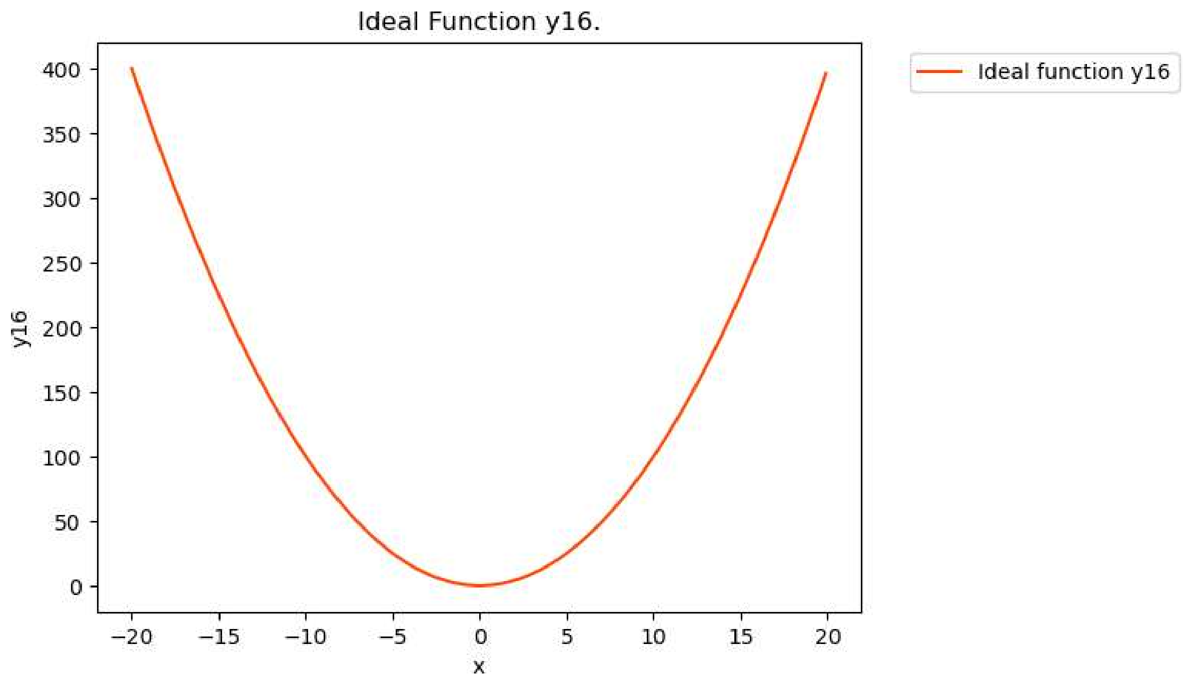

Ideal Function y11.
Ideal function y11
y11
x
20
15
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20

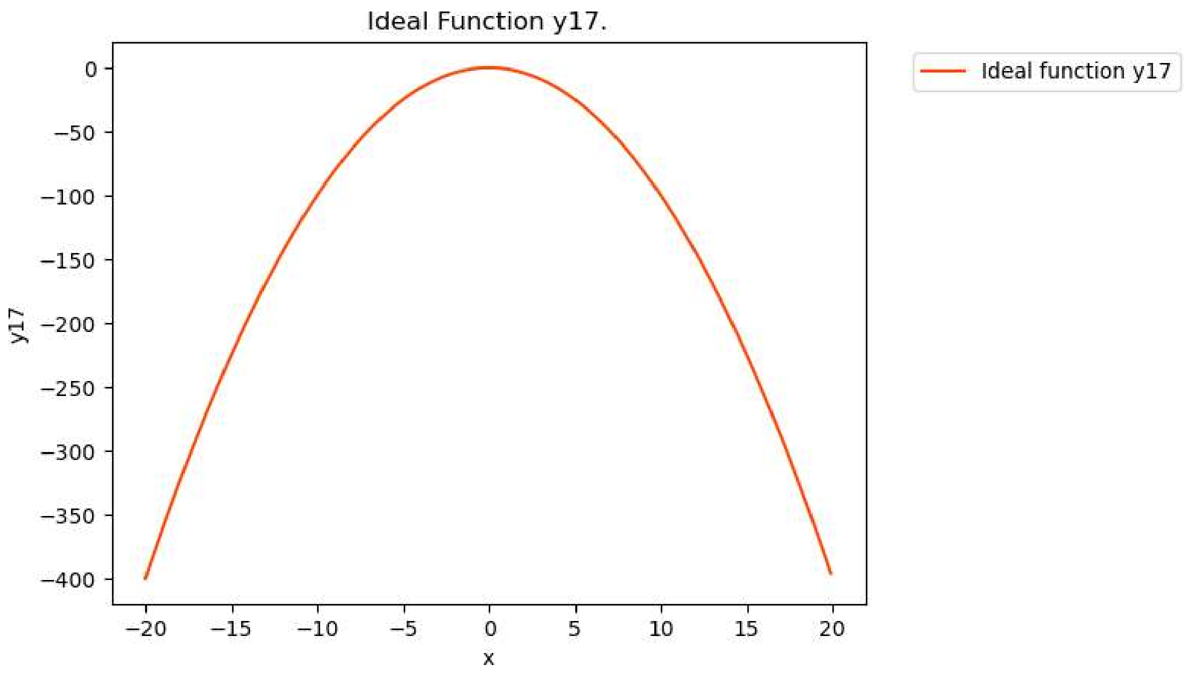

Ideal Function y12.
Ideal function y12
y12
x
60
40
20
0
−20
−40
−60
−20
−15
−10
−5
0
5
10
15
20

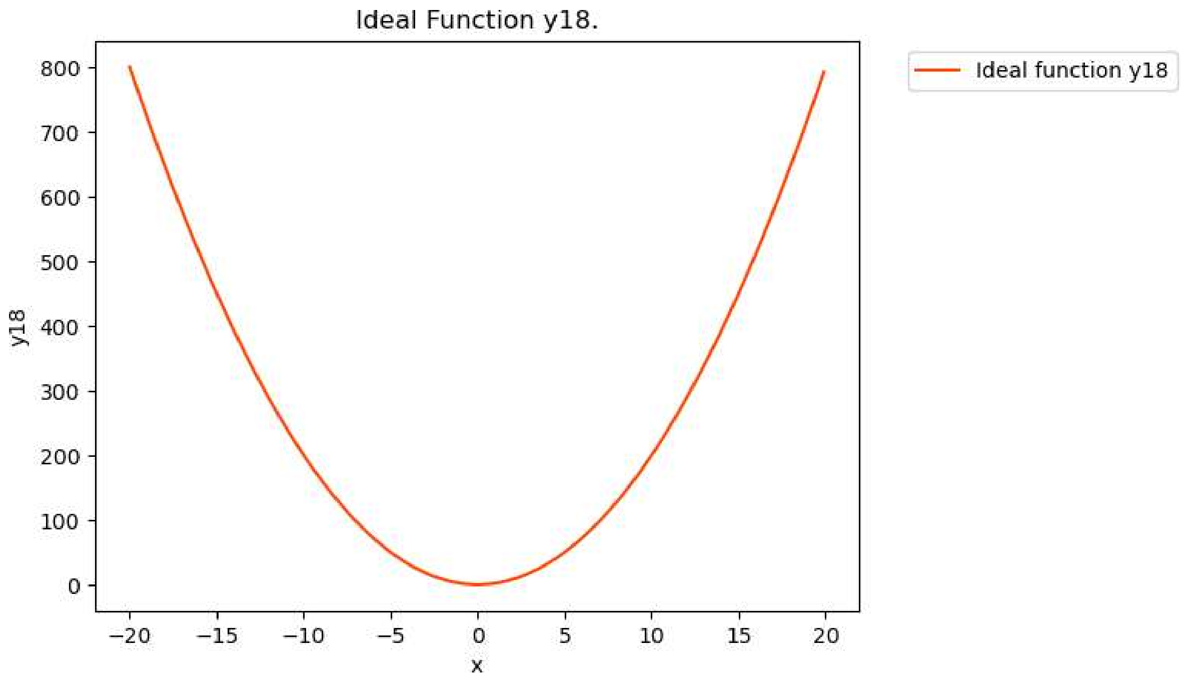

Ideal Function y13.
Ideal function y13
y13
x

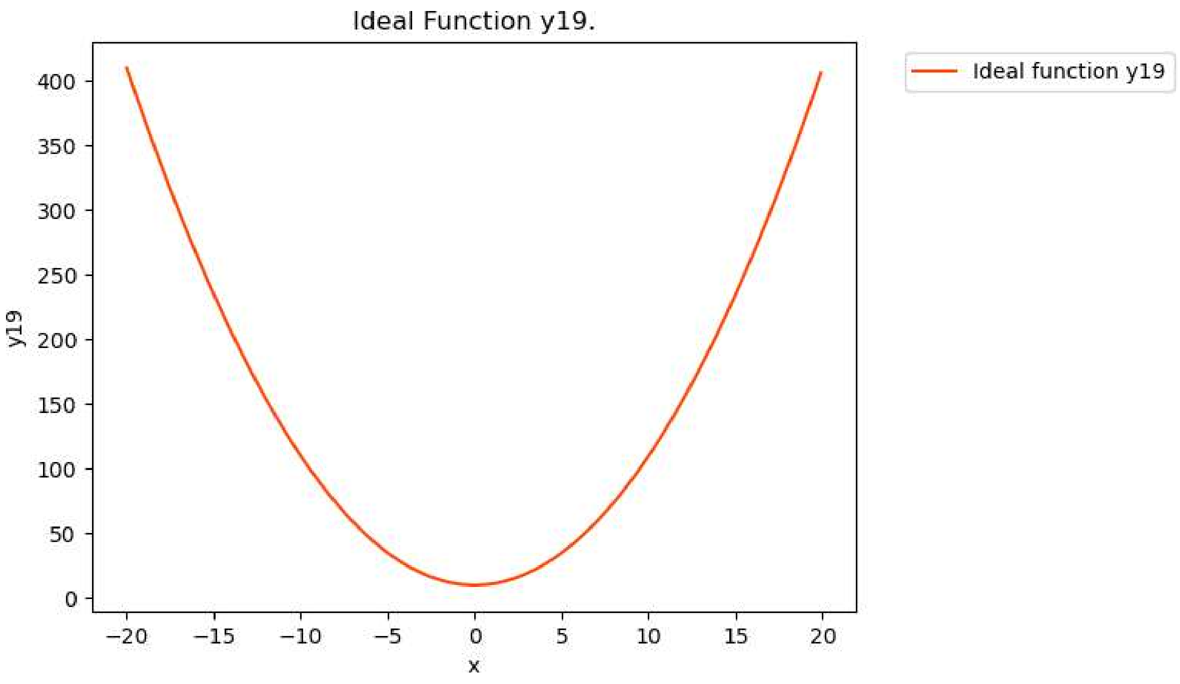

Ideal Function y14.
Ideal function y14
y14
x

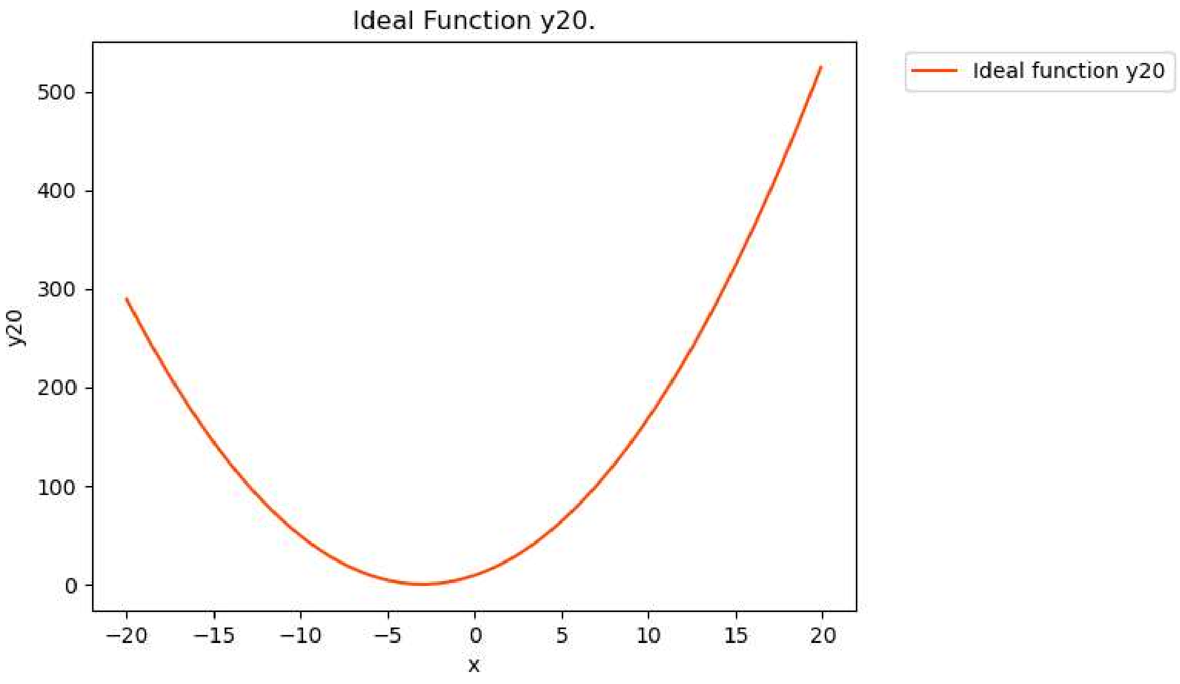

Ideal Function y15.
Ideal function y15
y15
x
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−20
−15
−10
−5
0
5
10
15
20

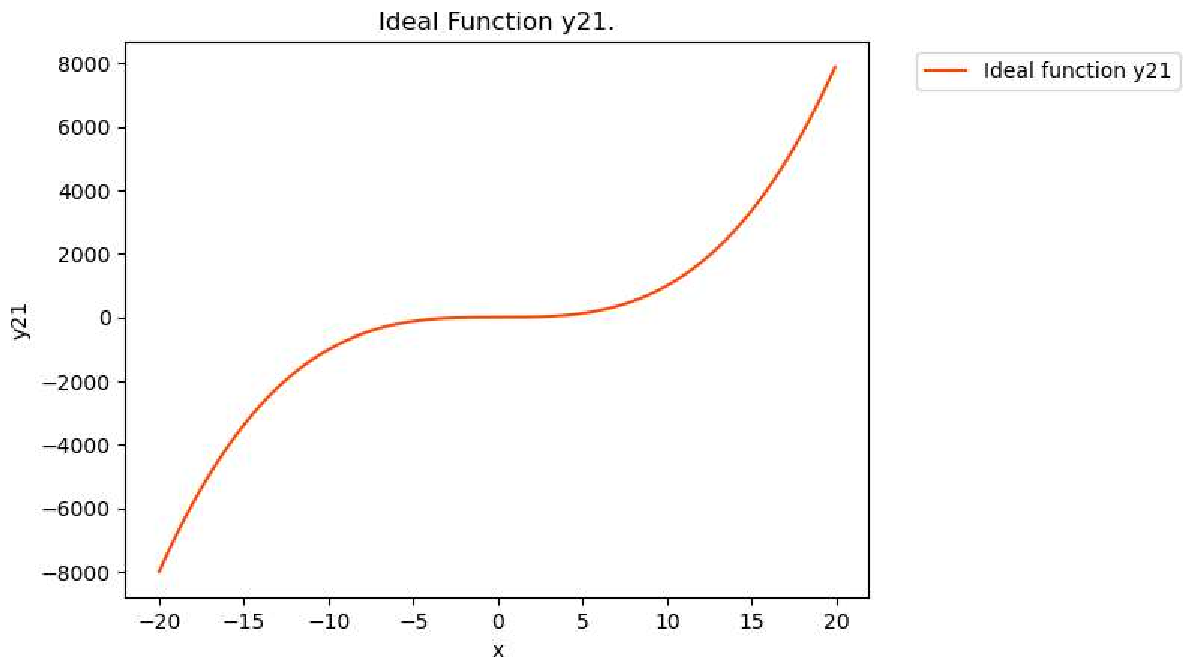

Ideal Function y16.
Ideal function y16
y16
x
400
350
300
250
200
150
100
50
0
−20
−15
−10
−5
0
5
10
15
20

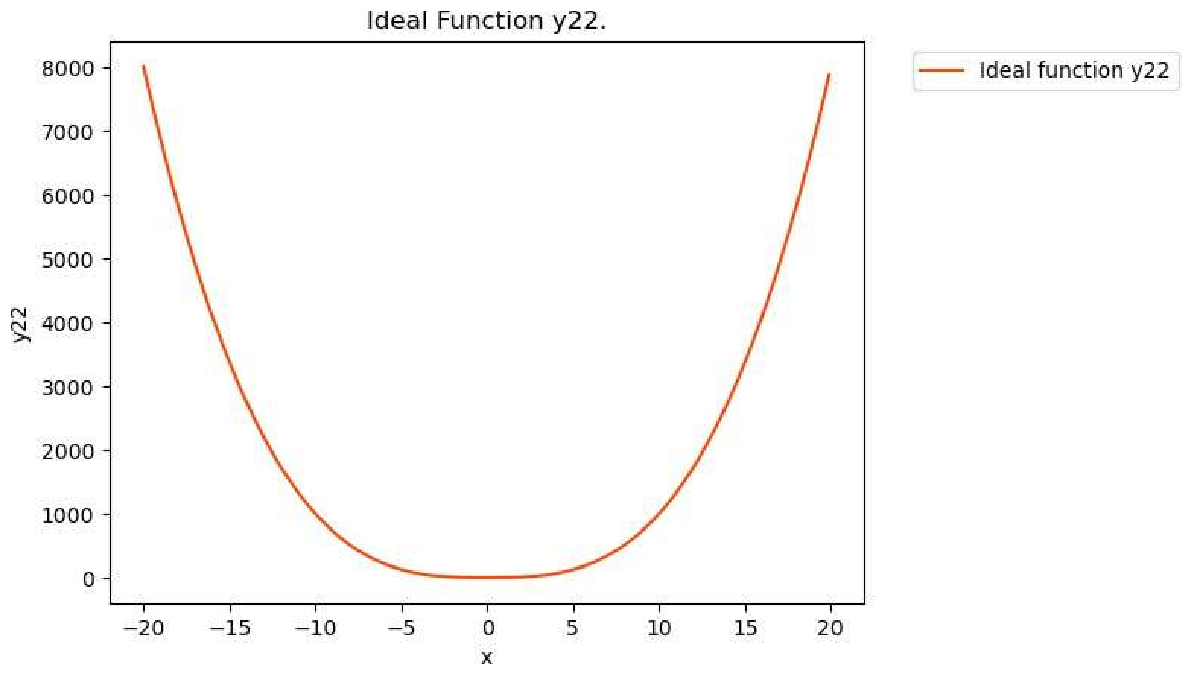

Ideal Function y17.
Ideal function y17
y17
x
0
−50
−100
−150
−200
−250
−300
−350
−400
−20
−15
−10
−5
0
5
10
15
20

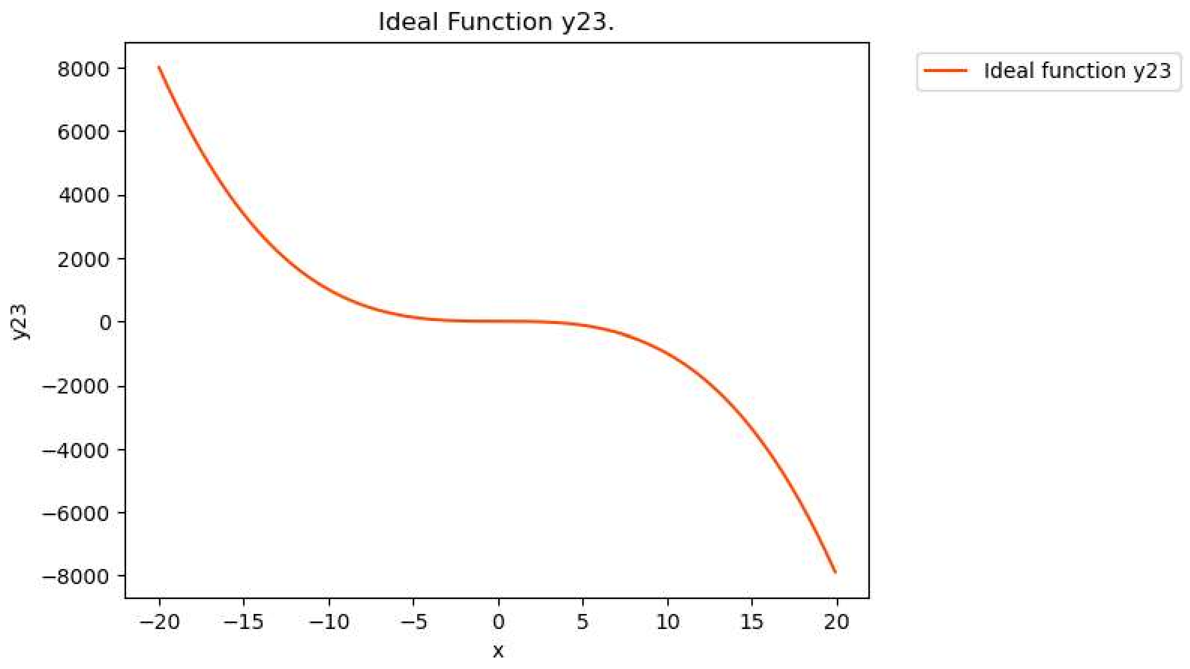

Ideal Function y18.
Ideal function y18
y18
x
800
700
600
500
400
300
200
100
0
−20
−15
−10
−5
0
5
10
15
20

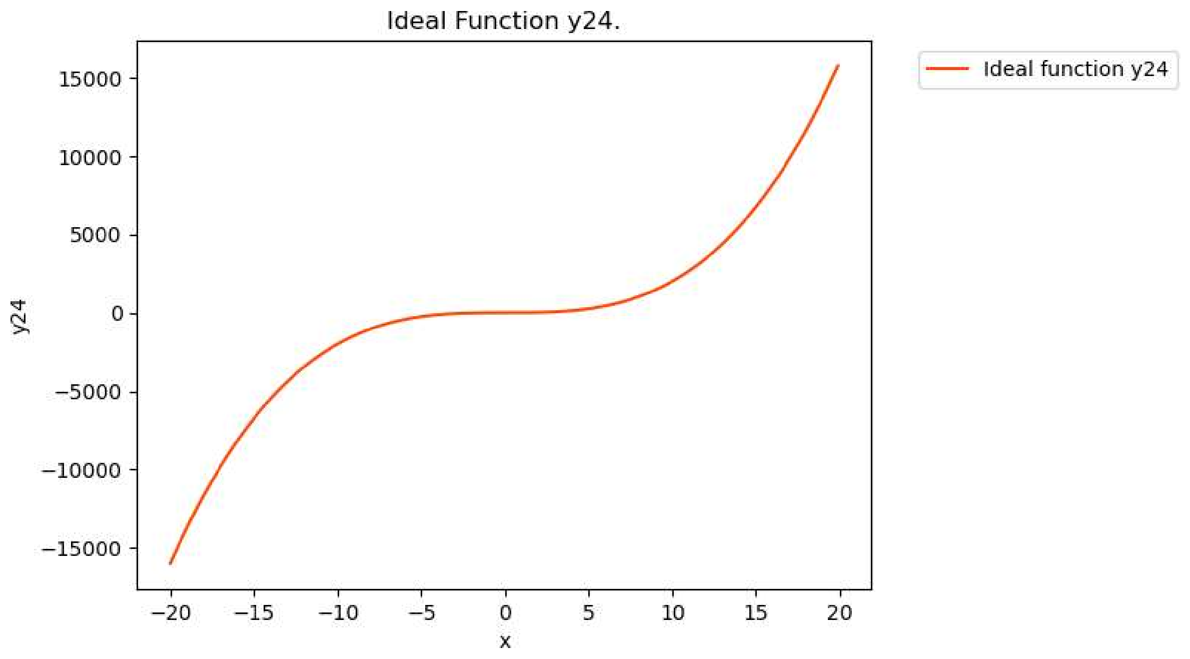

Ideal Function y19.
Ideal function y19
y19
x
400
350
300
250
200
150
100
50
0
−20
−15
−10
−5
0
5
10
15
20

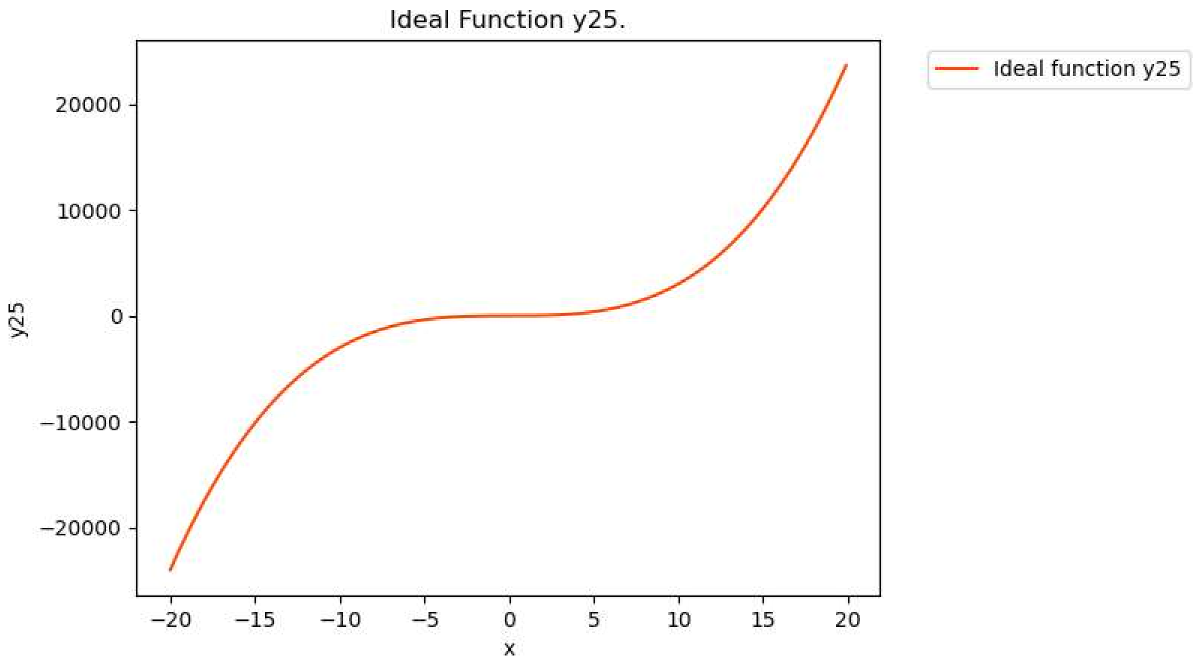

Ideal Function y20.
Ideal function y20
y20
x
500
400
300
200
100
0
−20
−15
−10
−5
0
5
10
15
20

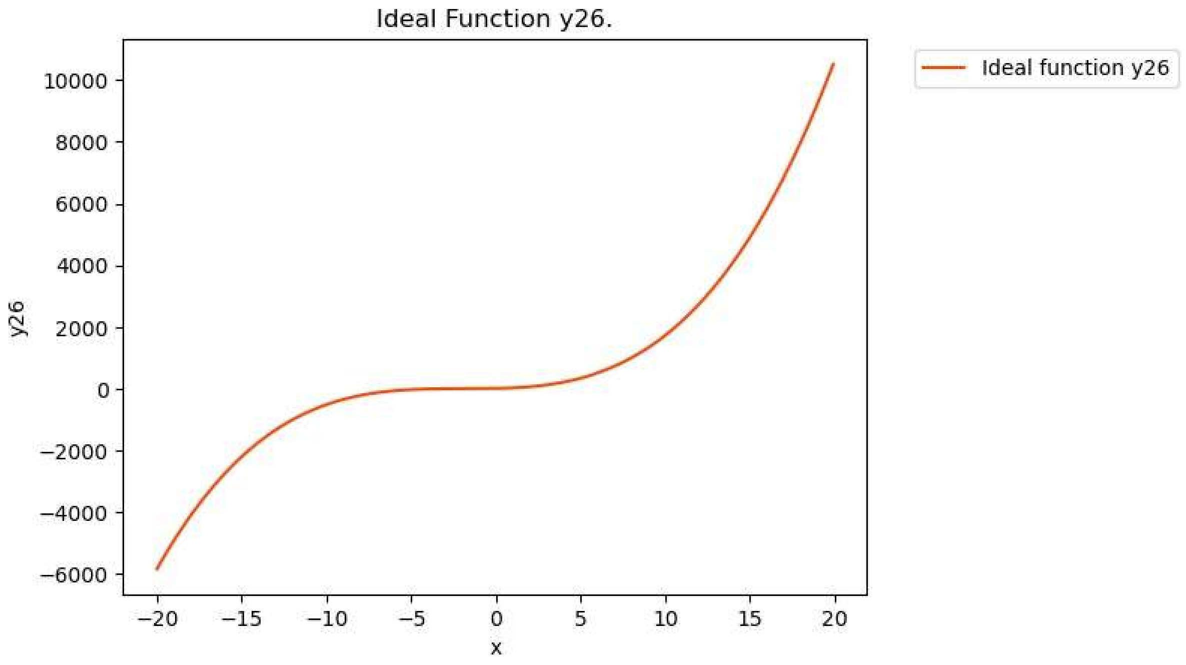

Ideal Function y21.
Ideal function y21
y21
x
8000
6000
4000
2000
0
−2000
−4000
−6000
−8000
−20
−15
−10
−5
0
5
10
15
20

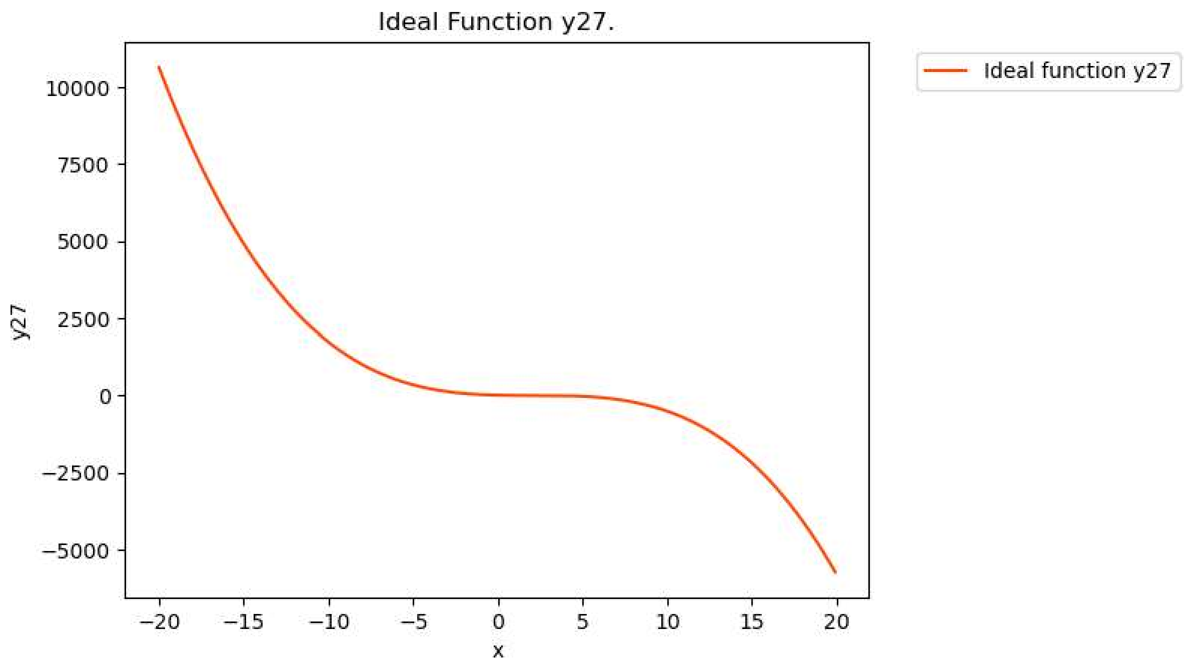

Ideal Function y22.
Ideal function y22
y22
x
8000
7000
6000
5000
4000
3000
2000
1000
0
−20
−15
−10
−5
0
5
10
15
20

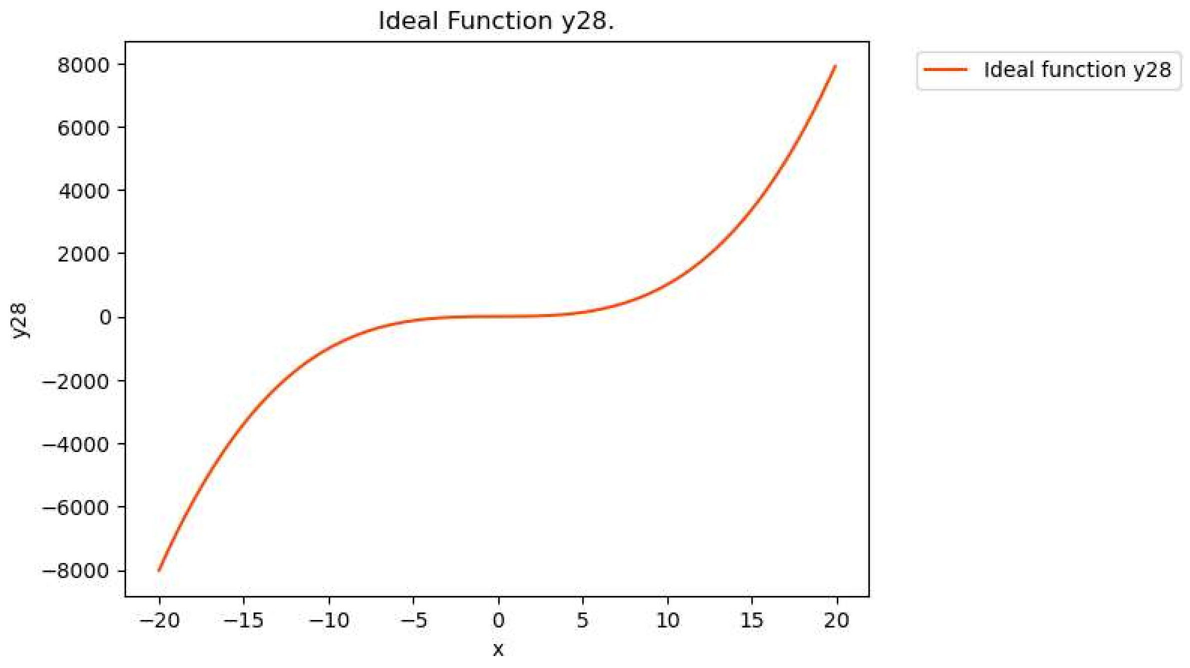

Ideal Function y23.
Ideal function y23
y23
x
8000
6000
4000
2000
0
−2000
−4000
−6000
−8000
−20
−15
−10
−5
0
5
10
15
20

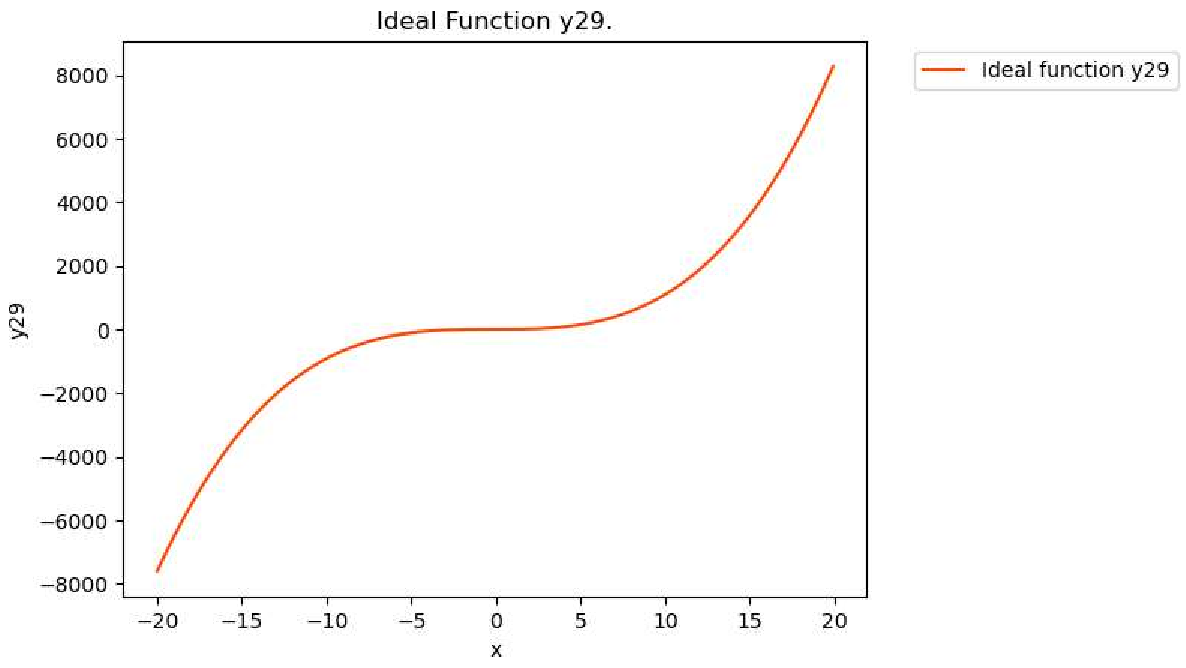

Ideal Function y24.
Ideal function y24
y24
x
15000
10000
5000
0
−5000
−10000
−15000
−20
−15
−10
−5
0
5
10
15
20

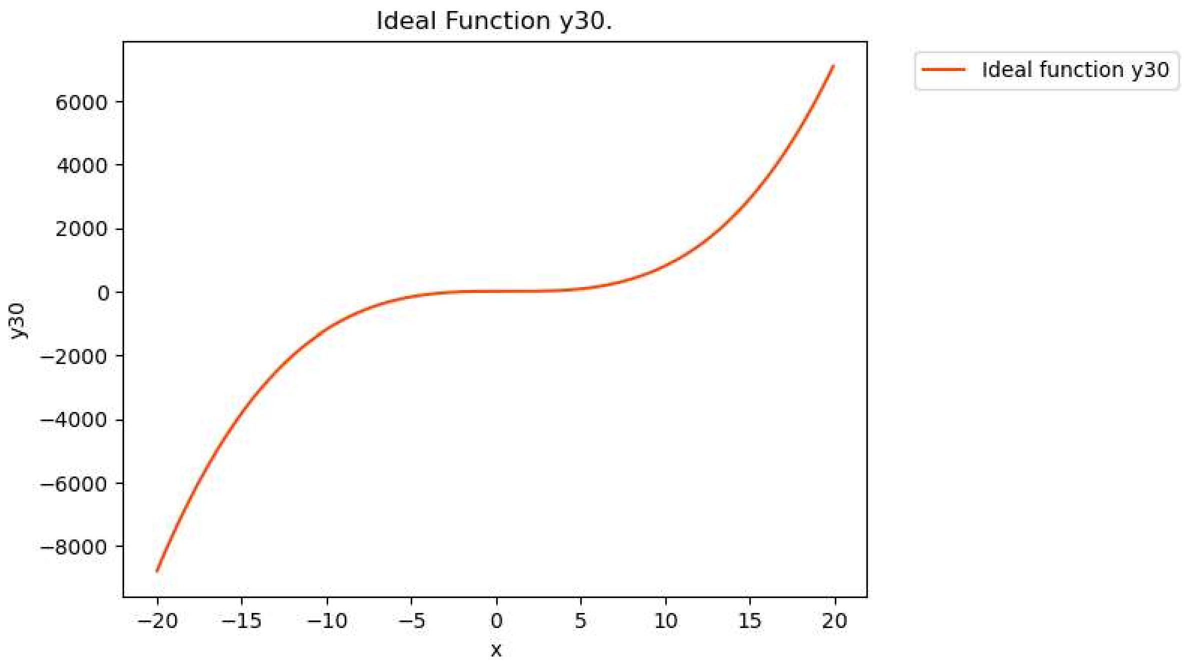

Ideal Function y25.
Ideal function y25
y25
x
20000
10000
0
−10000
−20000
−20
−15
−10
−5
0
5
10
15
20

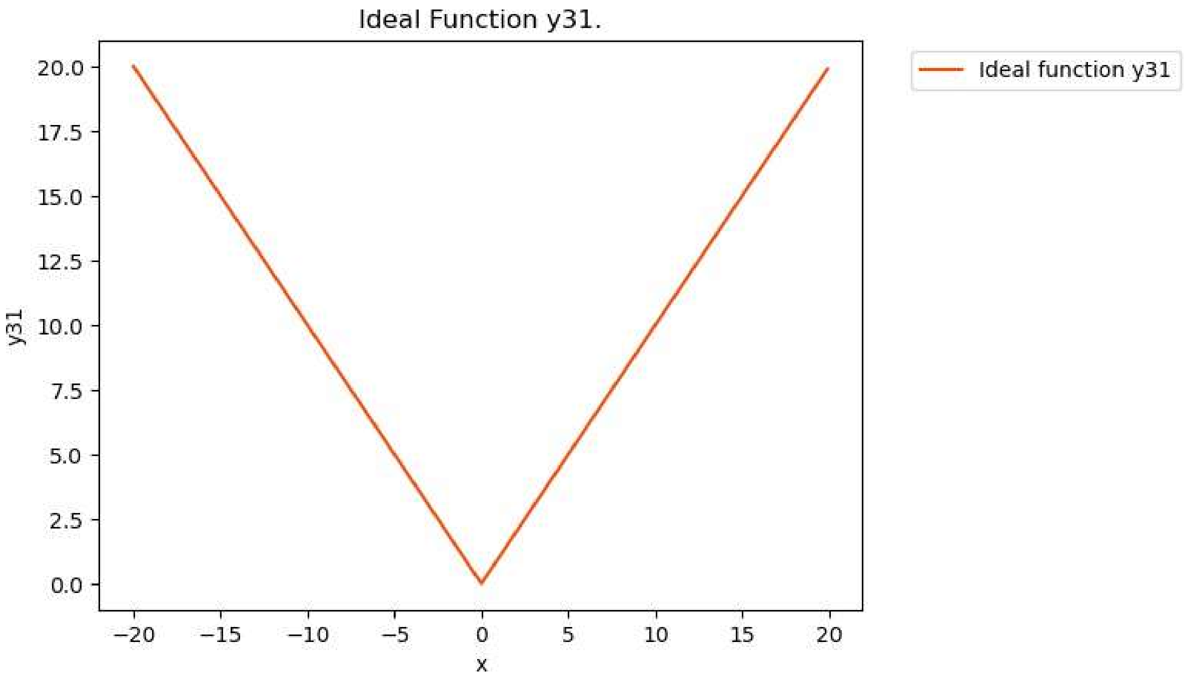

Ideal Function y26.
Ideal function y26
y26
x
10000
8000
6000
4000
2000
0
−2000
−4000
−6000
−20
−15
−10
−5
0
5
10
15
20

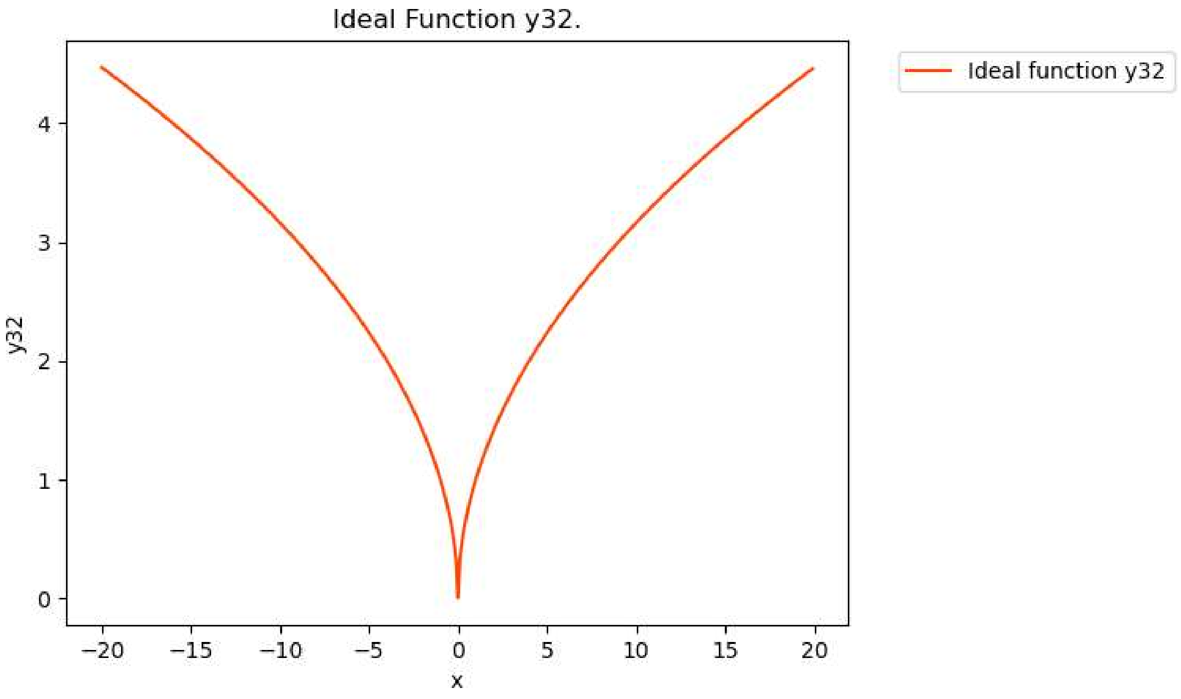

Ideal Function y27.
Ideal function y27
y27
x
10000
7500
5000
2500
0
−2500
−5000
−20
−15
−10
−5
0
5
10
15
20

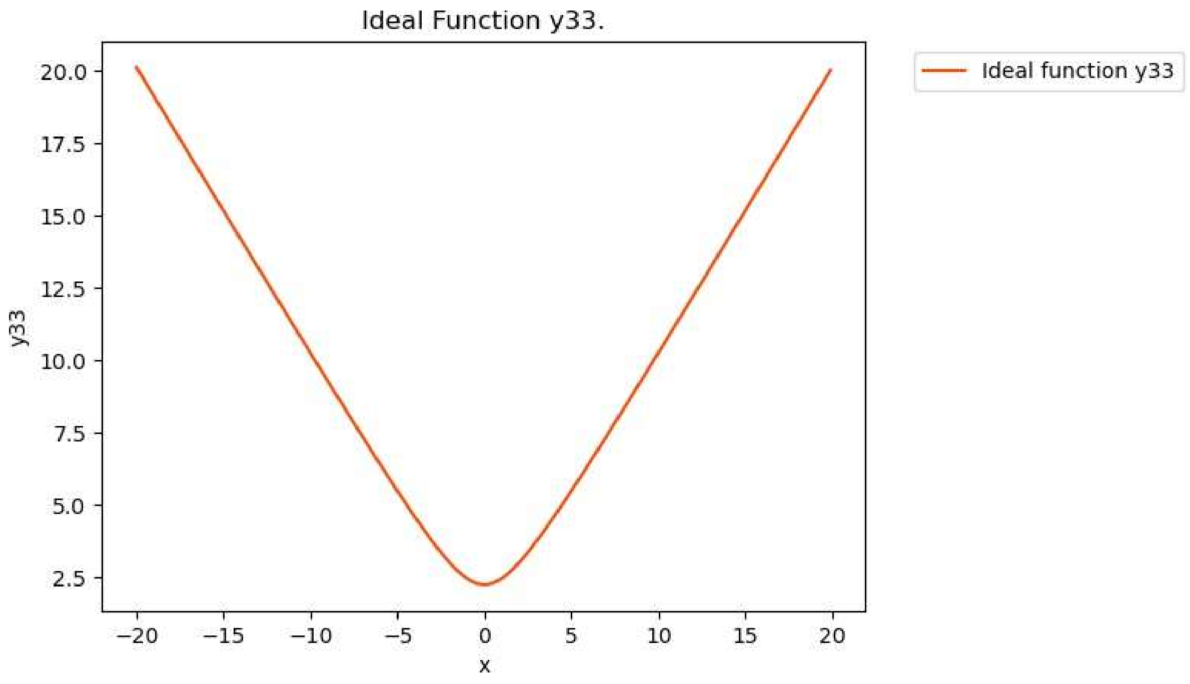

Ideal Function y28.
Ideal function y28
y28
x
8000
6000
4000
2000
0
−2000
−4000
−6000
−8000
−20
−15
−10
−5
0
5
10
15
20

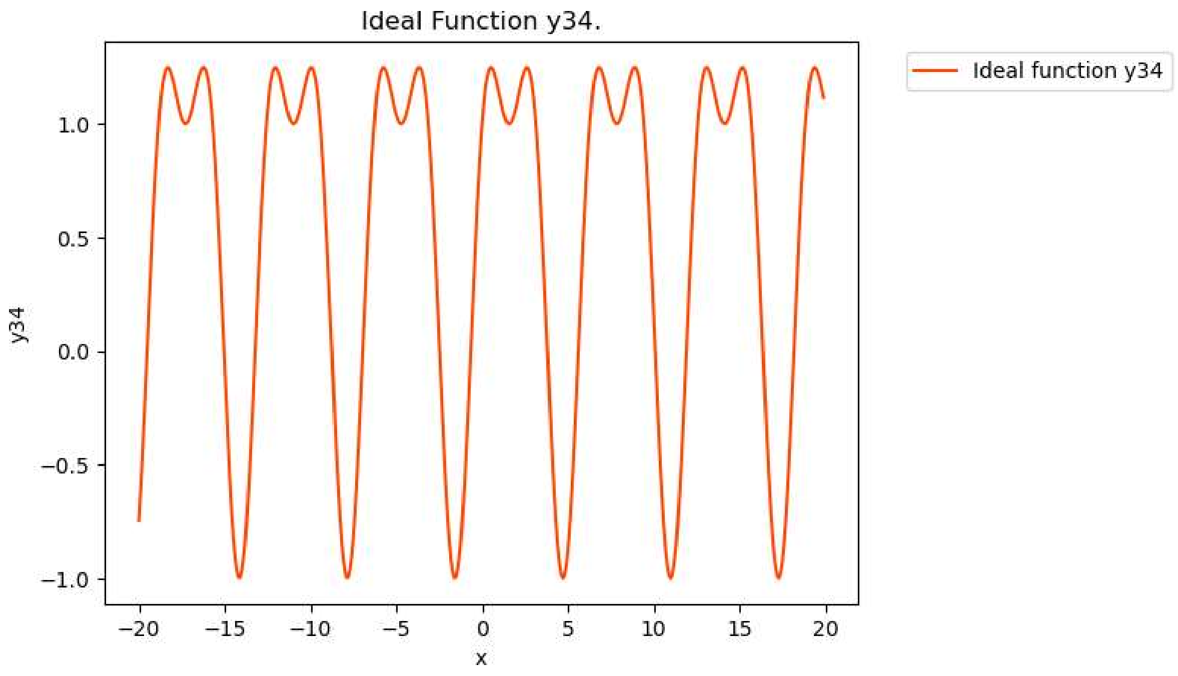

Ideal Function y29.
Ideal function y29
y29
x
8000
6000
4000
2000
0
−2000
−4000
−6000
−8000
−20
−15
−10
−5
0
5
10
15
20

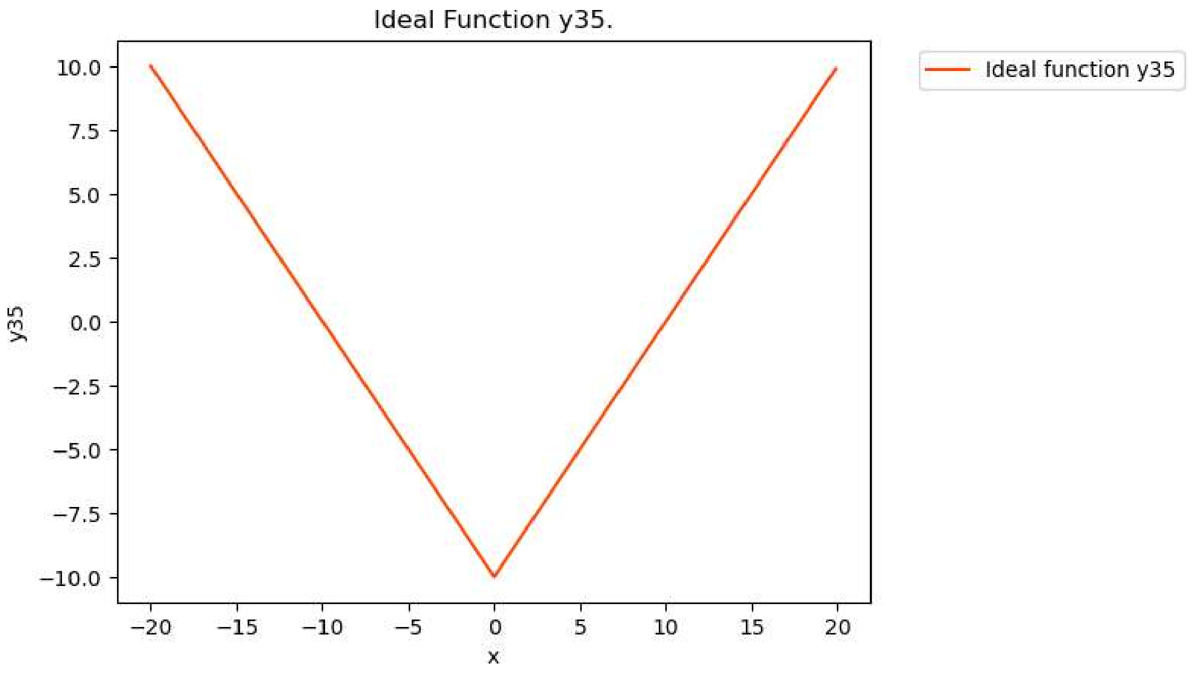

Ideal Function y30.
Ideal function y30
y30
x
6000
4000
2000
0
−2000
−4000
−6000
−8000
−20
−15
−10
−5
0
5
10
15
20

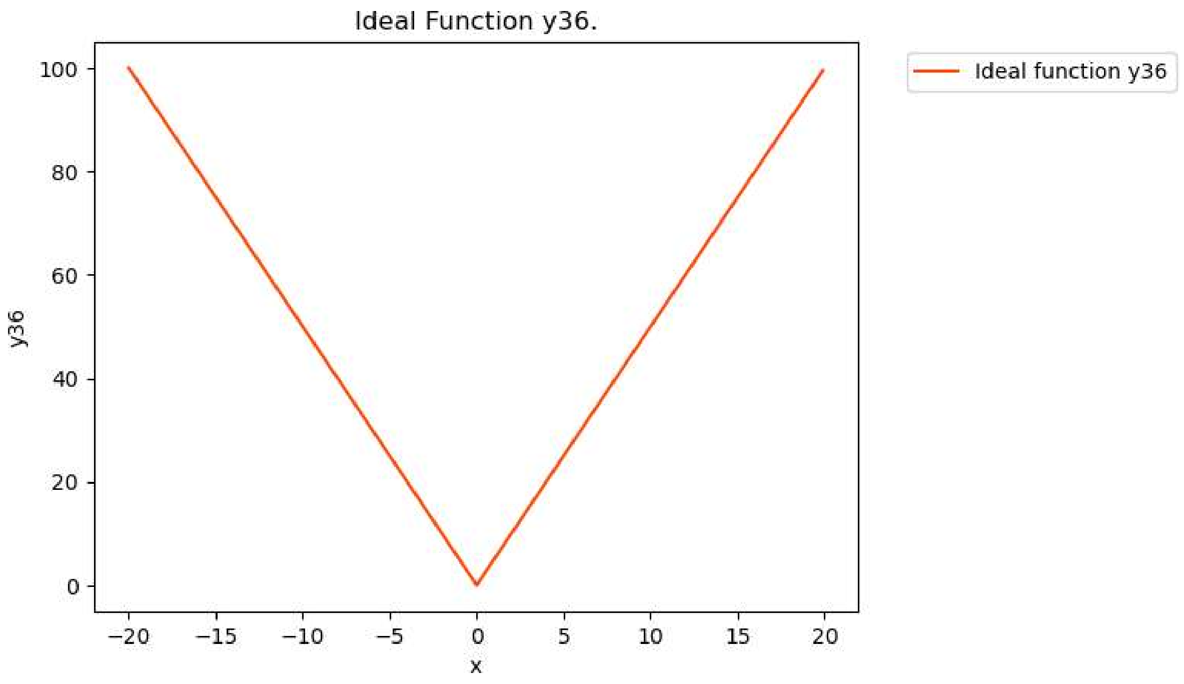

Ideal Function y31.
Ideal function y31
y31
x
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−20
−15
−10
−5
0
5
10
15
20

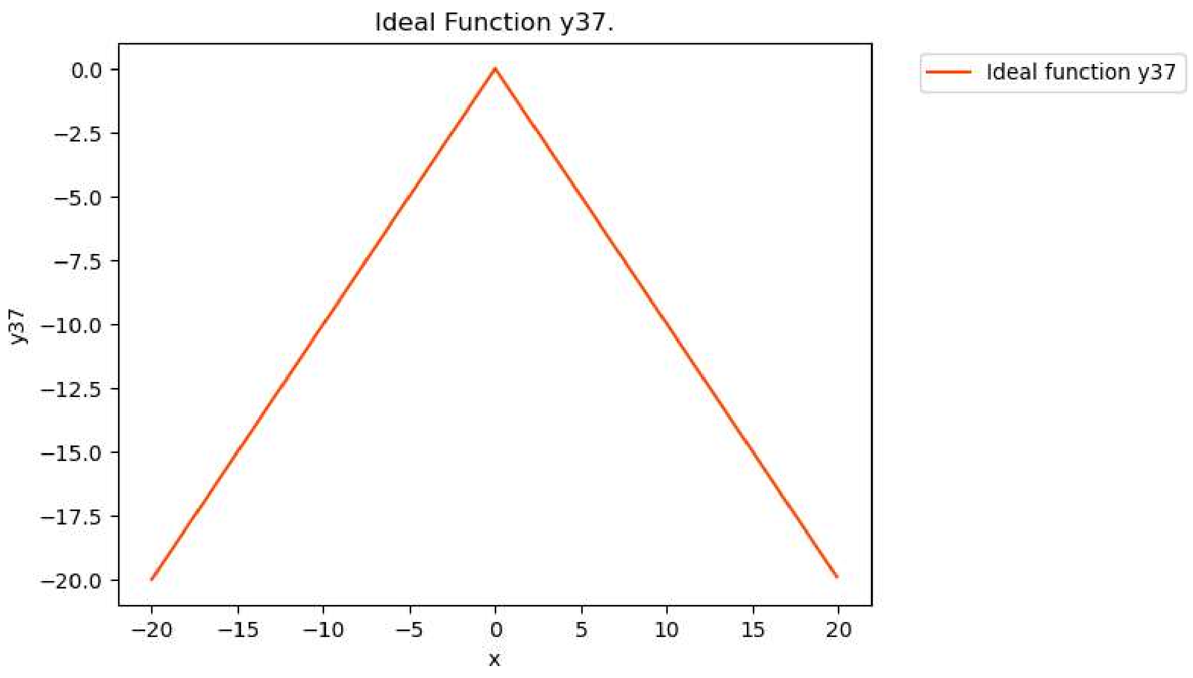

Ideal Function y32.
Ideal function y32
y32
x
4
3
2
1
0
−20
−15
−10
−5
0
5
10
15
20

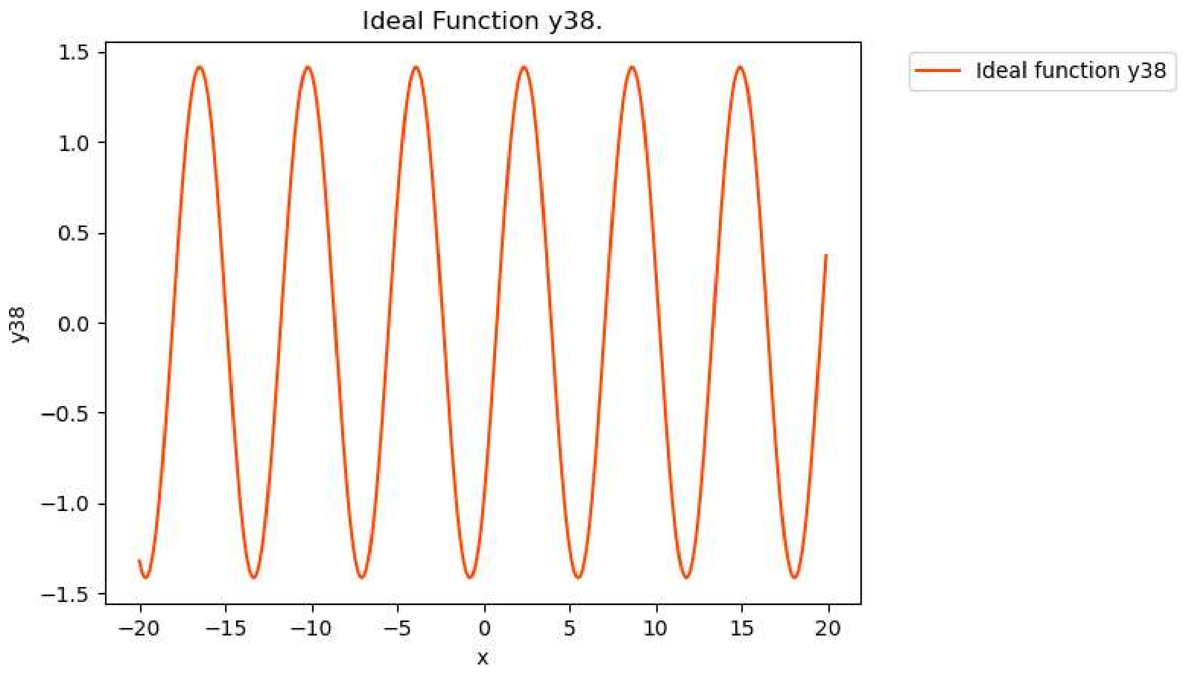

Ideal Function y33.
Ideal function y33
y33
x
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
−20
−15
−10
−5
0
5
10
15
20

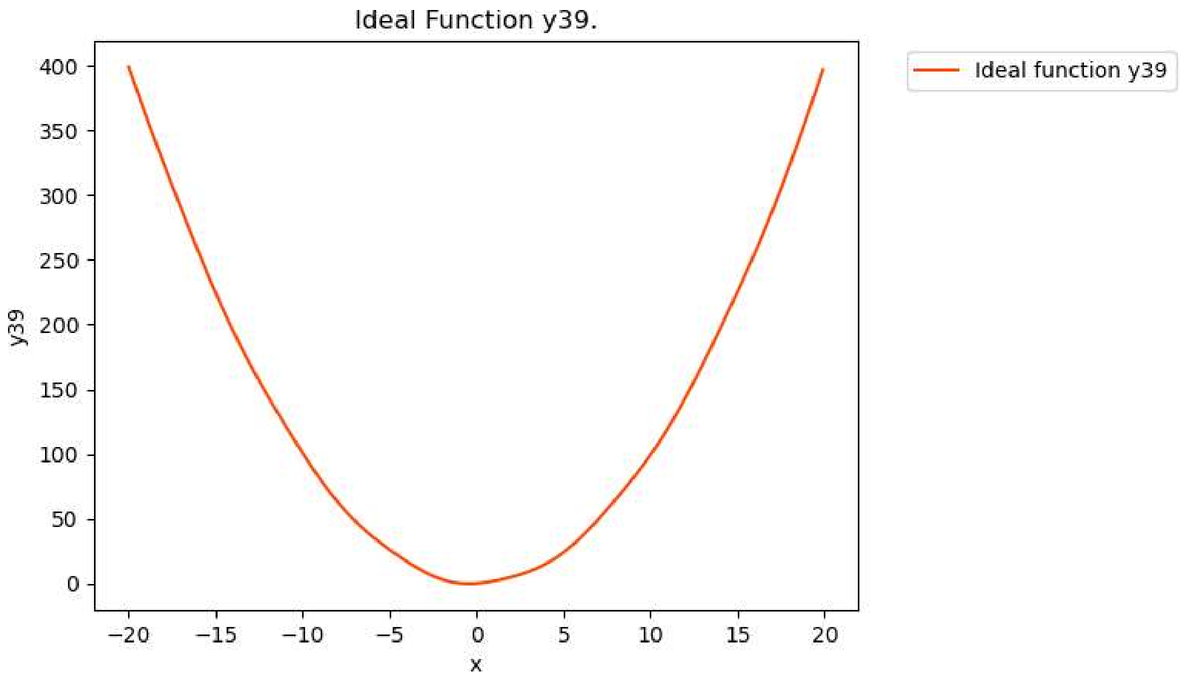

Ideal Function y34.
Ideal function y34
y34
x
1.0
0.5
0.0
−0.5
−1.0
−20
−15
−10
−5
0
5
10
15
20

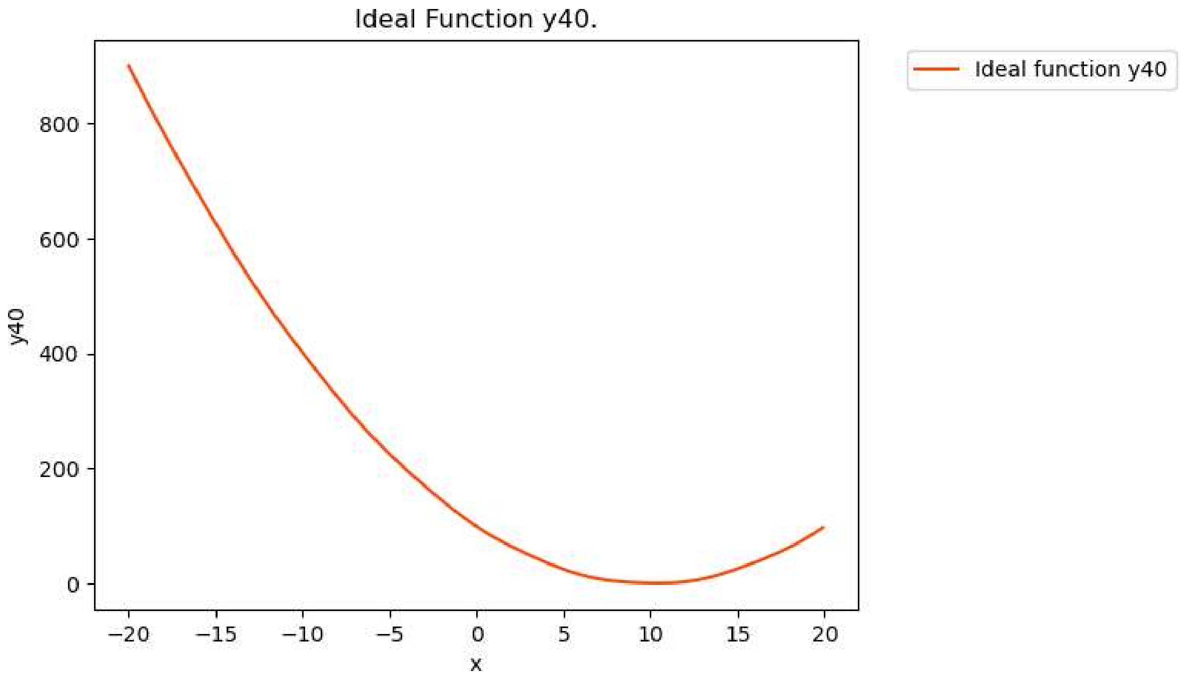

Ideal Function y35.
Ideal function y35
y35
x
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−20
−15
−10
−5
0
5
10
15
20

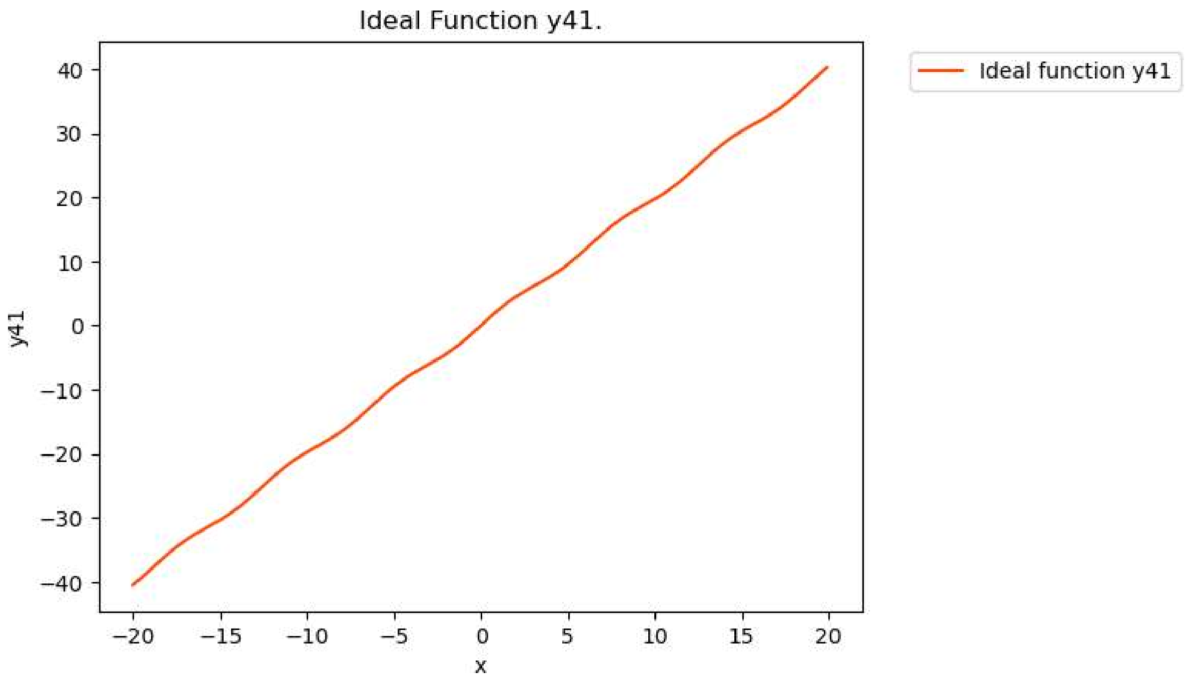

Ideal Function y36.
Ideal function y36
y36
x
100
80
60
40
20
0
−20
−15
−10
−5
0
5
10
15
20

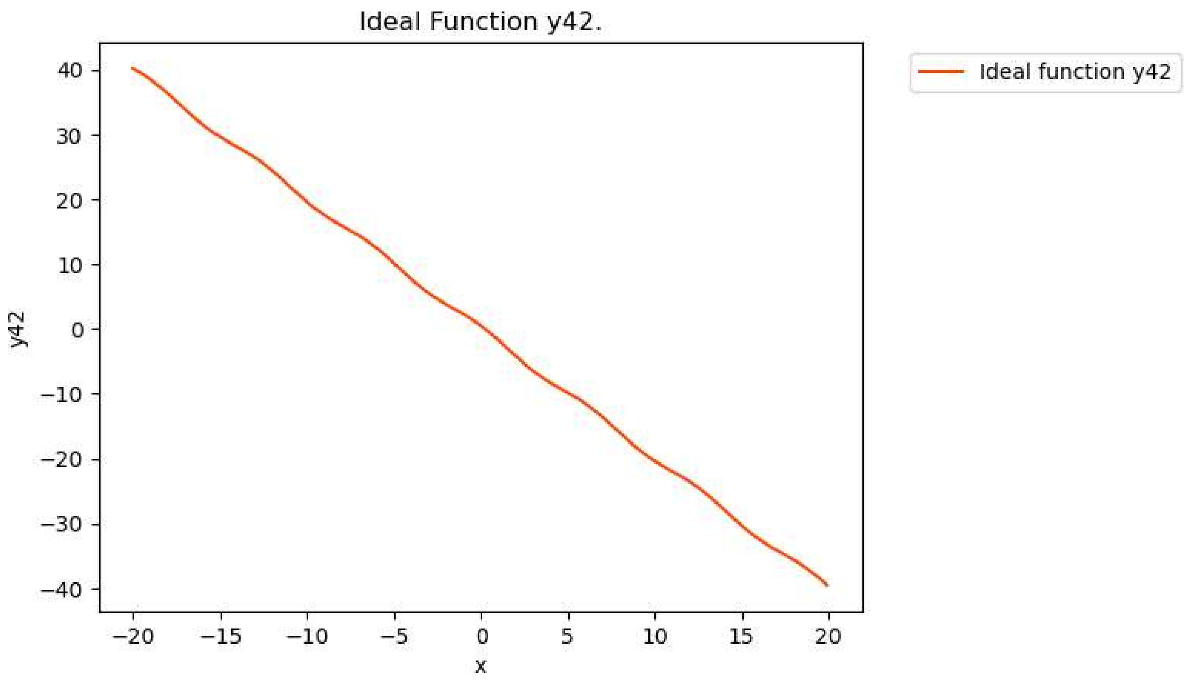

Ideal Function y37.
Ideal function y37
y37
x
0.0
−2.5
−5.0
−7.5
−10.0
−12.5
−15.0
−17.5
−20.0
−20
−15
−10
−5
0
5
10
15
20

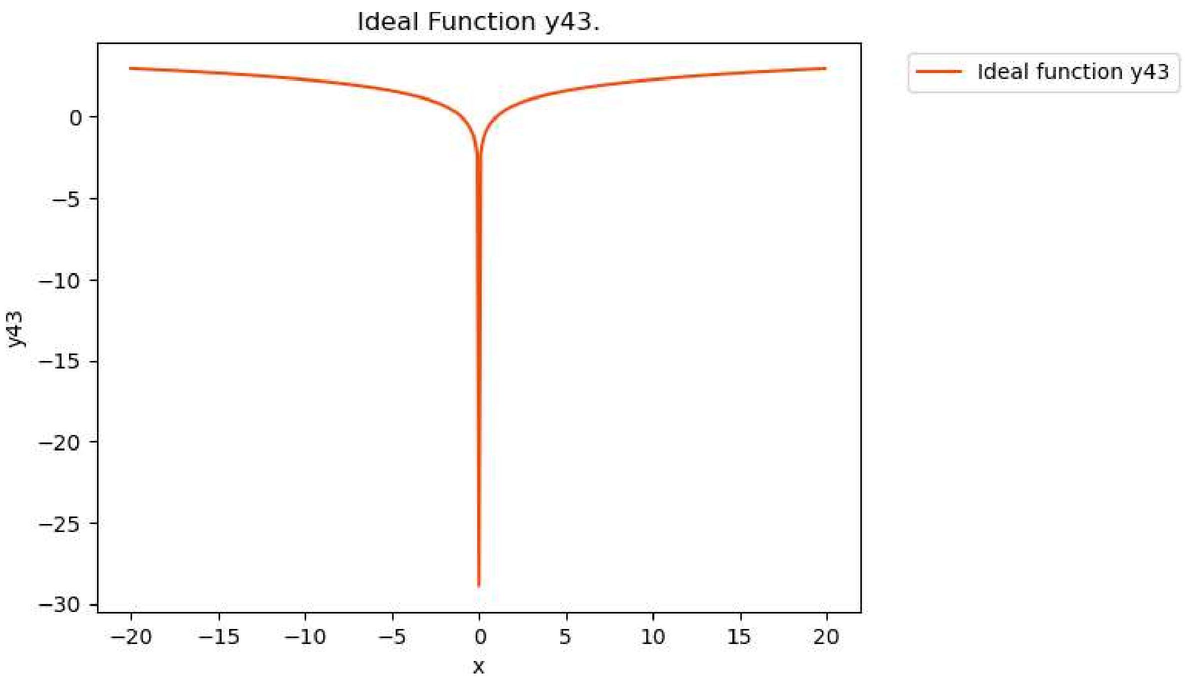

Ideal Function y38.
Ideal function y38
y38
x
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−20
−15
−10
−5
0
5
10
15
20

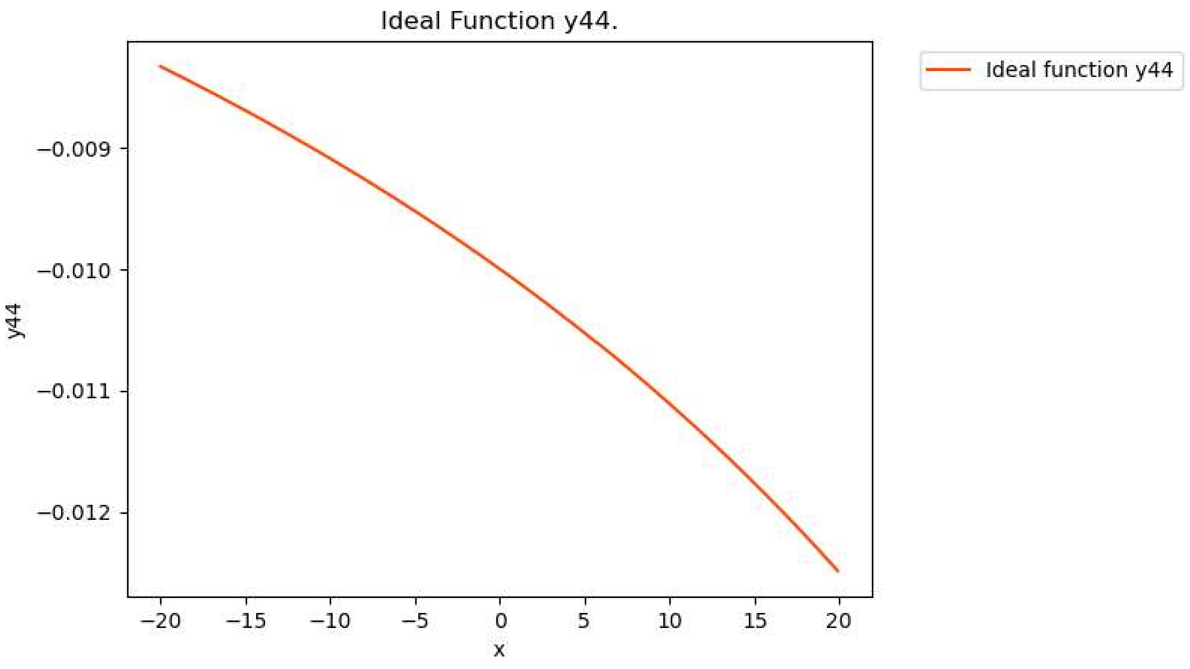

Ideal Function y39.
400
350
300
250
200
150
100
50
0
y39
−20
−15
−10
−5
0
5
10
15
20
x
Ideal function y39

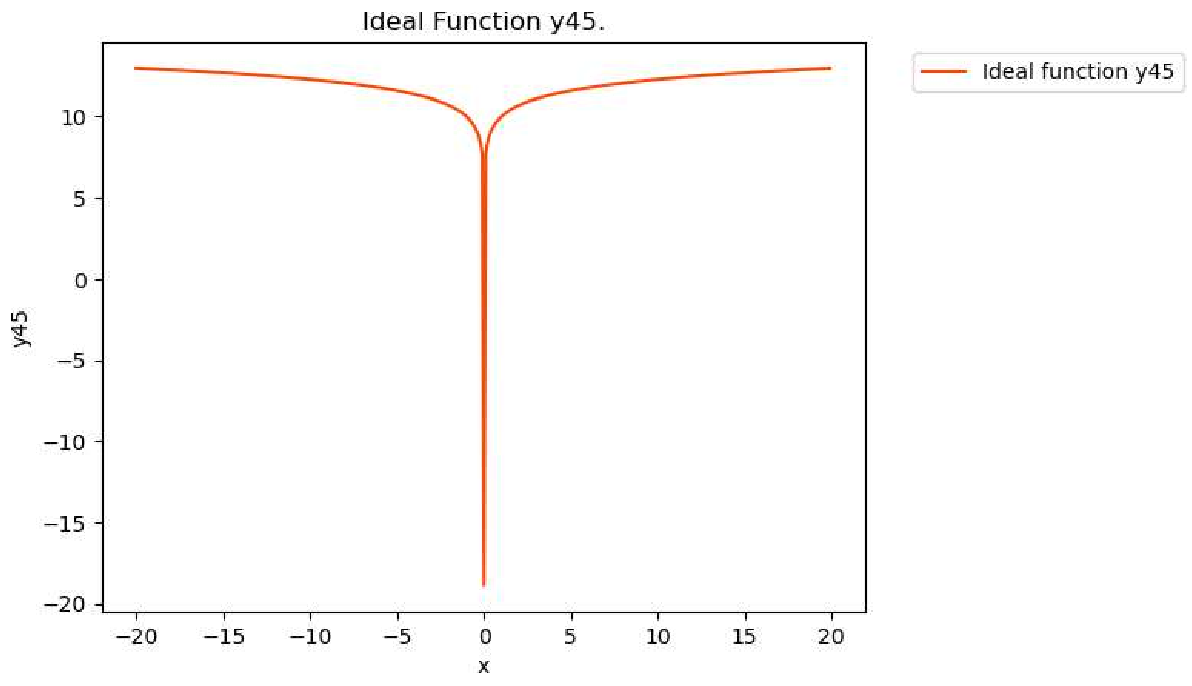

Ideal Function y40.
800
600
400
200
0
y40
−20
−15
−10
−5
0
5
10
15
20
x
Ideal function y40

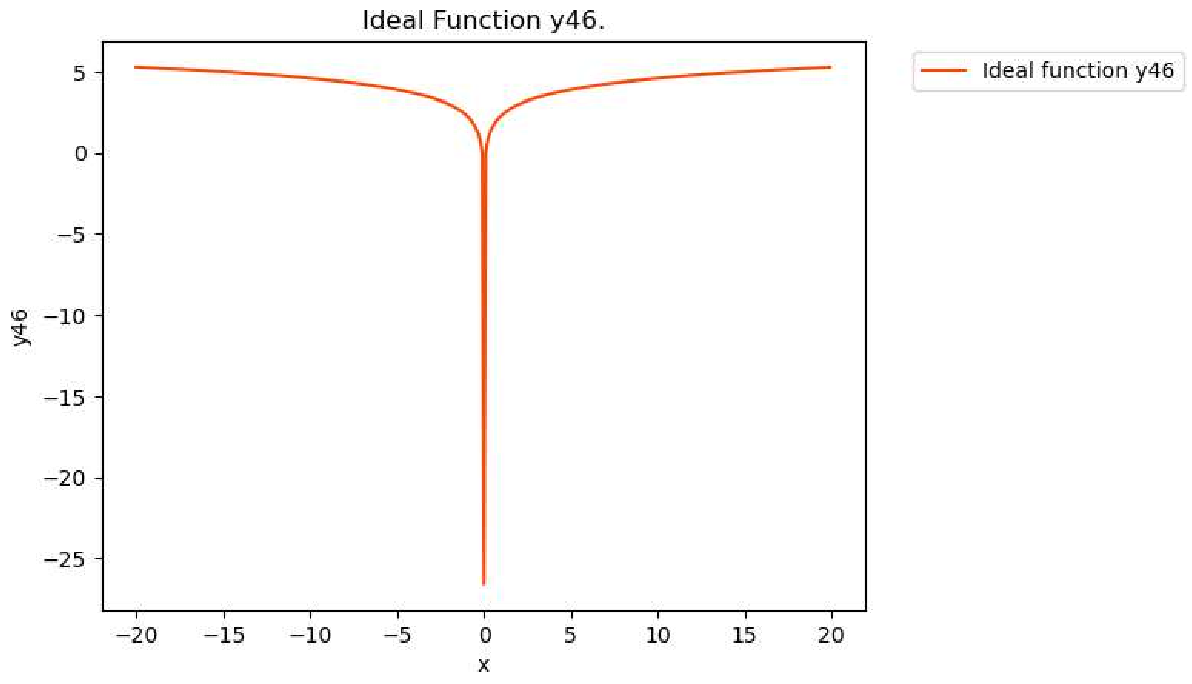

Ideal Function y41.
y41
x
Ideal function y41
40
30
20
10
0
−10
−20
−30
−40
−20
−15
−10
−5
0
5
10
15
20

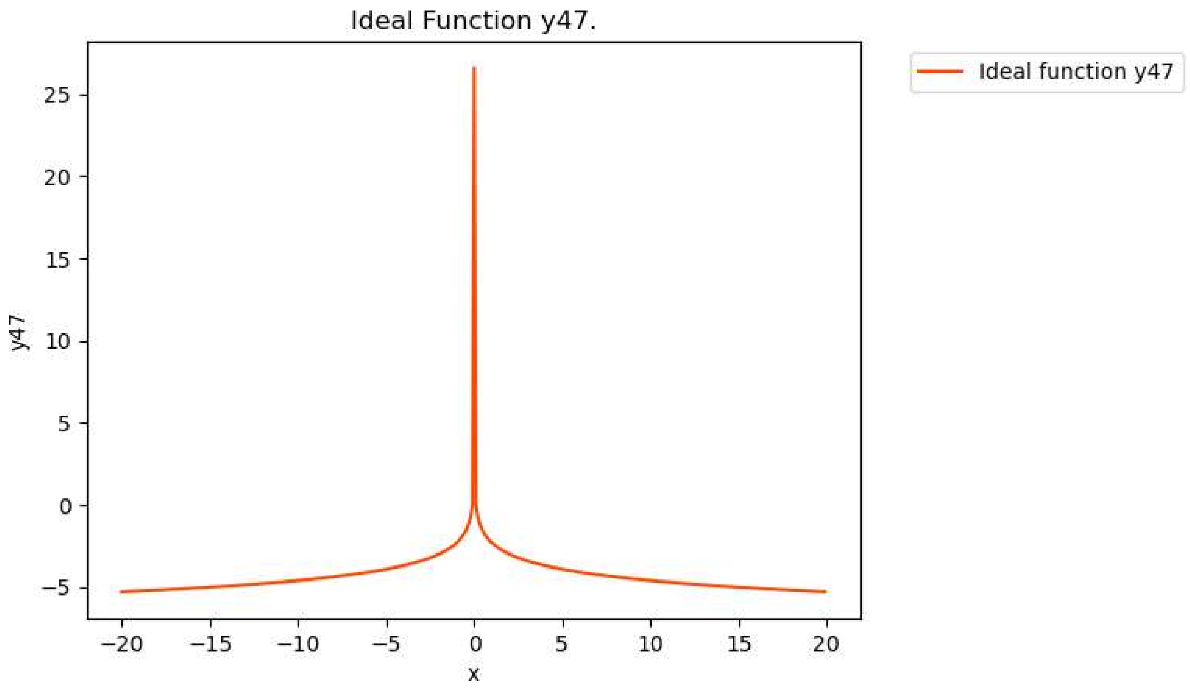

Ideal Function y42.
y42
x
Ideal function y42
40
30
20
10
0
−10
−20
−30
−40
−20
−15
−10
−5
0
5
10
15
20

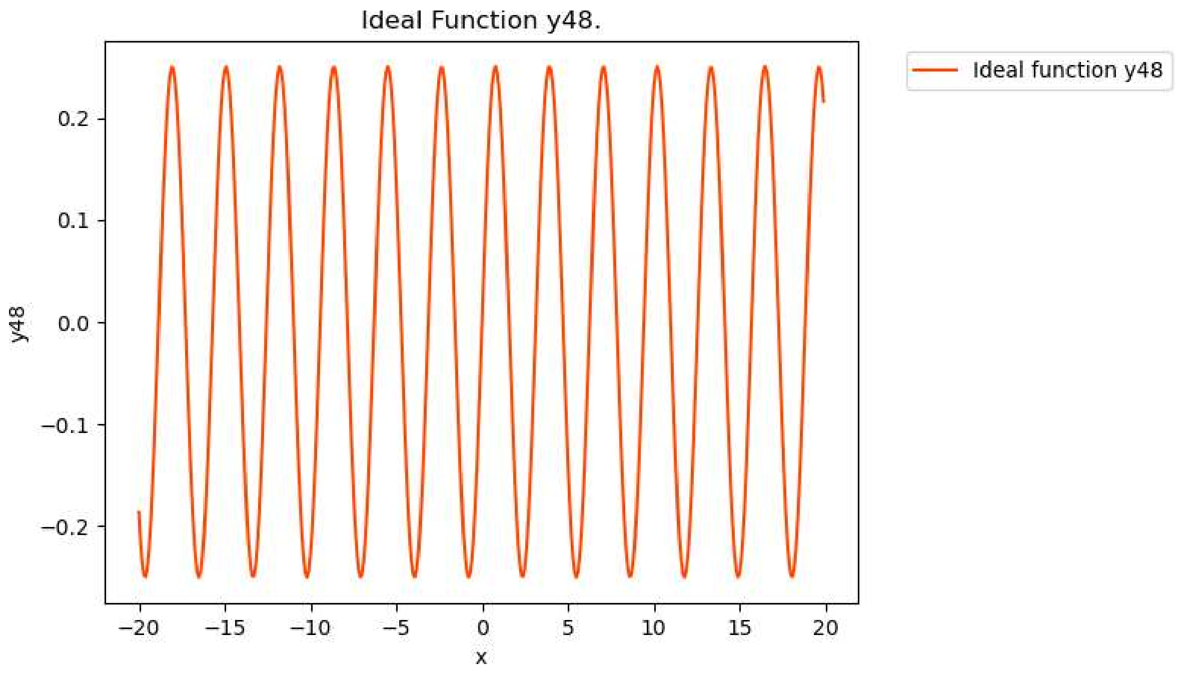

Ideal Function y43.
Ideal function y43
y43
x
0
−5
−10
−15
−20
−25
−30
−20
−15
−10
−5
0
5
10
15
20

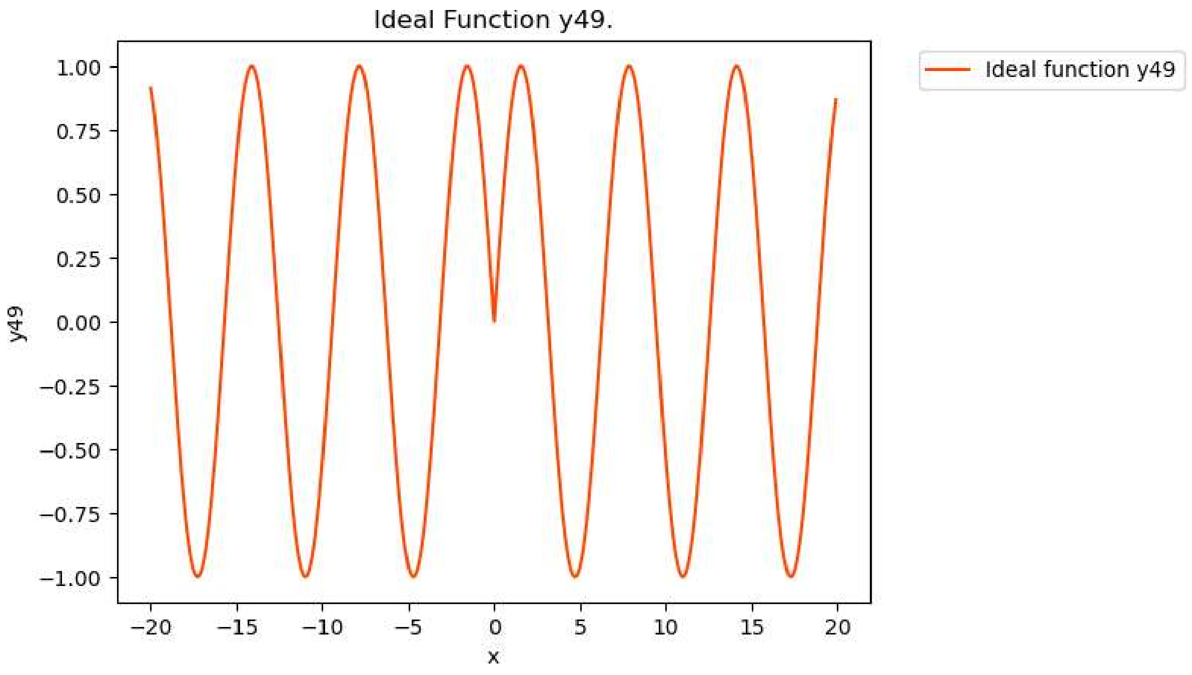

Ideal Function y44.
Ideal function y44
y44
x
−0.009
−0.010
−0.011
−0.012
−20
−15
−10
−5
0
5
10
15
20

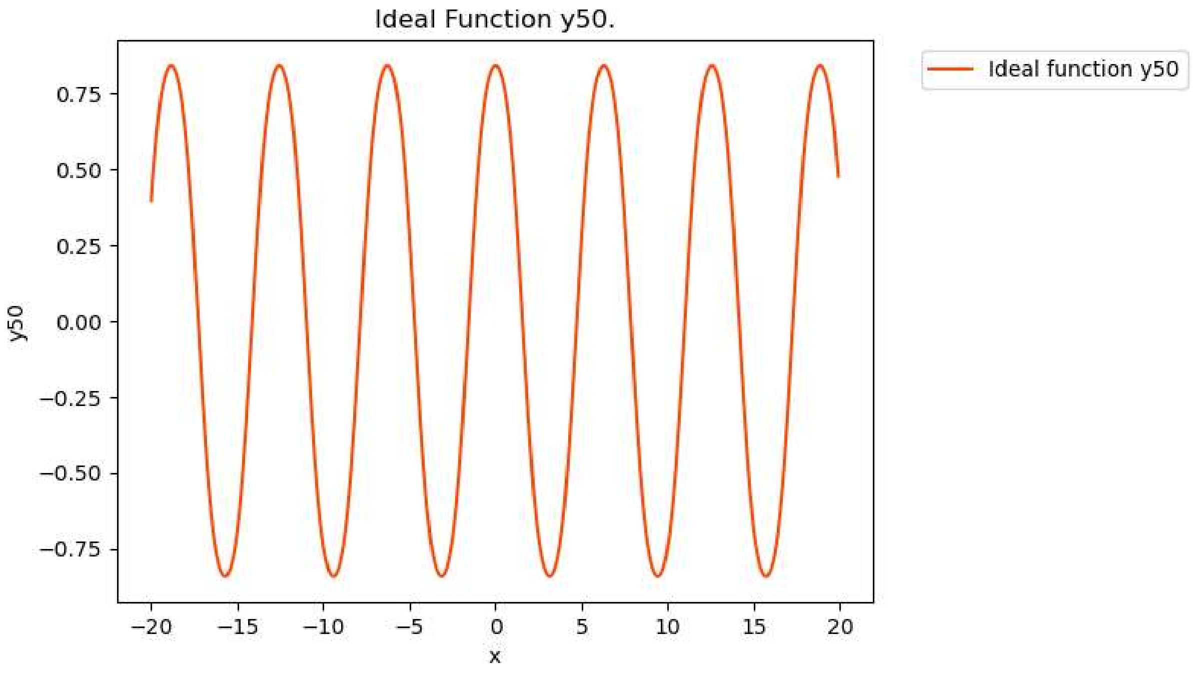

Ideal Function y45.
Ideal function y45
y45
x
10
5
0
−5
−10
−15
−20
−20
−15
−10
−5
0
5
10
15
20


Ideal Function y46.
Ideal function y46
y46
x
5
0
−5
−10
−15
−20
−25
−20
−15
−10
−5
0
5
10
15
20


Ideal Function y47.
Ideal function y47
y47
x
25
20
15
10
5
0
−5
−20
−15
−10
−5
0
5
10
15
20


Ideal Function y48.
Ideal function y48
y48
x
0.2
0.1
0.0
−0.1
−0.2
−20
−15
−10
−5
0
5
10
15
20


Ideal Function y49.
Ideal function y49
y49
x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−20
−15
−10
−5
0
5
10
15
20


Ideal Function y50.
Ideal function y50
y50
x
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−20
−15
−10
−5
0
5
10
15
20

2.6 iii) Unit testing framewor (unittest).

Unittest supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework (Python Software Foundation, 2024).

Some important object-oriented concepts of unit-test are as follows:

Concept	Definition
Test fixture	Represents the preparation needed to perform one or more tests, and any associated cleanup actions. This may involve, for example, creating temporary or proxy databases, directories, or starting a server process.
Test case	An individual unit of testing. It checks for a specific response to a particular set of inputs. Unit-test provides a base class, TestCase, which may be used to create new test cases.
Test suite	A collection of test cases, test suites, or both. It is used to aggregate tests that should be executed together.
Test runner	Is a component which orchestrates the execution of tests and provides the outcome to the user. The runner may use a graphical interface, a textual interface, or return a special value to indicate the results of executing the tests

Table 12: Unittest object-oriented concepts.

The unittest module provides a rich set of tools for constructing and running tests (Python Software Foundation, 2024). In this task, you have used a testcase that is created by subclassing unittest.TestCase. The three individual tests are defined by methods whose names start with the words test. This naming convention informs the test runner about which methods represent tests. The core of each test is a call to assertEqual(), assertNotEqual() or isinstance() to check for an expected result. Three assert methods are used in the unittest.TestCase code to accept a msg argument that, if specified, is used as the error message on failure. assertEqual(first, second, msg=None) test that first and second are equal, if the values do not compare equal, the test will fail (Python Software Foundation, 2024). assertNotEqual(first, second, msg=None) test that first and second are not equal, if the values do compare equal, the test will fail (Python Software Foundation, 2024). assertIsInstance(obj, cls, msg=None) test that an object is an instance of a class or a tuple of classes, as supported by isinstance(). The final block of the unittest code shows a simple way to run the tests. unittest.main() provides a command-line interface to the test script that produces an output.

2.6.1 Ideal unittest

Python-program:

```
import unittest
import pandas as pd
class TestDataWrangler(unittest.TestCase):
    ’’’
    Test the DataWrangler Class
    ’’’
    def test_load_data(self):
        ’’’
        test the load_data method that it successfully
        constructed a dataframe from the .csv file
        ’’’
        #self.dataWrangler = DataWrangler(”ideal.csv”)
        self.dataset = pd.read_csv(”ideal.csv”)
        self.df_data = pd.DataFrame(self.dataset)
        self.assertNotEqual(isinstance(self.df_data, pd.DataFrame), True, ”The returned value is of type DataFrame”)
        # self.assertEqual(isinstance(self.df_data, pd.DataFrame), True, ”The returned value is of type DataFrame”)
    def test_shape_of_data(self):
        ’’’
        test the shape_data method that it successfully
        returns the shape of the dataframe constructed
        ’’’
        self.dataset = pd.read_csv(”ideal.csv”)
        self.df_data = pd.DataFrame(self.dataset)
        df_shape = self.df_data.shape
        self.assertEqual(df_shape[0], 400, ”The tuple contains at index 0, the value 400,”+ ” which is our number
of rows”)
        self.assertEqual(df_shape[1], 51, ”The tuple contains at index 1, the value 51,”+ ” which is our number of
columns”)
    def test_summary_statistics(self):
        ’’’
        test the summary_statistics method that it successfully
        returns a dataframe bearing the summary statistics of each dataframe column
        ’’’
        self.dataset = pd.read_csv(”ideal.csv”)
        self.df_data = pd.DataFrame(self.dataset)
        self.df_summary = self.df_data.describe()
        self.assertEqual(self.df_summary.loc[’mean’,’x’], -0.049999999999999434, ”The mean value of x col-
umn”+ ”was truly computed”)
        self.assertEqual(self.df_summary.loc[’mean’,’y42’], 0.12280453926250061, ”The mean value of y1 col-
umn”+ ” was truly computed”)
        self.assertEqual(self.df_summary.loc[’mean’,’y41’], -0.10114118499999904, ”The mean value of y2
column”+ ” was truly computed”)
        self.assertEqual(self.df_summary.loc[’mean’,’y11’], -0.049999999999999434, ”The mean value of y3
column”+ ” was truly computed”)
        self.assertEqual(self.df_summary.loc[’mean’,’y48’], -0.0004656956999996426, ”The mean value of y4
column”+ ” was truly computed”)
if __name__ == ”__main__”:
    unittest.main()
```

Output of the ideal unittest for the selected best fit of four functions based on train dataset:

```
F
.
```

```
.
======================================================================
FAIL: test_load_data (__main__.TestDataWrangler)
four best fit ideal functions
----------------------------------------------------------------------
Traceback (most recent call last):
  File "ideal-four-unit-test.py", line 17, in test_load_data
    self.assertNotEqual(isinstance(self.df_data, pd.DataFrame), True, "The returned value is of type DataFrame")
AssertionError: True == True : The returned value is of type DataFrame
----------------------------------------------------------------------
Ran 3 tests in 1.099s
FAILED (failures=1)
** Process exited - Return Code: 1 **
Press Enter to exit terminal
```

Output of the ideal unittest for all of 50 functions (code is provided in the appendices and materials chapter):

```
F
.
.
======================================================================
FAIL: test_load_data (__main__.TestDataWrangler)
four best fit ideal functions
----------------------------------------------------------------------
Traceback (most recent call last):
  File "ideal-all-50-unit-test.py", line 17, in test_load_data
    self.assertNotEqual(isinstance(self.df_data, pd.DataFrame), True, "The returned value is of type DataFrame")
AssertionError: True == True : The returned value is of type DataFrame
----------------------------------------------------------------------
Ran 3 tests in 0.645s
FAILED (failures=1)
** Process exited - Return Code: 1 **
Press Enter to exit terminal
```

2.6.2 Test unittest

Python-program:

```
import unittest
import pandas as pd
class TestDataWrangler(unittest.TestCase):
  '''
  Test the DataWrangler Class
  '''
```

```
    def test_load_data(self):
        ”’
        test the load_data method that it successfully
        constructed a dataframe from the .csv file
        ”’
        #self.dataWrangler = DataWrangler(”test.csv”)
        self.dataset = pd.read_csv(”test.csv”)
        self.df_data = pd.DataFrame(self.dataset)
        self.assertNotEqual(isinstance(self.df_data, pd.DataFrame), True, ”The returned value is of type DataFrame”)
        # self.assertEqual(isinstance(self.df_data, pd.DataFrame), True, ”The returned value is of type DataFrame”)
    def test_shape_of_data(self):
        ”’
        test the shape_data method that it successfully
        returns the shape of the dataframe constructed
        ”’
        self.dataset = pd.read_csv(”test.csv”)
        self.df_data = pd.DataFrame(self.dataset)
        df_shape = self.df_data.shape
        self.assertEqual(df_shape[0], 100, ”The tuple contains at index 0, the value 100,”+ ” which is our number
of rows”)
        self.assertEqual(df_shape[1], 2, ”The tuple contains at index 1, the value 2,”+ ” which is our number of
columns”)
    def test_summary_statistics(self):
        ”’
        test the summary_statistics method that it successfully
        returns a dataframe bearing the summary statistics of each dataframe column
        ”’
        self.dataset = pd.read_csv(”test.csv”)
        self.df_data = pd.DataFrame(self.dataset)
        self.df_summary = self.df_data.describe()
        self.assertEqual(self.df_summary.loc[’mean’,’x’], 0.299000000000003, ”The mean value of x column”+
”was truly computed”)
        self.assertEqual(self.df_summary.loc[’mean’,’y’], 0.3254828044499999, ”The mean value of y column”+
” was truly computed”)
if __name__ == ”__main__”:
    unittest.main()
```

Output of the test unittest:

```
F
.
.
======================================================================
FAIL: test_load_data (__main__.TestDataWrangler)
```

```
test the load_data method that it successfully
----------------------------------------------------------------------
Traceback (most recent call last):
  File "Test-unit-test.py", line 16, in test_load_data
    self.assertNotEqual(isinstance(self.df_data, pd.DataFrame), True, "The returned value is of type DataFrame")
AssertionError: True == True : The returned value is of type DataFrame
----------------------------------------------------------------------
Ran 3 tests in 0.018s
FAILED (failures=1)

** Process exited - Return Code: 1 **
Press Enter to exit terminal
```

2.6.3 Train unittest

Python-program:

```
import unittest
import pandas as pd
class TestDataWrangler(unittest.TestCase):
    '''
    Test the DataWrangler Class
    '''
    def test_load_data(self):
        '''
        test the load_data method that it successfully
        constructed a dataframe from the .csv file
        '''
        #self.dataWrangler = DataWrangler("train.csv")
        self.dataset = pd.read_csv("train.csv")
        self.df_data = pd.DataFrame(self.dataset)
        self.assertNotEqual(isinstance(self.df_data, pd.DataFrame), True, "The returned value is of type DataFrame")
        # self.assertEqual(isinstance(self.df_data, pd.DataFrame), True, "The returned value is of type DataFrame")
    def test_shape_of_data(self):
        '''
        test the shape_data method that it successfully
        returns the shape of the dataframe constructed
        '''
        self.dataset = pd.read_csv("train.csv")
        self.df_data = pd.DataFrame(self.dataset)
        df_shape = self.df_data.shape
        self.assertEqual(df_shape[0], 400, "The tuple contains at index 0, the value 400,"+ " which is our number
of rows")
        self.assertEqual(df_shape[1], 5, "The tuple contains at index 1, the value 5,"+ " which is our number of
columns")
    def test_summary_statistics(self):
        '''
        test the summary_statistics method that it successfully
        returns a dataframe bearing the summary statistics of each dataframe column
        '''
        self.dataset = pd.read_csv("train.csv")
        self.df_data = pd.DataFrame(self.dataset)
        self.df_summary = self.df_data.describe()
        self.assertEqual(self.df_summary.loc['mean','x'], -0.049999999999999434, "The mean value of x column"+ "was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y1'], 0.10766606164249878, "The mean value of y1 column"+ " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y2'], -0.09423904243749917, "The mean value of y2 column"+ " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y3'], -0.05162830643749942, "The mean value of y3 column"+ " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y4'], 0.01263280748595, "The mean value of y4 column"+
" was truly computed")
if __name__ == "__main__":
    unittest.main()
```

Output of the train unittest:

```
F
.
.
======================================================================
FAIL: test_load_data (__main__.TestDataWrangler)
test the load_data method that it successfully
----------------------------------------------------------------------
Traceback (most recent call last):
    File "Trainunittest.py", line 16, in test_load_data
        self.assertNotEqual(isinstance(self.df_data, pd.DataFrame), True, "The returned value is of type DataFrame")
AssertionError: True == True : The returned value is of type DataFrame
----------------------------------------------------------------------
Ran 3 tests in 0.204s
FAILED (failures=1)

** Process exited - Return Code: 1 **
Press Enter to exit terminal
```

3 Additional Task: Version Control System Git.

Git is a version control system. It helps you to keep track of code changes and it is used to collaborate on code. (W3 schools, 2024).
Some functions of Git: projects management with Repositories, clone a project to work on a local copy, branch and merge to allow for work on different parts and versions of a project, pull the latest version of the project to a local copy, and push local updates to the main project (W3 schools, 2024).
Working with Git: Initialize Git on a folder to make it a Repository. Git now creates a hidden folder to keep track of changes in that folder (W3 schools, 2024). A file is considered modified when it is changed, added or deleted. You are allowed to select the modified files you want to Stage. The Staged files are Committed and that prompts Git to store a permanent snapshot of the files. Git allows you to see the full history of every commit. This means you can revert back to any previous commit. Git grant developers an opportunity to work together from anywhere in the world (W3 schools, 2024).
What is GitHub? GitHub is different from Git. GitHub makes tools that use Git.
Using Git with Command Line: For Windows, you can use Git bash, which is included in Git for Windows. For Mac and Linux you can use the built-in terminal (W3 schools, 2024).
Firstly, check if Git is properly installed.
git –version
git version 2.47.0.windows.2

Configure Git: introduce your information to Git.
git config –global user.name ”python-task”
git config –global user.email ”lesedi.matshehla@iu-study.org”
Create a new folder for your task: mkdir makes a new directory and cd changes the current working directory.
mkdir programmingwithpython
cd programmingwithpython
Initialize Git on the created folder:
git init
Initialized empty Git repository in C:/Users/lesed/programmingwithpython/.git/

ls will list the files in the directory. It is currently empty.
git status is used to check the status of your repository.
Now save all of the programs you have created for this task in the Git folder programmingwithpython and enter the command ls.
ls
Data-Wrangler python-program for chapter 1.ipynb, Data-Wrangler python-program for chapter 1.md, ideal-four-unit-test.ipynb, ideal-four-unit-test.py, Python-program that uses training data to choose best fit of four ideal functions part 1.ipynb, Python-program that uses training data to choose best fit of four ideal functions part 1.zip, Test unit-test.ipynb, Test unit-test.py, Train unit-test.ipynb, Train unit-test.py, Unit-test python-program of

all datasets.ipynb, Unit-test python-program of all datasets.md, Unit-test python-program of all datasets.py.
Then check the Git status and see if the saved files are part of your repository. For example:
```
git status
On branch master
Changes not staged for commit:
Untracked files:
    (use "git add ..." to include in what will be committed)
```
Add all files in the current directory to the Staging Environment:
```
git add –all
git status
On branch master
Changes to be committed:
    (use "git restore –staged <file>..." to unstage)
```
Git Commit changes the files from stage to commit for your repository.
Add clear messages to each commit to clarify what has changed. The output will show list of the created files.
```
git commit -m "Python-programs are added to repository."
[master 30b5e9b] Python-programs are added to repository.
```
Git Commit Log: use the log command to view the history of commits for a repository.
```
git log
commit 30b5e9bbbf9bfb96211da7c2aa5c4c57a39dd418 (HEAD -> master)
Author: python-task <lesedi.matshehla@iu-study.org>
Date: Fri Nov 15 18 : 27 : 28 2024 +0200
    Python-programs are added to repository.
```
Enter git command -help to see all the available options for the specific command and git help –all to see all possible commands. In Git, a branch is a new/separate version of the main repository (W3 schools, 2024). Your task is to create a new branch called "develop". Then confirm that you have created a new branch develop and move your current workspace from the master branch, to the new branch.
```
git branch develop
git branch
    develop
* master
git checkout develop
Switched to branch 'develop'
git add –all
git status
On branch develop
nothing to commit, working tree clean
```

Finally, add the pdf document 20241115_Lesedi_Matshehla_3210470_DLMDSPWP01 and the latex overleaf source you have used to complete your assignment to the branch develop. Enter these command lines to answer the additional task completely: git push origin master, git remote add origin <server>, git push origin develop, git pull, git merge develop.

Conclusion

Importing data into Python is very useful in data science, big data analytics, and other related fields. The ability to obtain insights and patterns from this data is becoming increasingly important due to the consistent rise in data generation. Flexibility and variety of modules made Python to be a perfect tool for data analysis, whether you are dealing with small or big datasets. In this task, we have explored read_csv() technique of how to load data into dataframe in Python using pandas (Richman, 2023). After importing data into Python, you should know how to find the shape of dimension of DataFrame. The shape of a dataset stated number of rows and number of columns (Geeks for Geeks, 2023). Summary statistics are fundamental components of data analysis that provide essential insights into datasets. Analysts can effectively summarize and interpret data by understanding measures of central tendency, variability, and distribution shape. This led to informed decision-making across various fields of data science and computing (Story of Mathematics, 2024).

You have observed that missing values are not available in the datasets. Python-program that investigated missing values have not returned any missing values. Following this, the code for filling missing values failed because there was no missing values to replace. You are encouraged to figure out missing values and deal with them extensively. The impact of incomplete datasets could mean a serious problem on the method you have used for data collection. If a survey is not properly designed, digital data capture system have some connection issues or population failed to complete all variables required for data collection, the organization responsible for data collection can use the results of missing values to improve developments of qualitative and quantitative data distribution to reach successful products. Duplicated rows can also affect the conclusion of data analytics. For this reason, it is crucial to figure out duplicate rows and remove them. Fortunately, there is an option to keep rows that will not return duplicate rows. An outlier is not always a form of incorrect data, this means you have to be careful with them in data cleansing. What you should do with an outlier depends on its most likely cause (Bhandari, 2024). Some outliers represent true values from nature of variation in the population. True outliers should always be retained in your dataset because these just represent nature of variations in your sample. In addition, true outliers are also found in variables with skewed distributions where many data points are spread far from the mean in one direction. Other outliers that do not represent true values can be caused by many possible sources such as measurement errors, unrepresentative sampling, and data entry or processing errors (Bhandari, 2024). Graph visualizations like box plots detect outliers by calculating lower and upper bounds limits. Some Visualizations represent different curves or charts of the datasets. The structure of a datasets movement is represented very well by the chart graphs. Any chart shows a simple interpretation of data values which is suitable for Python-program language.

The unit-test framework in Python provides a comprehensive set of tools for creating and running tests. In addition to the test case sub-class you have used in this task, unit-test supports test suite, test fixture, and test runner sub-classes. As a result, it is a powerful tool for ensuring code quality and reliability (Python Software Foundation, 2024).

List of Appendices

Python provides several libraries and tools for data analysis.

1. **Built-in Statistics Module:**
   - Python's standard library includes the statistics module, which provides functions for calculating basic statistical measures such as mean, median, mode, variance, and standard deviation.
   - You can use this module to perform descriptive statistics on numeric data.
2. **NumPy:**
   - NumPy is a powerful library for numerical computing in Python.
   - It offers efficient array operations and mathematical functions, making it ideal for statistical calculations.
   - You can calculate mean, median, variance, and other statistics using NumPy arrays.
3. **SciPy:**
   - SciPy builds upon NumPy and provides additional statistical functions.
   - It includes modules like scipy.stats that offer various probability distributions, hypothesis tests, and statistical models.
4. **Pandas:**
   - Pandas is widely used for data manipulation and analysis.
   - It provides data structures like DataFrames and Series, which are useful for handling tabular data.
   - You can compute summary statistics, group data, and perform data exploration using Pandas.

The choice of python library depends on programmers' specific needs. Pandas is a great choice for tabular data. Furthermore, consider using Numpy or Scipy for some specialized statistical tasks. Please read through these libraries and the included Jupyter Notebook codes to maximize a basic knowledge of learning Python.
Save excel file to see boolean results of all duplicate rows or other functions using a logic of a code as follows:

```
df_duplicated = df_data.duplicated()
print(df_duplicated)
# determining the name of the file
df_duplicated_data_excel = 'df_duplicated_data_ideal.xlsx'
# saving to excel
df_duplicated.to_excel(df_duplicated_data_excel)
print('DataFrame is written to Excel File successfully.')
```

Appendices and materials

1. **D**
   Data:
   Information, especially facts or numbers, collected to be examined, considered, and used to help decision making, or information in an electronic form that can be stored and used by a computer.
2. **F**
   Function:
   A quantity whose value depends on another value and changes with that value.
3. **M**
   Mathematics:
   The science of numbers, forms, amounts, and their relationships.
   Mean:
   A number that is the result of adding a group of numbers together and then dividing the result by how many numbers were in the group.
   Median:
   The value that is the middle one in a set of values arranged in order of size.
   Mode:
   The number or value that appears most often in a particular set.
4. **O**
   Outcome:
   The result or effect of an action, situation, or event.
5. **P**
   Parameter:
   A set of facts which describes and puts limits on how something should happen or be done.
   Probability:
   A number that represents how likely it is that a particular thing will happen.
6. **Q**
   Quantity:
   The amount or number of something, especially that can be measured.
   Quartile:
   One of four equal parts that a set of figures is divided into, in order of size, amount, value, etc.
7. **S**
   Science:
   (Knowledge from) the careful study of the structure and behavior of the physical world, especially by watching, measuring, and doing experiments, and the development of theories to describe the results of these activities.
   Software:
   The instructions that control what a computer can do; computer programs.
   Standard deviation:
   Is a measure that is used to quantify the amount of variation or dispersion of a set of data values.
   Statistics:
   The science of using information discovered from collecting, organizing, and studying numbers.
8. **V**
   Variable:
   A number, amount, or situation that can change.
   Variance:
   The fact that two or more things are different, or the amount or number by which they are different.

Data-Wrangler python-program for chapter 1.

Firstly, install all of the required packages to execute the code successfully.

```
pip install config

pip install data-wrangler

pip install Pandas-Data-Exploration-Utility-Package

pip list
```

Restart Kernel to activate the list of installed packages.

Read through the program and comments. Edit the view and layout of the code to execute without errors.

```
'''
Imports all the modules needed by the class.
'''
import multiprocessing as mp
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import sys
#import config.config as cf
import config as cf
class DataWrangler(object):
#class DataWrangler:
    """
    A simple DataWrangler class.
    Attributes:
        file_name (str): The name of the file bearing our dataset to be wrang
led.
    Methods:
        load_data(): Loads the dataset from the file into a pandas dataframe.
        shape_of_data(df_data): Returns the shape of the dataframe passed to
it.
        summary_statistics(column_data): Returns a summary statistics of pand
as dataframe
        column passed to it.
        find_missing_values(df_data): Returns a list of missing values (nan)
by columns.
        duplicated_rows(df_data): Returns duplicated rows
        drop_duplicates(df_data): Returns a pandas dataframe with duplicated
rows dropped.
        fill_missing_values(df_data): Fills missing values (nan) in the dataf
rame with the mean
        value of each column
        find_outliers(df_column): Returns all values that are outliers in the
```

```
numeric
        column passed to it.
        is_outlier(value): Returns the value if it is an outlier
        fill_outliers_with_mean(outliers, df_column): fills all entries in th
e dataframe
        column passed to it, that are outliers with the mean value of the col
umn.
        sort_data(df_data): Sorts the dataframe by the 'x' column

        Example:
            data_wrangler = DataWrangler(file_name="train.csv")
            df_data = data_wrangler.load_data()
            print(df_data.head(5))  # Output: The first five rows of the trai
n dataset
    """
    def __init__(self, file_name):
        '''
        Constructor for the DataWrangler Class
        file_name: The Name of the file from which we get our raw data for wr
angling
        '''
        self.file_name = file_name
        self.lower_bound_outlier = 0
        self.upper_bound_outlier = 0
    def load_data(self):
        '''
        Loads the file received by constructor to pandas dataframe
        return: A pandas dataframe bearing the content of our file passed to
constructor
        '''
        try:
            #df_data = pd.read_csv(self.file_name)
            df_data = pd.read_csv(filepath_or_buffer=cf.INPUT_FILE_PATH+self.
file_name,
                                sep=",", encoding="latin1")
        except:
            exception_type, exception_value, exception_traceback = sys.exc.in
fo()
            print("Exception Type: {}\n Exception Value:{}".format(
                exception_type, exception_value))
            file_name, line_number, procedure_name, line_code = traceback.ext
ract_tb(
                exception_traceback)[-1]
            print("File Name: {}\n Line Number:{} \n Procedure Name: {} \n"+
                "Line Code: {}".format(file_name, line_number, procedure_name
,
                                                    line_code))
        finally:
            pass
        return df_data
    def shape_of_data(self, df_data):
        '''
        Computes the shape of the dataframe passed to it
        return: A tuple, which is the shape of the dataframe.
        '''
        df_shape = df_data.shape
        return df_shape
    def summary_statistics(self, column_data):
        '''
        Computes the summary statistic of each column of the dataframe
        return: A dataframe, which gives a summary statistic of each
        column in the dataframe.
        '''
        return column_data.describe()
    def find_missing_values(self, df_data):
        '''
        Computes the count of missing values in each column of the dataframe
        return: Returns a pandas series bearing the details on the count of m
issing
        values in each column of the dataframe
        '''
        df_result = df_data.apply(lambda x: sum(x.isnull()), axis=0)
        return df_result
    def duplicated_rows(df_data):
        '''
        Finds all duplicated rows in the dataframe
        return: Returns a dataframe bearing duplicated rows in the dataframe
        '''
        df_duplicated = df_data.duplicated()
        return df_duplicated
    def drop_duplicated(self, df_data):
        '''
        Drops all duplicated rows in the dataframe and keeps the first
        occurrence
        return: Returns a dataframe bearing no duplicated rows
        '''
        df_data = df_data.drop_duplicates(keep="first")
        return df_data
    def fill_missing_values(self, df_data):
        '''
        Fills all missing values in each column with the mean value of the
        column
        return: Returns a dataframe bearing no missing value
        '''
        df_data = df_data.fillna(df_data['x':'y4'].mean())
        return df_data
    def find_outliers(self, df_column):
        '''
        Finds all outlier values in the dataframe column passed to it
        return: Returns a list bearing all outlier values in the column
        '''
        q3 = np.percentile(df_column, 75)
        q1 = np.percentile(df_column, 25)
        iqr = q3-q1
        # Computes the outlier upper and lower bound values
        self.lower_bound_outlier = q1 - (1.5 * iqr)
        self.upper_bound_outlier = q3 + (1.5 * iqr)
        pool_obj = mp.Pool()
        outliers = pool_obj.map(is_outlier, df_column)
        return outliers
    def is_outlier(self, value):
        '''
        Determines if a value is an outlier based on our
        dataset column in consideration
        return: The Numeric Value passed to it is less than the computed
        lower bound outlier determinant or greater than the upper bound
        outlier determinant
        '''
        if value < self.lower_bound_outlier:
            return value
        if value > self.upper_bound_outlier:
            return value
    def fill_outliers_with_mean(self, outliers, df_column):
        '''
        Replaces all outlier values in the dataframe column passed to it with
        the mean of the values in the column
        return: The Dataframe column with outliers replaced by the mean value
        '''
        mean = df_column.mean()
        new_value = {}
        for value in outliers:
            new_value.update({value:mean})
            df_column.replace(to_replace=new_value, inplace=True)
        return df_column
    def sort_data(self, df_data):
        '''
        Sorts the dataframe passed to it by the column x
        return: The sorted dataframe
        '''
        sorted_df_data = df_data.sort_values(by='x')
        return sorted_df_data
print("Data Wrangler program executed without errors.")

import pandas_exploration_util.viz.explore as pe
#from DW import DataWrangler
#from data_exploration.data_wrangler import DataWrangler
if __name__ == "__main__":
    #unittest.main()
    # Please change file name and uncomment some part of the program to execu
te ideal, train and test files, respectively.
```

```
    print("load_data(): Loads the dataset from the file into a pandas datafra
me. Remember to change file name.")
    file_name = "test.csv"
    dataWrangler = DataWrangler(file_name)
    df_data = pd.read_csv("test.csv")

    print(df_data)
    print("")
    print("The compressed view of dataset file to do a quick check of your Da
taFrame.")
    display(df_data)
    print("")

    print("Return: A tuple, which is the shape of the dataframe.")
    df_shape= dataWrangler.shape_of_data(df_data)
    print(df_shape)

    print("")
    print("Computes the summary statistic of each column of the dataframe.")
    print("Return: A dataframe, which gives a summary statistic of each colum
n in the dataframe.")
    column_data = dataWrangler.summary_statistics(df_data)
    print(column_data)
    print("")
    display(column_data)

    print("")
    print("Computes the count of missing values in each column of the datafra
me.")
    print("Return: Returns a pandas series bearing the details on the count o
f missing values in each column of the dataframe.")
    df_result= dataWrangler.find_missing_values(df_data)
    print(df_result)
    print("")
    display(df_result)

    print("")
    print("Finds all duplicated rows in the dataframe.")
    print("Return: Returns a dataframe bearing duplicated rows in the datafra
me.")
    print("")
    #df_duplicated= dataWrangler.duplicated_rows()
    df_duplicated = df_data.duplicated()
    print(df_duplicated)
    print("")
    display(df_result)


    print("")
    print("Drops all duplicated rows in the dataframe and keeps the first occ
urrence.")
    print("Return: Returns a dataframe bearing no duplicated rows")
    print("")
    #df_data = dataWrangler.drop_duplicated()
    df_data = df_data.drop_duplicates(keep="first")
    print(df_data)

#     print("")
#     print("Returns a dataframe bearing no missing value.")
#     #df_data= dataWrangler.fill_missing_values(df_data)
#     #df_data = df_data.fillna(df_data['x':'y4'].mean())
#     print(df_data)

    #Sort values in each column
    sorted_df_data = df_data.apply(lambda s: s.sort_values().values)
    print("\nDataFrame with sorted values in each column:\n")
    print(sorted_df_data)

    print( "\nMedian.\n")
    q2 = df_data.quantile([0.5])
    print(q2)
    print("")
    display(q2)
    # determining the name of the file
    dfq2_excel = 'Q2_Train.xlsx'
    # saving to excel
    q2.to_excel(dfq2_excel)
    print('DataFrame is written to Excel File successfully.')

    print( "\nFirst Quartile.\n")
    q1 = df_data.quantile([0.25])
    print(q1)
    print("")
    display(q1)
    # determining the name of the file
    dfq1_excel = 'Q1_Train.xlsx'
    # saving to excel
    q1.to_excel(dfq1_excel)
    print('DataFrame is written to Excel File successfully.')
    #dfq1 = pd.DataFrame({'Variables': ['x', 'y1', 'y2', 'y3', 'y4'],
                         #'Values': [-10.025, -20.312566, -19.64256, -9.99980
7, -0.19824]})
    #print(dfq1)


    print("\nThird Quartile.\n")
    q3 = df_data.quantile([0.75])
    print(q3)
    print("")
    display(q3)
    # determining the name of the file
    dfq3_excel = 'Q3_Train.xlsx'
    # saving to excel
    q3.to_excel(dfq3_excel)
    print('DataFrame is written to Excel File successfully.')
    #dfq3 = pd.DataFrame({'Variables': ['x', 'y1', 'y2', 'y3', 'y4'],
                         #'Values': [9.925, 19.606209, 19.478971, 9.992209, 0
.236008]})
    #print(dfq3)

    # Convert Q1 to a matrix.
    print("")
    print("Convert Q1 to a matrix.")
    print("")
    matrix_q1 = q1.values
    print(matrix_q1)

    # Convert Q3 to a matrix.
    print("")
    print("Convert Q3 to a matrix.")
    print("")
    matrix_q3 = q3.values
    print(matrix_q3)

    # Calculate IQR = Q3 - Q1
    print("IQR")
    print("")
    IQR = matrix_q3 - matrix_q1
    print(IQR)

    # Convert the IQR matrix to a Pandas DataFrame.
    print("")
#     print("Convert the IQR matrix to an Ideal Pandas DataFrame.")
#     df_IQR_Ideal = pd.DataFrame(IQR, columns=['x', 'y1', 'y2', 'y3', 'y4',
'y5', 'y6', 'y7', 'y8', 'y9', 'y10',
#                                           'y11', 'y12', 'y13', 'y14', 'y15',
'y16', 'y17', 'y18', 'y19', 'y20',
#                                          'y21', 'y22', 'y23', 'y24', 'y25', '
y26', 'y27', 'y28', 'y29', 'y30',
#                                          'y31', 'y32', 'y33', 'y34', 'y35', '
y36', 'y37', 'y38', 'y39', 'y40',
#                                          'y41', 'y42', 'y43', 'y44', 'y45', '
y46', 'y47', 'y48', 'y49', 'y50'])

#     print("Convert the IQR matrix to a Test Pandas DataFrame.")
#     df_IQR_Test = pd.DataFrame(IQR, columns=['x', 'y'])
#     print("Convert the IQR matrix to a Train Pandas DataFrame.")
#     df_IQR_Train = pd.DataFrame(IQR, columns=['x', 'y1', 'y2', 'y3', 'y4'])
#     # Print the DataFrame
#     print("")
#     print(df_IQR_Train)
#     # determining the name of the file
#     df_IQR_excel = 'IQR_Train.xlsx'
#     # saving to excel
#     df_IQR_Train.to_excel(df_IQR_excel)
#     print('DataFrame is written to Excel File successfully.')

    print("")
    print("1.5 * IQR")
    print("")
    df_cIQR = 1.5 * IQR
    print(df_cIQR)
    print("")
#     print("Convert the 1.5*IQR matrix to an Ideal Pandas DataFrame.")
#     df_cIQR_Ideal = pd.DataFrame(df_cIQR, columns=['x', 'y1', 'y2', 'y3', '
y4', 'y5', 'y6', 'y7', 'y8', 'y9', 'y10',
#                                           'y11', 'y12', 'y13', 'y14', 'y15',
'y16', 'y17', 'y18', 'y19', 'y20',
#                                          'y21', 'y22', 'y23', 'y24', 'y25', '
y26', 'y27', 'y28', 'y29', 'y30',
#                                          'y31', 'y32', 'y33', 'y34', 'y35', '
y36', 'y37', 'y38', 'y39', 'y40',
#                                          'y41', 'y42', 'y43', 'y44', 'y45', '
y46', 'y47', 'y48', 'y49', 'y50'])

#     print("Convert the 1.5*IQR matrix to a Test Pandas DataFrame.")
#     df_cIQR_Test = pd.DataFrame(df_cIQR, columns=['x', 'y'])
#     print("Convert the 1.5*IQR matrix to a Train Pandas DataFrame.")
#     df_cIQR_Train = pd.DataFrame(df_cIQR, columns=['x', 'y1', 'y2', 'y3', '
y4'])

    # Print the DataFrame.
#     print("")
#     print(df_cIQR_Train)
#     # determining the name of the file
#     df_cIQR_excel = 'cIQR_Train.xlsx'
#     # saving to excel
#     df_cIQR_Train.to_excel(df_cIQR_excel)
#     print('DataFrame is written to Excel File successfully.')

    #Calculate your lower bound.
    print("")
    print("Lower bound")
    print("")
    df_IQR_lower_bound = matrix_q1 - df_cIQR
    print(df_IQR_lower_bound)
    print("")
#     print("Convert the lower bound matrix to an Ideal Pandas DataFrame.")
#     df_IQR_lower_bound_Ideal = pd.DataFrame(df_IQR_lower_bound, columns=['x
', 'y1', 'y2', 'y3', 'y4', 'y5', 'y6', 'y7', 'y8', 'y9', 'y10',
#                                           'y11', 'y12', 'y13', 'y14', 'y15',
'y16', 'y17', 'y18', 'y19', 'y20',
#                                          'y21', 'y22', 'y23', 'y24', 'y25', '
y26', 'y27', 'y28', 'y29', 'y30',
#                                          'y31', 'y32', 'y33', 'y34', 'y35', '
y36', 'y37', 'y38', 'y39', 'y40',
#                                          'y41', 'y42', 'y43', 'y44', 'y45', '
y46', 'y47', 'y48', 'y49', 'y50'])

#     print("Convert the lower bound matrix to a Test Pandas DataFrame.")
#     df_IQR_lower_bound_Test = pd.DataFrame(df_IQR_lower_bound, columns=['x'
, 'y'])
#     print("Convert the lower bound matrix to a Train Pandas DataFrame.")
#     df_IQR_lower_bound_Train = pd.DataFrame(df_IQR_lower_bound, columns=['x
', 'y1', 'y2', 'y3', 'y4'])

#     # Print the DataFrame.
#     print("")
#     print(df_IQR_lower_bound_Train)
#     # determining the name of the file
#     df_IQR_lower_bound_excel = 'IQR_lower_bound_Train.xlsx'
#     # saving to excel
#     df_IQR_lower_bound_Train.to_excel(df_IQR_lower_bound_excel)
#     print('DataFrame is written to Excel File successfully.')



    #Calculate your upper bound.
    print("")
    print("Upper bound")
    print("")
    df_IQR_upper_bound = matrix_q3 + df_cIQR
    print(df_IQR_upper_bound)
    print("")
#     print("Convert the upper bound matrix to an Ideal Pandas DataFrame.")
#     df_IQR_upper_bound_Ideal = pd.DataFrame(df_IQR_upper_bound, columns=['x
', 'y1', 'y2', 'y3', 'y4', 'y5', 'y6', 'y7', 'y8', 'y9', 'y10',
#                                           'y11', 'y12', 'y13', 'y14', 'y15',
'y16', 'y17', 'y18', 'y19', 'y20',
#                                          'y21', 'y22', 'y23', 'y24', 'y25', '
y26', 'y27', 'y28', 'y29', 'y30',
#                                          'y31', 'y32', 'y33', 'y34', 'y35', '
y36', 'y37', 'y38', 'y39', 'y40',
#                                          'y41', 'y42', 'y43', 'y44', 'y45', '
y46', 'y47', 'y48', 'y49', 'y50'])
```

```
#     print("Convert the upper bound matrix to a Test Pandas DataFrame.")
#     df_IQR_upper_bound_Test = pd.DataFrame(df_IQR_upper_bound, columns=['x'
, 'y'])
#     print("Convert the upper bound matrix to a Train Pandas DataFrame.")
#     df_IQR_upper_bound_Train = pd.DataFrame(df_IQR_upper_bound, columns=['x
', 'y1', 'y2', 'y3', 'y4'])

#     # Print the DataFrame.
#     print("")
#     print(df_IQR_upper_bound_Train)
#     # determining the name of the file
#     df_IQR_upper_bound_excel = 'IQR_upper_bound_Train.xlsx'
#     # saving to excel
#     df_IQR_upper_bound_Train.to_excel(df_IQR_upper_bound_excel)
#     print('DataFrame is written to Excel File successfully.')


    # Remember to change a file_name.
    df_data.plot()
    plt.title('Line plot of Train DataFrame')
    plt.xlabel('Index')
    plt.ylabel('Train values')
    plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
    plt.show()


    # Remember to change columns of the box plot for different file_names.
    fig, at = plt.subplots()
    # Ideal set up of y1-y10.
#     at.boxplot = df_data.boxplot(column =['x', 'y1', 'y2', 'y3', 'y4', 'y5'
, 'y6', 'y7', 'y8', 'y9', 'y10'])
    # Ideal set up of y11-y20.
#     at.boxplot = df_data.boxplot(column =['y11', 'y12', 'y13', 'y14', 'y15'
, 'y16', 'y17', 'y18', 'y19', 'y20'])
    # Ideal set up of y21-y30.
#     at.boxplot = df_data.boxplot(column =['y21', 'y22', 'y23', 'y24', 'y25'
, 'y26', 'y27', 'y28', 'y29', 'y30'])
#     # Ideal set up of y31-y40.
#     at.boxplot = df_data.boxplot(column =['y31', 'y32', 'y33', 'y34', 'y35'
, 'y36', 'y37', 'y38', 'y39', 'y40'])
#     # Ideal set up of y41-y50.
#     at.boxplot = df_data.boxplot(column =['y41', 'y42', 'y43', 'y44', 'y45'
, 'y46', 'y47', 'y48', 'y49', 'y50'])
    # Train set up.
#     at.boxplot = df_data.boxplot(column =['x', 'y1', 'y2', 'y3', 'y4'])
    # Test set up.
    at.boxplot = df_data.boxplot(column =['x', 'y'])
    at.set_title('Box plot of columns in Test dataframe without the outliers.
')
    at.set_xlabel('Test columns')
    at.set_ylabel('Test values')
    plt.show

    # Remember to change columns of the box plot for different file_names.
#     fig, aty = plt.subplots()
#     aty.boxplot = df_data.boxplot(column =['y43', 'y45', 'y46', 'y47'])
#     aty.set_title('Box plot of some columns in Ideal dataframe with outlier
s.')
#     aty.set_xlabel('Ideal columns')
#     aty.set_ylabel('Ideal values')
#     plt.show
```

```
"""
This is a Python-program that uses training data to choose the
four ideal functions which are the best fit out of the fifty provided (C) *.

    i)  Afterwards, the program uses the test data provided (B) to determine
        for each and every x-y-pair of values whether or not they can be
assigned to the
        four chosen ideal functions**; if so, the programs to executes the
mapping
        and saves it together with the deviation at hand.

    ii) All data is visualized logically

    iii) Where possible, suitable unit-test were compiled

"""

import sys      # Standard library imports
import pandas as pd    # related third party imports
import numpy as np
from matplotlib import pyplot as plt
from sqlalchemy import create_engine


class FindFunctions:
    def __init__(self):
        pass

    def find_ideal_matches(self, train_fun, ideal_fun):

        """

        function finds matches between training functions and ideal functions
based on min(MSE)
        :param train_fun: define training functions
        :param ideal_fun: define ideal functions set
        :return: ideal functions dataframe and their deviations
        """

        # find last parameters of both fucntions
        if isinstance(train_fun, pd.DataFrame) and isinstance(ideal_fun,
pd.DataFrame):
            ideal_lcol = len(ideal_fun.columns)
            train_lrow = train_fun.index[-1] + 1
            train_col = len(train_fun.columns)

            # Loop and find perfect four functions
            index_list = []  # here 4 ideal indexes will be strored
            least_square = []  # here 4 ideal MSEs will be stored
            for j in range(1, train_col):  # Loop through 4 train functions
                least_square1 = []
                for k in range(1, ideal_lcol):  # Loop through 50 ideal 
functions
                    MSE_sum = 0  # Sum MSE

                    for i in range(train_lrow):  # calculate MSE Y value of 
train and Y value of ideal function
                        z1 = train_fun.iloc[i, j]  # Train y value
                        z2 = ideal_fun.iloc[i, k]  # Ideal y value
                        MSE_sum = MSE_sum + ((z1 - z2) ** 2)

                    least_square1.append(MSE_sum / train_lrow)

                min_least = min(least_square1)
                index = least_square1.index(min_least)  # find index of the 
ideal function
                index_list.append(index + 1)
                least_square.append(min_least)

            per_frame = pd.DataFrame(list(zip(index_list, least_square)), 
columns=["Index", "least_square_value"])

            return per_frame

        else:
            raise TypeError("Given arguments are not of Dataframe type.")

    def find_ideal_via_row(self, test_fun):

        """
        determine for each and every x-y-pair of values whether they can be 
assigned to the four chosen ideal functions
        :param test_fun: Dataframe with x and y values
        :return: test function paired with values from the four ideal 
functions
        """

        if isinstance(test_fun, pd.DataFrame):
            test_lrow = test_fun.index[-1] + 1  # Last row of the test df 
(used for loop)
            test_lcol = len(test_fun.columns)  # Last columns of the test df 
(used for loop)
            # print(test)

            ideal_index = []  # List to store index of ideal function
            deviation = []  # List to store Deviation
            for j in range(test_lrow):  # Loop through rows
```

```
                MSE_l = []  # list to store all four deviations

                for i in range(2, test_lcol):  # loop through columns 2, 3,
4, 5
                    z1 = test_fun.iloc[j, 1]
                    z2 = test_fun.iloc[j, i]
                    MSE_sum = ((z2 - z1) ** 2)  # calculate MSE
                    MSE_l.append(MSE_sum)  # append MSE to the MSE_l list

                min_least = min(MSE_l)  # select min deviation in MSE_l

                if min_least < (np.sqrt(2)):
                    deviation.append(min_least)  # append min_least to the
deviation list
                    index = MSE_l.index(min_least)  # select index of the
min_least to find ideal function
                    ideal_index.append(index)  # append index to the
ideal_index list

                else:

                    deviation.append(min_least)
                    ideal_index.append("Miss")  # no criteria match


            # Add two new columns to the test
            test["Deviation"] = deviation
            test["Ideal index"] = ideal_index

            return test

        else:
            raise TypeError("Given argument is not of Dataframe type.")

    def prepare_graphs(self, x_fun, x_par, y1_fun, y1_par, y2_fun, y2_par,
show_plots=True):

        """
        function prepares a plot based on given paramaters
        :param x_fun: x function
        :param x_par: x position
        :param y1_fun: y1 function
        :param y1_par: y1 position
        :param y2_fun: y2 function
        :param y2_par: y2 position
        :param show_plots: True/False to display plot
        :return: graph of x and y
        """
```

```
            x = x_fun.iloc[:, x_par]    # x
            y1 = y1_fun.iloc[:, y1_par]     # y1 (training function)
            y2 = y2_fun.iloc[:, y2_par]     # y2 (ideal function)

            # print(y1, y2)

            plt.plot(x, y1, c="r", label="Train function")  # plot both axis
            plt.plot(x, y2, c="b", label="Ideal function")
            plt.xlabel("x")
            plt.ylabel("y")
            plt.legend(loc=3)

            if show_plots is True:
                plt.show()  # show current plot
                plt.clf()   # clear plots
            elif show_plots is False:
                pass
            else:
                pass  # no paramater show_plots or wrong paramater show_plots was
given


print("The first part of the program executed without errors.")

class SqliteDb(FindFunctions):

    """
    Load data into Sqlite database
    """

    def db_and_table_creation(self, dataframe, db_name, table_name):

        """
        function creates a database from a dataframe input
        :param dataframe: dataframe
        :param db_name: database name
        :param table_name: table name
        :return: database file into the same folder as the project
        """
        try:
            engine = create_engine(f"sqlite:///{IU_Assignment}.db",
echo=True)  # insert name of the DB
            sqlite_connection = engine.connect()  # connect to the DB

            for i in range(len(dataframes)):  # Loop through list of
dataframes
                dataframez = dataframe[i]
                dataframez.to_sql(table_name[i], sqlite_connection,
if_exists="fail")  # load dataframe to DB
            sqlite_connection.close()   # close connection

        except Exception:
            exception_type, exception_value, exception_traceback =
sys.exc_info()  # get exception info
            print(exception_type, exception_value, exception_traceback)
#return exception info to the user

print("The database created successfully.")

# read CSV files and load them into Dataframes
train = pd.read_csv("train.csv")
ideal = pd.read_csv("ideal.csv")
test = pd.read_csv("test.csv")

print("Read CSV files and load them into Dataframes.")
print("")

# Check data formats
print("Train dataset")
print(train)
print("")

print("Ideal dataset")
print(ideal)
print("")

print("Test dataset")
print(test)
print("")



# get ideal functions based on train data
print("Get ideal functions based on train data.")
df = FindFunctions().find_ideal_matches(train, ideal)
print(df)

# plot graph of all 4 pair functions together
print("")
print("The selected four ideal functions 11, 41, 42, and 48 are the best fit
of training data ")
print("Plot graph of all 4 pair functions together.")

graph = FindFunctions()
for i in range(1, len(train.columns)):
    graph.prepare_graphs(train, 0, train, i, ideal, df.iloc[i-1, 0], False)
```

```
# Clean test df
test = test.sort_values(by=["x"], ascending=True)   # sort by x
test = test.reset_index()   # reset index
test = test.drop(columns=["index"])     # drop old index column

print("")
print("Clean test df")

# Get x, y values of each of the 4 ideal functions
print("")
print("Get x, y values of each of the 4 ideal functions.")
ideals = []
for i in range(0, 4):
    ideals.append(ideal[["x", f"y{str(df.iloc[i, 0])}"]])

# merge test and 4 ideal functions
print("")
print("Merge test and 4 ideal functions.")
for i in ideals:
    test = test.merge(i, on="x", how="left")

# determine for each and every x-y-pair of values whether or not they can be
assigned to the four chosen ideal functions
print("")
print("Determine for each and every x-y-pair of values whether or not they
can be assigned to the four chosen ideal functions.")
test = FindFunctions().find_ideal_via_row(test)
print(test)

# Replace values with ideal function names
print("")
print("Replace values with ideal function names.")
for i in range(0, 4):
    test["Ideal index"] = test["Ideal index"].replace([i], str(f"y{df.iloc[i,
0]}"))

# add y values to another test_fun (used later for scatter plot)
print("")
print("Add y values to another test_fun (used later for scatter plot).")
test_scat = test
test_scat["ideal y value"] = ""
for i in range(0, 100):
    k = test_scat.iloc[i, 7]
    if k == "y42":
        test_scat.iloc[i, 8] = test_scat.iloc[i, 2]
    elif k == "y41":
        test_scat.iloc[i, 8] = test_scat.iloc[i, 3]
    elif k == "y11":
        test_scat.iloc[i, 8] = test_scat.iloc[i, 4]
    elif k == "y48":
        test_scat.iloc[i, 8] = test_scat.iloc[i, 5]
    elif k == "Miss":
        test_scat.iloc[i, 8] = test_scat.iloc[i, 1]
print(test_scat)

# Drop other columns that are not used
print("")
print("Drop other columns that are not used.")
test = test.drop(columns=["y42", "y41", "y11", "y48", "ideal y value"])
print(test)

# rename columns for the train table
print("")
print("Rename columns for the train table.")
train = train.rename(columns={"y1": "Y1 (training func)", "y2": "Y2 (training
func)",
                              "y3": "Y3 (training func)", "y4": "Y4 (training
func)"})
print(train)

# rename columns for the ideal table
print("")
print("Rename columns for the ideal table.")
for col in ideal.columns:      # rename columns in ideal to fit criteria
    if len(col) > 1:    # if column name is not x, therefore > 1
        ideal = ideal.rename(columns={col: f"{col} (ideal func)"})

print(ideal)
ideal_excel = 'Rename_Ideal_Columns.xlsx'
# saving to excel
ideal.to_excel(ideal_excel)
print('DataFrame is written to Excel File successfully.')

# rename columns for the test table
print("")
print("Rename columns for the test table.")
test = test.rename(columns={"x": "X (test func)",
                            "y": "Y (test func)",
                            "Deviation": "Delta Y (test func)",
                            "Ideal index": "No. of ideal func"})
print(test)

# Load data to sqlite
#assignment_database
print("")
print("Load data to sqlite.")
dbs = SqliteDb()
```

```
dataframes = [train, ideal, test]
table_names = ["train_table", "ideal_table", "test_table"]
dbs.db_and_table_creation(dataframes, "assignment_database", table_names)

# Visualization
# train functions
print("")
print("Visualization.")
print("Train functions.")
plt.clf()
x = train.iloc[:, 0]
for i in range(1, len(train.columns)):
    plt.plot(x, train.iloc[:, i], c="g", label=f"Train function y{i}")
    plt.title(f"Train function y{i}.")
    plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
    plt.xlabel("x")
    plt.ylabel(f"y{i}")
    plt.show()
    plt.clf()

# ideal functions (4 chosen)
print("")
print("Ideal functions (4 chosen).")
plt.clf()
x = train.iloc[:, 0]
for i in range(0, df.index[-1] + 1):
    y = df.iloc[i, 0]  # get ideal y column number (42, 41, 11, 48)
    plt.plot(x, ideal.iloc[:, y], c="#FF4500", label=f"Ideal function y{y}")
    plt.title(f"The Chosen Ideal Function y{y}.")
    plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
    plt.xlabel("x")
    plt.ylabel(f"y{y}")
    plt.show()
    plt.clf()

# ideal functions (all 50)
print("")
print("Ideal functions (all 50).")
plt.clf()
x = ideal.iloc[:, 0]
for i in range(1, len(ideal.columns)):
    plt.plot(x, ideal.iloc[:, i], c="#FF4500", label=f"Ideal function y{i}")
    plt.title(f"Ideal Function y{i}.")
    plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
    plt.xlabel("x")
    plt.ylabel(f"y{i}")
    plt.show()
    plt.clf()
```

```
# test scatter (show points of test.csv)
print("")
print("Test scatter (show points of test.csv).")
plt.clf()  # clear previous plots
plt.scatter(test.iloc[:, 0], test.iloc[:, 1])  # select x and y values
plt.title("Test scatter (show points of test.csv).")
plt.xlabel("x")
plt.ylabel("y")
plt.show()

plt.clf()  # clear previous plots
# create lists to visualize test_scat dataframe
print("")
print("Create lists to visualize test_scat dataframe.")
x1 = []
x2 = []
x3 = []
x4 = []
xm = []
y1 = []
y2 = []
y3 = []
y4 = []
ym = []

# append x and y values to the upper lists
print("")
print("Append x and y values to the upper lists.")
for i in range(0, 100):
    k = test_scat.iloc[i, 7]
    if k == "y42":
        x1.append(test_scat.iloc[i, 0])  # append x value of y42 to the x1
list
        y1.append(test_scat.iloc[i, 8])  # append y value of y42 to the y1
list
    elif k == "y41":
        x2.append(test_scat.iloc[i, 0])  # append x value of y41 to the x2
list
        y2.append(test_scat.iloc[i, 8])  # append y value of y41 to the y2
list
    elif k == "y11":
        x3.append(test_scat.iloc[i, 0])  # append x value of y11 to the x3
list
        y3.append(test_scat.iloc[i, 8])  # append y value of y11 to the y3
list
    elif k == "y48":
        x4.append(test_scat.iloc[i, 0])  # append x value of y48 to the x4
```

```
list
        y4.append(test_scat.iloc[i, 8])  # append y value of y48 to the y4
list
    elif k == "Miss":
        xm.append(test_scat.iloc[i, 0])  # append x value of "Miss" values to
the xm list
        ym.append(test_scat.iloc[i, 8])  # append y value of "Miss" values to
the ym list

# plot ideal functions and test y-values on the same scatter plot
print("")
print("Plot ideal functions and test y-values on the same scatter plot.")
plt.scatter(x1, y1, marker="o", label="Test - y42", color="r")
plt.scatter(x2, y2, marker="s", label="Test - y41", color="b")
plt.scatter(x3, y3, marker="^", label="Test - y11", color="g")
plt.scatter(x4, y4, marker="d", label="Test - y48", color="#FFD700")
plt.scatter(xm, ym, marker="x", label="Test - Miss", color="#000000")
plt.plot(ideal.iloc[:, 0], ideal.iloc[:, 42], label="Ideal - Y42",
color="#FA8072")
plt.plot(ideal.iloc[:, 0], ideal.iloc[:, 41], label="Ideal - Y41",
color="#1E90FF")
plt.plot(ideal.iloc[:, 0], ideal.iloc[:, 11], label="Ideal - Y11",
color="#7CFC00")
plt.plot(ideal.iloc[:, 0], ideal.iloc[:, 48], label="Ideal - Y48",
color="#FFA500")
plt.title("Scatter plot of ideal functions and test y-values.")
plt.xlabel("x")
plt.xlabel("y")
plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
plt.show()
```

Unit-test code of all 50 ideal functions.

```
import unittest
import pandas as pd

class TestDataWrangler(unittest.TestCase):
    '''
    Test the DataWrangler Class
    '''
    def test_load_data(self):
        '''
        four best fit ideal functions
        test the load_data method that it successfully
        constructed a dataframe from the .csv file
        '''
        #self.dataWrangler = DataWrangler("ideal.csv")
        self.dataset = pd.read_csv("ideal.csv")
        self.df_data = pd.DataFrame(self.dataset)
        self.assertNotEqual(isinstance(self.df_data, pd.DataFrame), True,
"The returned value is of type DataFrame")
        # self.assertEqual(isinstance(self.df_data, pd.DataFrame), True, "The
returned value is of type DataFrame")

    def test_shape_of_data(self):
        '''
        four best fit ideal functions
        test the shape_data method that it successfully
        returns the shape of the dataframe constructed
        '''
        self.dataset = pd.read_csv("ideal.csv")
        self.df_data = pd.DataFrame(self.dataset)
        df_shape = self.df_data.shape
        self.assertEqual(df_shape[0], 400, "The tuple contains at index 0,
the value 400,"+
                         " which is our number of rows")
        self.assertEqual(df_shape[1], 51, "The tuple contains at index 1, the
value 51,"+
                         " which is our number of columns")

    def test_summary_statistics(self):
        '''
        four best fit ideal functions
        test the summary_statistics method that it successfully
        returns a dataframe bearing the summary statistics of each dataframe
column
        '''
        self.dataset = pd.read_csv("ideal.csv")
        self.df_data = pd.DataFrame(self.dataset)
        self.df_summary = self.df_data.describe()
        self.assertEqual(self.df_summary.loc['mean','x'], -
0.04999999999999434, "The mean value of x column"+
                         " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y1'], -
0.002282363249999295, "The mean value of y1 column"+
                         " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y2'], 0.045609215055,
"The mean value of y2 column"+
                         " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y3'], 9.9977176375, "The
mean value of y3 column"+
                         " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y4'], 5.045609232, "The
mean value of y4 column"+
                         " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y5'], -9.9977176375,
"The mean value of y5 column"+
                         " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y6'],
0.002282363249999295, "The mean value of y6 column"+
                         " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y7'], -
0.054390777963499996, "The mean value of y7 column"+
                         " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y8'], 0.0307260006, "The
mean value of y8 column"+
                         " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y9'], 0.09121843113,
"The mean value of y9 column"+
                         " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y10'], -
0.36205663077499994, "The mean value of y10 column"+
                         " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y12'], 1.85, "The mean
value of y12 column"+
                         " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y13'], -
5.0999999999999999, "The mean value of y13 column"+
                         " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y14'],
0.04999999999999434, "The mean value of y14 column"+
                         " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y15'],
3.0249999999999995, "The mean value of y15 column"+
                         " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y16'], 133.335, "The
mean value of y16 column"+
                         " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y17'], -133.335, "The
mean value of y17 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y18'], 266.67, "The mean 
value of y18 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y19'], 143.335, "The 
mean value of y19 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y20'], 142.035, "The 
mean value of y20 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y21'], -20.0, "The mean 
value of y21 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y22'], 2000.05, "The 
mean value of y22 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y23'], 20.0, "The mean 
value of y23 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y24'], -40.0, "The mean 
value of y24 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y25'], -55.0, "The mean 
value of y25 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y26'], 787.41, "The mean 
value of y26 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y27'], 828.61, "The mean 
value of y27 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y28'], -
20.049999999999855, "The mean value of y28 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y29'], 
113.33499999999985, "The mean value of y29 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y30'], -281.67, "The 
mean value of y30 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y31'], 10.0, "The mean 
value of y31 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y32'], 2.9810999359828, 
"The mean value of y32 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y33'], 
10.423256193500002, "The mean value of y33 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y34'], 0.5070004845431, 
"The mean value of y34 column"+
                     " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y35'],
8.526512829121202e-16, "The mean value of y35 column"+
                     " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y36'], 50.0, "The mean
value of y36 column"+
                     " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y37'], -10.0, "The mean
value of y37 column"+
                     " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y38'], -
0.04789158018424998, "The mean value of y38 column"+
                     " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y39'], 133.33271742251,
"The mean value of y39 column"+
                     " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y40'], 234.289390960375,
"The mean value of y40 column"+
                     " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y43'], 1.933862925375,
"The mean value of y43 column"+
                     " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y44'], -
0.01013142119125, "The mean value of y44 column"+
                     " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y45'],
11.933862884999998, "The mean value of y45 column"+
                     " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y46'], 4.236448031, "The
mean value of y46 column"+
                     " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y47'], -4.236448031,
"The mean value of y47 column"+
                     " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y49'],
0.029571229945000704, "The mean value of y49 column"+
                     " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y50'],
0.040335506957499996, "The mean value of y50 column"+
                     " was truly computed")


        # The chosen four functions with best fit based on train dataset.
        self.assertEqual(self.df_summary.loc['mean','y42'],
0.12280453926250061, "The mean value of y42 column"+
                     " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y41'], -
0.10114118499999904, "The mean value of y41 column"+
                     " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y11'], -
0.049999999999999434, "The mean value of y11 column"+
                     " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y48'], -
0.0004656956999996426, "The mean value of y48 column"+
                     " was truly computed")


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

Unit-test code of the chosen best fit of four ideal functions based on train dataset.

```
import unittest
import pandas as pd

class TestDataWrangler(unittest.TestCase):
    '''
    Test the DataWrangler Class
    '''
    def test_load_data(self):
        '''
        four best fit ideal functions
        test the load_data method that it successfully
        constructed a dataframe from the .csv file
        '''
        #self.dataWrangler = DataWrangler("ideal.csv")
        self.dataset = pd.read_csv("ideal.csv")
        self.df_data = pd.DataFrame(self.dataset)
        self.assertNotEqual(isinstance(self.df_data, pd.DataFrame), True,
"The returned value is of type DataFrame")
        # self.assertEqual(isinstance(self.df_data, pd.DataFrame), True, "The
returned value is of type DataFrame")

    def test_shape_of_data(self):
        '''
        four best fit ideal functions
        test the shape_data method that it successfully
        returns the shape of the dataframe constructed
        '''
        self.dataset = pd.read_csv("ideal.csv")
        self.df_data = pd.DataFrame(self.dataset)
        df_shape = self.df_data.shape
        self.assertEqual(df_shape[0], 400, "The tuple contains at index 0,
the value 400,"+
                     " which is our number of rows")
        self.assertEqual(df_shape[1], 51, "The tuple contains at index 1, the
value 51,"+
                     " which is our number of columns")
    def test_summary_statistics(self):
        '''
        four best fit ideal functions
        test the summary_statistics method that it successfully
        returns a dataframe bearing the summary statistics of each dataframe
column
        '''
        self.dataset = pd.read_csv("ideal.csv")
        self.df_data = pd.DataFrame(self.dataset)
        self.df_summary = self.df_data.describe()
        self.assertEqual(self.df_summary.loc['mean','x'], -
0.04999999999999434, "The mean value of x column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y42'],
0.12280453926250061, "The mean value of y42 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y41'], -
0.10114118499999904, "The mean value of y41 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y11'], -
0.04999999999999434, "The mean value of y11 column"+
                          " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y48'], -
0.0004656956999996426, "The mean value of y48 column"+
                          " was truly computed")


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

Test dataset unit-test.

```
import unittest
import pandas as pd

class TestDataWrangler(unittest.TestCase):
    '''
    Test the DataWrangler Class
    '''
    def test_load_data(self):
        '''
        test the load_data method that it successfully
        constructed a dataframe from the .csv file
        '''
        #self.dataWrangler = DataWrangler("test.csv")
        self.dataset = pd.read_csv("test.csv")
        self.df_data = pd.DataFrame(self.dataset)
        self.assertNotEqual(isinstance(self.df_data, pd.DataFrame), True,
"The returned value is of type DataFrame")
        # self.assertEqual(isinstance(self.df_data, pd.DataFrame), True, "The
returned value is of type DataFrame")

    def test_shape_of_data(self):
        '''
        test the shape_data method that it successfully
        returns the shape of the dataframe constructed
        '''
        self.dataset = pd.read_csv("test.csv")
        self.df_data = pd.DataFrame(self.dataset)
        df_shape = self.df_data.shape
        self.assertEqual(df_shape[0], 100, "The tuple contains at index 0,
the value 100,"+
                         " which is our number of rows")
        self.assertEqual(df_shape[1], 2, "The tuple contains at index 1, the
value 2,"+
                         " which is our number of columns")

    def test_summary_statistics(self):
        '''
        test the summary_statistics method that it successfully
        returns a dataframe bearing the summary statistics of each dataframe
column
        '''
        self.dataset = pd.read_csv("test.csv")
        self.df_data = pd.DataFrame(self.dataset)
        self.df_summary = self.df_data.describe()
        self.assertEqual(self.df_summary.loc['mean','x'], 0.299000000000003,
"The mean value of x column"+
                         " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y'], 0.3254828044499999,
"The mean value of y column"+
                         " was truly computed")


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

Train dataset unit-test.

```
import unittest
import pandas as pd

class TestDataWrangler(unittest.TestCase):
    '''
    Test the DataWrangler Class
    '''
    def test_load_data(self):
        '''
        test the load_data method that it successfully
        constructed a dataframe from the .csv file
        '''
        #self.dataWrangler = DataWrangler("train.csv")
        self.dataset = pd.read_csv("train.csv")
        self.df_data = pd.DataFrame(self.dataset)
        self.assertNotEqual(isinstance(self.df_data, pd.DataFrame), True, 
"The returned value is of type DataFrame")
        # self.assertEqual(isinstance(self.df_data, pd.DataFrame), True, "The 
returned value is of type DataFrame")

    def test_shape_of_data(self):
        '''
        test the shape_data method that it successfully
        returns the shape of the dataframe constructed
        '''
        self.dataset = pd.read_csv("train.csv")
        self.df_data = pd.DataFrame(self.dataset)
        df_shape = self.df_data.shape
        self.assertEqual(df_shape[0], 400, "The tuple contains at index 0, 
the value 400,"+
                             "which is our number of rows")
        self.assertEqual(df_shape[1], 5, "The tuple contains at index 1, the 
value 5,"+" which is our number of columns")

    def test_summary_statistics(self):
        '''
        test the summary_statistics method that it successfully
        returns a dataframe bearing the summary statistics of each dataframe 
column
        '''
        self.dataset = pd.read_csv("train.csv")
        self.df_data = pd.DataFrame(self.dataset)
        self.df_summary = self.df_data.describe()
        self.assertEqual(self.df_summary.loc['mean','x'], -
0.049999999999999434, "The mean value of x column"+ " was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y1'], 
0.10766606164249878, "The mean value of y1 column"+" was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y2'], -
0.09423904243749917, "The mean value of y2 column"+" was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y3'], -
0.05162830643749942, "The mean value of y3 column"+" was truly computed")
        self.assertEqual(self.df_summary.loc['mean','y4'], 0.01263280748595, 
"The mean value of y4 column"+ " was truly computed")


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

Bibliography

Bhandari, P. (2024). How to Find Outliers — Meaning, Formula Examples. (URL: https://www.scribbr.co.uk/stats/statistical-outliers/ [last access: 1.11.2024]).

Chen, B. (2021). Finding and removing duplicate rows in Pandas DataFrame. (URL: https://towardsdatascience.com/finding-and-removing-duplicate-rows-in-pandas-dataframe-c6117668631f [last access: 12.11.2024]).

Geeks for Geeks (2023). Pandas df.size, df.shape and df.ndim Methods. (URL: https://www.geeksforgeeks.org/python-pandas-df-size-df-shape-and-df-ndim/ [last access: 12.11.2024]).

Geeks for Geeks (2024). Least Square Method — Definition Graph and Formula. (URL: https://www.geeksforgeeks.org/least-square-method/ [last access: 13.11.2024]).

Harvard Business School Online (2024). Data Wrangling: What It Is Why It's Important. (URL: https://online.hbs.edu/blog/post/data-wrangling [last access: 30.10.2024]).

Keita, Z. (2023). Top Techniques to Handle Missing Values Every Data Scientist Should Know. (URL: https://www.datacamp.com/tutorial/techniques-to-handle-missing-data-values [last access: 12.11.2024]).

Python Software Foundation (2021-2024). unittest — Unit testing framework. (URL: https://docs.python.org/3/library/unittest.html [last access: 8.11.2024]).

Richman, J. (2023). How to Load Data into Python: The Guide. (URL: https://estuary.dev/load-data-python/#:~:text=Loading%20data%20into%20Python [last access: 12.11.2024]).

Statistics How To (2024). Mean Squared Error: Definition and Example. (URL: https://www.statisticshowto.com/probability-and-statistics/statistics-definitions/mean-squared-error/ [last access: 13.11.2024]).

Story of Mathematics (2024). Summary Statistics – Explanation and Examples. (URL: https://www.storyofmathematics.com/summary-statistics/ [last access: 12.11.2024]).

Varsity Tutors (2024). Line of best fit. (URL: https://www.varsitytutors.com/hotmath/hotmath_help/topics/line-of-best-fit [last access: 13.11.2024]).

W3 schools (1999-2024). Git Tutorial. (URL: https://www.w3schools.com/git/default.asp?remote=github [last access: 15.11.2024]).