



DATA ANALYSIS USING PYTHON LIBRARIES IN THIS ERA VS WITHOUT LIBRARIES

Course Name: Programming With Python DLMDSPWP01

Course of Study: MSc Computer Science On-Campus

Type of Thesis: Research & Practical

Date: 9th June 2024

Author Name: Abdul Ghani Chishti

Author Email: abdul-ghani.chishti@iu-study.org

Matriculation Number: 4242853

Tutor Name: Cosmina Croitoru & Tai Le Quy

Table of Contents

1. INTRODUCTION	2
2.DATA ANALYSIS USING PYTHON WITHOUT LIBRARIES	3
2.1. Tabular data.	3
2.2. Manual Calculations	3
2.3. Graphical Representation.	4
2.4. Iterative Processes.	4
2.5. Sorting Algorithms.	5
2.6. Data Structures	5
3.PYTHON LIBRARIES FOR DATA ANALYSIS	5
3.1. OVERVIEW OF PYTHON LIBRARIES	5
3.1.1. Numpy:	5
3.1.2. Pandas:	5
3.1.3. SciPy:	6
3.1.4. Scikit-learn:	6
3.1.5. Matplotlib:	6
3.1.6. Seaborn:	6
3.2.DATA ANALYSIS USING PYTHON LIBRARIES	7
4. COMPARISON BETWEEN DATA ANALYSIS USING PYTHON LIBRARIES AND WITHOUT LIBRARIES	10
4.1. Detail Comparison	11
4.2. Future of Data Analysis using Libraries	12
5.CONCLUSION	13
6. Relation between Research Work & Practical	13
7.REFERNECES:	14
8. APPENDIX	15
8.1 About Code	15
8.2 Code	15
8.3 RESULT	20
8.4 VISUALIZATION	21

1. INTRODUCTION

In the era of data and where data is the key for everything, the importance of efficient data analysis cannot be overstated. Data analysis is easier now a days because of various libraries available in python programming language for doing all sorts of analysis as compared to the era when there were no such libraries available. Data Analysis using python before the libraries were introduced was such a hectic task as the size of the data increased the amount of effort it took to analyze, clean and perform statistical methods takes a lot of time.

The primary objective of the research is to evaluate how Python libraries changes the data analysis processes, improve the efficiency and its accuracy. The traditional methods used earlier before the python libraries evaluation and its process will be discussed. The different techniques and analysis procedures that researcher take to complete the data analysis and the research will also focus on the methodologies that were used using different python libraries such as (Numpy, Pandas, Matplotlib) and prior so that there will be clear distinction between the efficiency of the output of both libraries and without libraries.

The discussion will not delve into the structure of the libraries or types of algorithm used in the libraries but instead the focus will remain on the practical implementation and the advancement the libraries offers. Additionally, the old methods before libraries will also be discussed to give the gest of both of them equally. Furthermore, Finally the comparative analysis will highlight how the traditional methods and python libraries enhance the data analysis process to get the clear understanding of how python programming language libraries revolutionize the data analysis.

2.DATA ANALYSIS USING PYTHON WITHOUT LIBRARIES

Data analysis when modern libraries are not introduced were difficult and requires more time than it can be done now in seconds. When there were no libraries for data analysis each and every calculation should be performed manually in code. For each statistical or scientific computation a manual code is written, to start the data analysis all the numerical, statistical, scientific methods are created that will be used on the data. On the other hand, data is manually handled using basic fileing procedures that will read or write the file (if using csv or xls). This requires another piece of code that will handle the data that will accurately read the file and than on that data the analysis is performed. All this process has to be done in order to perform a simple data analysis but if the data is too large than it will takes a lot of time.

However, Traditional technique for data analysis was used when there were no such advance libraries available in python programming language. Every data scientist use this technique to start the data analysis. It is as follows:

2.1. Tabular data.

The Data can initially be in raw form or unstructured. For data analysis, the data is converted into tabular format so that it will be easily readable and can have clear understanding of how the data will be in structured format. Some data can already be in a structured format that will minimize the time of converting the data into structured form. The tabular data is often used for small data sets for clear data representation. Some data can be analyzed without tabular format as well.

2.2. Manual Calculations

To perform the necessary calculation in python when there were no libraries, all functions has to be made manually using arithmetic and logic expressions. The basic statistical functions also be made manually consuming extra time and making the process of data analysis slower. The basic statistical methods used in early stages of data analysis are mean, median, mode, standard deviation etc. These basic functions were made manually to perform statistical processes on data.

Using arithmetic and logical operators of python programming mathematical or statistical functions can be made. Here is the code snippet of some functions in python without using any advance libraries:

Fig.1 Calculating Mean of the data without using libraries

```
[3]: students_scores = [
    {'name': 'Ahmed', 'score': 88},
    {'name': 'Ali', 'score': 92},
    {'name': 'Omer', 'score': 85},
    {'name': 'Dawood', 'score': 91},
    {'name': 'Adam', 'score': 88},
]

def calculate_mean(scores):
    total = 0
    for student in scores:
        total = total + student['score']
    mean = total / len(scores)
    return mean
calculate_mean(students_scores)

[3]: 88.8
```

Source: own representation

Fig. 2 Calculating Standard Deviation

```
def calculate_standard_deviation(scores, mean):
    total_squared_diff = 0
    for student in scores:
        total_squared_diff += (student['score'] - mean) ** 2
    variance = total_squared_diff / len(scores)
    standard_deviation = variance ** 0.5
    return standard_deviation
calculate_standard_deviation(students_scores, mean)

2.4819347291981715
```

Source: Own representation

2.3. Graphical Representation.

Graphical representation of the data helps to understand the data more easily and effectively but when there was no advance python libraries for visualization the sketches were drawn manually or by using drawing tools in python.

2.4. Iterative Processes.

In conventional data analysis, iteration is a basic method. A fundamental technique is looping through data points to carry out calculations, update counts, or verify conditions. For tasks like figuring out modes, summing frequencies, and computing means, this iterative method is crucial for data analysis so that one procedure can be repeated if needed.

2.5. Sorting Algorithms.

For many data analysis tasks, such as determining medians or generating frequency distributions, sorting is an essential first step. Sorting algorithms such as bubble sort, insertion sort, or quicksort can be implemented with the help of simple Python structures. This enables ordered data processing, which is essential for many statistical computations.

2.6. Data Structures

Traditional data analysis requires the use of fundamental data structures like lists, dictionaries, and tuples. Lists are used for managing immutable data, dictionaries for keeping frequency counts, and lists for storing datasets. Gaining proficiency with these structures makes it more effective and efficient data analysis and manipulation.

3.PYTHON LIBRARIES FOR DATA ANALYSIS

Data analysis is becoming more crucial in many industries that work on large datasets or the industries whose work is data driven. Python program language with its extensive libraries has made data analysis easier and efficient. From at least two decades, Python language has gone in advance scientific computing (*McKinney, W. (2012), p.2*) and is simple to handle that makes it the good choice for data analysis. Python libraries provide various functionalities such as data manipulation, statistical analysis and visualization. Python libraries that are widely used are Numpy, Pandas, Scipy and Matplotlib. These libraries help in data analysis of the data that is too large to be handled manually. Some libraries also provide graphical representations for the data that help the data scientists and researchers for better understanding.

3.1. OVERVIEW OF PYTHON LIBRARIES

Python has a vast variety of libraries and for data analysis there are all sorts of libraries and even alternative libraries that can be used for the same purpose. Here are some key libraries that are widely used in data analysis.

3.1.1. Numpy:

Numpy (Numerical Python) is a package used for extensive numerical computations. To perform any sort of computation in Python, the Numpy library is used as it can provide great efficiency and gives faster output. It can also do numerical computing on multi-dimensional data like arrays and objects. (Saabith, A. S., Vinodhraj, T 2020, p24).

3.1.2. Pandas:

Pandas (Python Data Analysis) is the most popular library it is widely used for the purpose of data manipulation alongside with Numpy, and Scipy. Pandas is used for large volume structured data

whose size goes around in gigabytes. (Stepanek, H. (2020.p1) It also provide the feature of handling the missing data in the dataset if any. For real world data analysis, pandas is used as it has high performance and can be used easily without any complications.

3.1.3. SciPy:

SciPy (Scientific Python) it is a collection of packages used for scientific computations and calculations. SciPy also contains statistical methods that helps to calculate the statistics of the data. SciPy is very helpful for real world data analysis. Some of its basic functions are stats, optimize, integrate etc (McKinney, W. (2012), p.6).

3.1.4. Scikit-learn:

Scikit-learn is a machine learning package in python programming language. Thanks to its ease of use structure and performance that helps data scientist and researcher to utilize it for data driven work and analysis. Scikit-learn also provide graph visualization for easy analysis. (Bonald, T., De Lara, N., Lutz, Q., & Charpentier, B. 2020, p1).

3.1.5. Matplotlib:

Matplotlib is widely used for generating plots, graphs and two or three dimension visuals for the data. Matplotlib provides all sorts of output that can be used by data scientist. It used along side with pandas together it will create a better lineup for data analysis and better output. (Saabith, A. S., Vinothraj, T 2020,p24-25)

3.1.6. Seaborn:

Seaborn is a visualization library built by using matplotlib. It provides basic visualizations of the data that are frequently used. Seaborn is mostly used with Pandas library data frames. Plots that are made by seaborn are often similar with matplotlib. (Sial, A. H., Rashdi, S. Y. S., & Khan, A. H. 2021, p2)

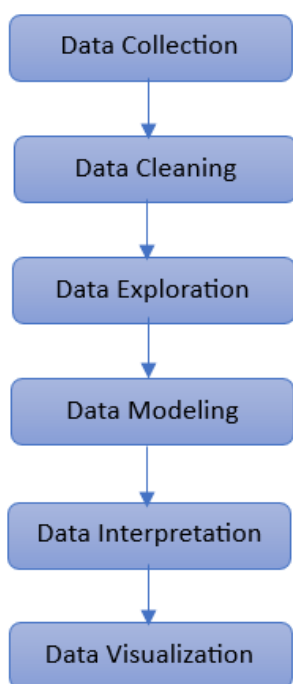
These are some most popular python libraries used for data analysis now a days and these libraries also providing assistance for machine learning. By using these libraries a data scientist can easily perform its numerical. scientific and statistical analysis is short period of time with less error chances and fast outcomes. Even with the graphical and plot representations of the data alongside the calculations.

3.2.DATA ANALYSIS USING PYTHON LIBRARIES

Data analysis is crucial for the data scientists and researchers. Using python libraries for data analysis makes it easier, faster and more efficient. By these libraries data analysis is simpler from manually calculating and analyze the data. Data scientist often wants to perform various calculations that takes a lot more time and by using these libraries the results are produced in seconds, making it time efficient as well. Earlier there were no big data available that requires a lot of processing before the analysis but in this era every thing rely on data and the size of data is increasing for each sector exponentially and to analyze huge data the old tactics will not work accurately. Using python for analysis will not only boost the outcome efficiency but also minimize the human work load to calculate and evaluate all the data by themselves even using a programming language it takes time if any library is not used along with the analysis.

The data analysis using Python have some series of steps to follow like a workflow that has to be done in order because skipping any step can lead to inaccurate results. The basic data analysis workflow using python according to (www.polymersearch.com) is shown in figure 1

Fig.3 Data Analysis workflow



Source: Own representation

The above figure shows the steps that is used in data analysis. Starting from the Data Collection which basically can be databases, csv (comma separated file), excel files etc. These form of data files are used. In python these files are loaded using pandas library that are than further processed.

Processed means handling of empty or null data. Here is the example of data collection using pandas and creating a data frame

Fig.4 Data Collection using Pandas

```
import pandas as pd

# Load dataset
df = pd.read_csv('data.csv')
df.head()
```

	Type_of_Renewable_Energy	Installed_Capacity_MW	Energy_Production_MWh	Energy_Consumption_MWh	Energy_Storage_Capacity_MWh
0	4	93.423205	103853.2206	248708.4892	2953.248771
1	4	590.468942	190223.0649	166104.1642	5305.174042
2	1	625.951142	266023.4824	424114.6308	2620.192622
3	1	779.998728	487039.5296	308337.7316	1925.250307
4	3	242.106837	482815.0856	360437.7705	3948.945383

Source: Own representation

After data collection, if there is any null values it can create ambiguity in the output so it has to be handled. For null or not available records in the data it can be dropped from the data for precautionary measure. By using Pandas library it makes it easier to handle null values just by using single line of code it is done. It is considered to be cleaning of the data. Here is the code snippet for the dropping of null values from the data.

Fig.5 Dropping Null values from data

```
import pandas as pd

# Load dataset
df = pd.read_csv('data.csv')

# Data Cleaning
df.dropna(inplace=True) # Drop missing values
df.head()
```

	Type_of_Renewable_Energy	Installed_Capacity_MW	Energy_Production_MWh	Energy_Consumption_MWh	Energy_Storage_Capacity_MWh
0	4	93.423205	103853.2206	248708.4892	2953.248771
1	4	590.468942	190223.0649	166104.1642	5305.174042
2	1	625.951142	266023.4824	424114.6308	2620.192622
3	1	779.998728	487039.5296	308337.7316	1925.250307
4	3	242.106837	482815.0856	360437.7705	3948.945383

Source: Own representation

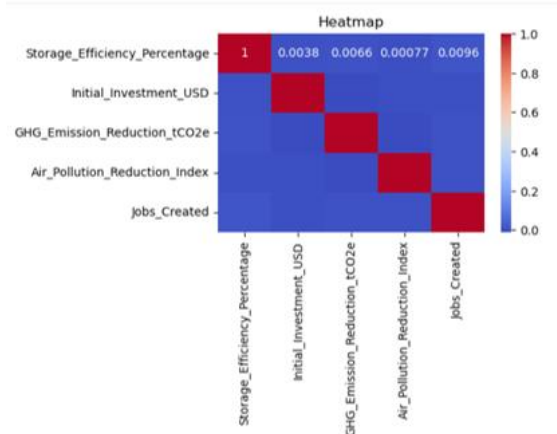
Now after data cleaning, data exploration, modeling, interpretation can be done according to the data scientist. After that data analysis final step is for data visualization can be done. There are many types of graph that are used by data scientist for data analysis some are heatmap, line graph, histogram

etc. These graphs are made by using matplotlib library. Here is the different types of visualizations using matplotlib in python.

Fig.6 heatmap

```
# Creating a correlation matrix
corr_matrix = df[important_columns].corr()

# Plotting the heatmap
plt.figure(figsize=(5, 3))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Heatmap')
plt.show()
```



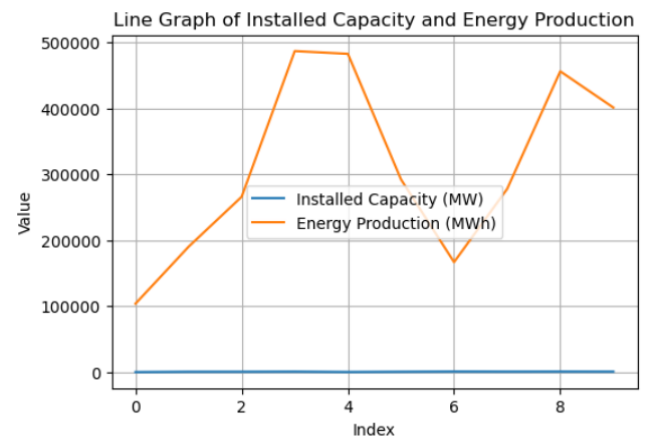
Source: Own representation

Fig.7 Line graph

```
df = df.head(10)

plt.figure(figsize=(6, 4))
plt.plot(df['Installed_Capacity_MW'], label='Installed Capacity (MW)')
plt.plot(df['Energy_Production_MWh'], label='Energy Production (MWh)')

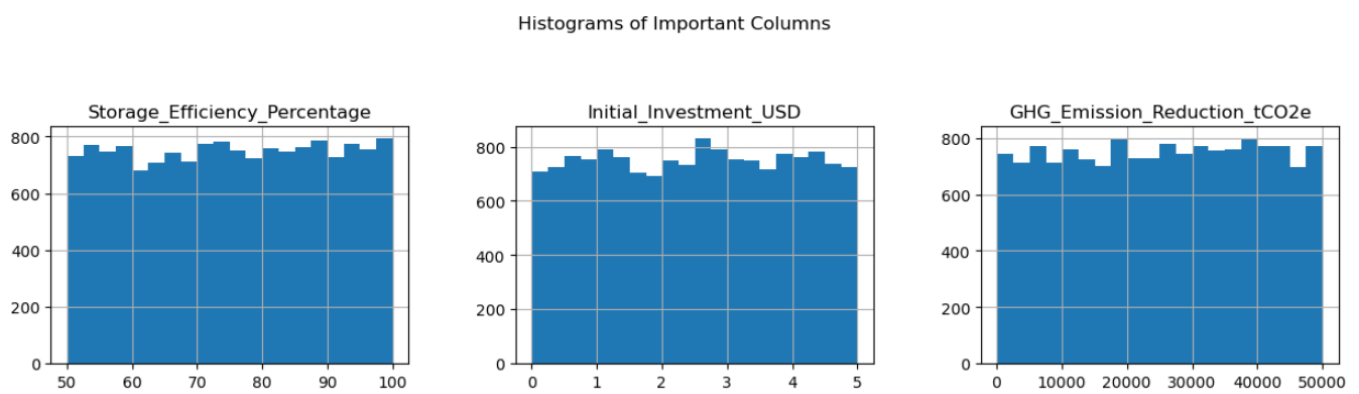
plt.title('Line Graph of Installed Capacity and Energy Production')
plt.xlabel('Index')
plt.ylabel('Value')
plt.legend()
plt.grid(True)
plt.show()
```



Source: Own representation

Fig.8 Histogram

```
df[important_columns].hist(bins=20, figsize=(15, 10), layout=(3, 3))
plt.suptitle('Histograms of Important Columns')
plt.show()
```



Source: Own representation

These are some of the data visualizations for the data analysis using matplotlib python library. There are many more graph and charts available to use. Making graphs and chart become very simple just by using matplotlib and it is saving the time for creating the graphs manually like traditionally used

before these libraries were launched. (International Journal of Innovative Technology and Exploring Engineering (IJITEE),2019, p.3)

4. COMPARISON BETWEEN DATA ANALYSIS USING PYTHON LIBRARIES AND WITHOUT LIBRARIES

Data analysis using python without libraries is not being in practice since there are many advance python libraries that are available that facilitates the data scientist to efficiently work on data with all sort of computation and visualizations. While before these libraries the data manipulation, cleaning and visualization takes longer time and have chances of some errors due to manual computation is performed on the data. Using Python libraries can also minimize the human efforts and the time it takes to each and every step manually and make separate custom functions for it will also be replaced. The table below shows the side by side qualitative comparison for both data analysis using python with or without libraries.

Table.1 Data analysis with python vs without python

Aspect	With Python Libraries	Without Python Libraries
Efficiency and Speed	Fast Execution	Slow Execution
Data Loading	Efficient	Slow
Data Manipulation	Fast Execution	Slow Execution
Statistical Analysis	Fast, reliable	Slow, higher computation
Visualization	Efficient, Easy	Manual drawing
Ease of use	Easy to write and understand	Require extensive code
Dependencies	Require installation of external library	No dependencies
Development time	Rapid development	Time consuming
Educational Value	Good for learning	Excellent for understanding concepts

Source: Own representation

4.1. Detail Comparison

According to Li Jun, & Li Ling. (2010) python has increased its processing speed in every releases libraries like pandas, numpy and scipy are used on large size datasets. Loading data using pandas is efficient as it is designed to handle large file swiftly. While with old traditional methods are significantly slower for example: manual reading a csv file and parsing it line by line will take a lot much longer because reading files involve iterations over each line and splitting it makes it slower and prone to error. So using a library that will be a better option while doing data analysis using python language.

Data manipulation is one of the fundamentals of data analysis and by using pandas for high level functions like grouping, merge, aggregate become easier and faster as the data manipulation without library require custom functions and explicit loops making it complex and slower. According to Bröker, O., Chinellato, O., & Geus, R. (2005, p.972) the complexity of the code and calculations makes it slower to execute and hence using python libraries for data manipulation is better option rather than making custom functions manually. Other than that, statistical analysis using python libraries are also vary efficient. The commonly used library for statistical analysis is scipy. (Nunez-Iglesias, Van der Walt, & Dashnow, 2017, p.223). Its output is faster and reliable while if statistical analysis is done manually the chances of errors become higher. According to Cai, X., Langtangen, H. P., & Moe, H. (2005, p.55) the numerical computing in python without library like numpy is making the execution a lot more slower as the complexity and length of the numerical computation is, so library is preferable.

Data Visualization is key for better analysis as it provide the summarized outcome from the statistical analysis. The most widely used data visualization library in python language is matplotlib and it provide all sorts of graphs, charts. Its use is straightforward and easy to use. (ABDİMANAPOVA, G., ZHAİDAKBAYEVA, & ALDESHOV, S.2023 p1). The line of code will be minimal and it can process large data aswell. While manually drawing graphs and charts require excessive time and precision to be accurate as any minor mistake can change the output of the analysis as the data analyst will go on with the error output. Furth more, the libraries that are used in data analysis cannot be used without installing it into the development environment. Each and every library has to be installed before it can be used in the program. Without any installed library nothing will work. It might cause dependency issues in some environments. On the other hand, there is no dependencies are required if there is no use of external libraries which will not cause any dependencies issues in the data analysis work.

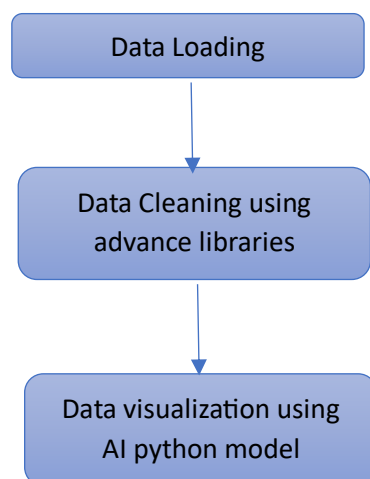
Development time for data analysis using python is much faster than that of the without one because all the functions that are used are not required to be written manually while without library work will takes time for creating each functionally manually. But for concepts it is preferable the data analyst should practice manual before using libraries.

4.2. Future of Data Analysis using Libraries

Data Analysis is getting advanced day by day and in some years, it will be more advanced. This modern era using data for each and everything and size of data is exceeding to handle it the upgradation is required for everything. Now for its future the data analysis will be also taking over by the artificial intelligence to make the data analysis more efficient and accurate than ever before. In near future it may be possible that python library will provide full three dimensional visualization and real time data analysis with the help of machine learning libraries in python.

The vast advancement of technology will also impact how the data analysis is performed and how can it be done without any human interactions to avoid human error and to get 100% accurate results. This will take a lot of time happen as the artificial intelligence is still in its development stage to take over whole data analysis and statistical analysis by itself. But some form of AI managed data analysis is still available which is again based on data. The future scope of the data analysis is illustrated in the figure below

Fig.9: Data analysis future scope



5.CONCLUSION

A good understanding of statistical ideas and basic programming concepts are necessary to perform data analysis in Python using conventional approaches without the use of libraries. Traditional methods provide important insights into the underlying mechanics of data processing and improve problem-solving abilities, even when modern libraries offer practical and effective tools for data analysis. Gaining proficiency with these fundamental techniques can help one understand the intricacies of data analysis and the power of programming on a deeper level.

Python libraries are highly recommended for most practical data analysis jobs because of their efficiency, sophisticated functionality, and user-friendliness. Nevertheless, doing data analysis without libraries offers more insightful education and a more thorough comprehension of the underlying ideas, which can be useful in educational settings or in settings where external dependencies are strictly limited. By comparing the data analysis techniques using python libraries and without the libraries gives the conclusion that the using the libraries to perform the task of data analysis is beneficial as compared to the data analysis without using python library due to its time consumption and slow response time.

6. Relation between Research Work & Practical

So, by considering the given datasets and the requirements of the practical I used multiple libraries to solve it which are explained above in detail and each library has its own responsibility like I used panda to read all the given dataset which is in CSV format, I used numpy to perform calculation based on mathematics. Also I used curve_fit function from the scipy library to fetch out all the predicted 'Y values' by manipulating the given training dataset. On the other hand it is also possible to solve the practical without using the libraries but it is not a good approach because the number of **lines of code** would be too much and also the **complexity** of the functions will increase and the chances of **occurrence of bugs** will also increase. One more thing is that there is **no way in python to visualize** the data without using its library that's why I used bokeh for this. This is how my research work is directly connected to the given data.

7.REFERNECES:

- [1]. McKinney, W. (2012). *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. " O'Reilly Media, Inc."
- [2]. Saabith, A. S., Vinothraj, T., & Fareez, M. (2020). Popular python libraries and their application domains. *International Journal of Advance Engineering and Research Development*, 7(11).
- [3]. Stepanek, H. (2020). *Thinking in Pandas*. Berkeley, CA, USA: Apress.
- [4]. Bonald, T., De Lara, N., Lutz, Q., & Charpentier, B. (2020). Scikit-network: Graph analysis in python. *Journal of Machine Learning Research*, 21(185), 1-6.
- [5]. Sial, A. H., Rashdi, S. Y. S., & Khan, A. H. (2021). Comparative analysis of data visualization libraries Matplotlib and Seaborn in Python. *International Journal*, 10(1), 45.
- [6]. Nunez-Iglesias, J., Van der Walt, S., & Dashnow, H. (2017). *Elegant SciPy: The Art of Scientific Python*. " O'Reilly Media, Inc."
- [7]. Jun, L., & Ling, L. (2010, October). Comparative research on Python speed optimization strategies. In *2010 International Conference on Intelligent Computing and Integrated Systems* (pp. 57-59). IEEE.
- [8]. Bröker, O., Chinellato, O., & Geus, R. (2005). Using Python for large scale linear algebra applications. *Future Generation Computer Systems*, 21(6), 969-979.
- [9]. ABDİMANAPOVA, G., ZHAİDAKBAYEVA, L., & ALDESHOV, S. (2023). GRAPHICS CAPABILITIES IN PYTHON. *Avrasya Sosyal ve Ekonomi Araştırmaları Dergisi*, 10(1), 1-14.
- [10]. Cai, X., Langtangen, H. P., & Moe, H. (2005). On the performance of the Python programming language for serial and parallel scientific computations. *Scientific Programming*, 13(1), 31-56.

8. APPENDIX

8.1 About Code

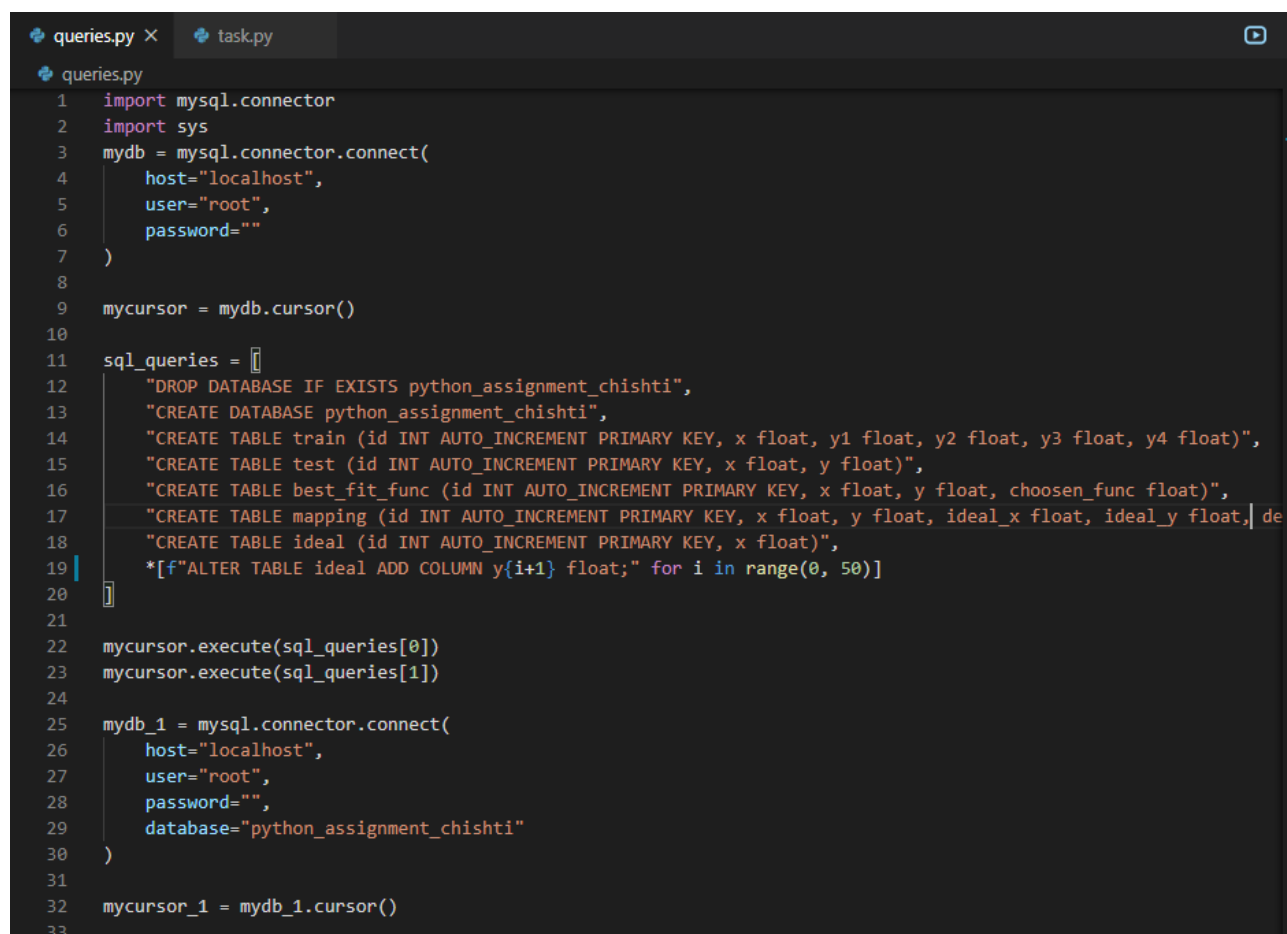
Github Link: https://github.com/abdul-ghani-chishti/programming_with_python_assignment.git

Note: The practical work have two python files the first one named queries.py and task.py

- 1) Queries.py -> is responsible to create database if not exist and also to create tables.
- 2) Task.py -> it is a main file which have all the operations.
- 3) Used <https://docs.bokeh.org/en/latest/> for visualization.
- 4) Used <https://www.w3schools.com/python/> for code as a reference.
- 5) Used <https://pythonnumericalmethods.studentorg.berkeley.edu/notebooks/chapter16.04-Least-Squares-Regression-in-Python.html> for least square method.

8.2 Code

Queries.py



```
1 import mysql.connector
2 import sys
3 mydb = mysql.connector.connect(
4     host="localhost",
5     user="root",
6     password=""
7 )
8
9 mycursor = mydb.cursor()
10
11 sql_queries = [
12     "DROP DATABASE IF EXISTS python_assignment_chishti",
13     "CREATE DATABASE python_assignment_chishti",
14     "CREATE TABLE train (id INT AUTO_INCREMENT PRIMARY KEY, x float, y1 float, y2 float, y3 float, y4 float)",
15     "CREATE TABLE test (id INT AUTO_INCREMENT PRIMARY KEY, x float, y float)",
16     "CREATE TABLE best_fit_func (id INT AUTO_INCREMENT PRIMARY KEY, x float, y float, choosen_func float)",
17     "CREATE TABLE mapping (id INT AUTO_INCREMENT PRIMARY KEY, x float, y float, ideal_x float, ideal_y float, de",
18     "CREATE TABLE ideal (id INT AUTO_INCREMENT PRIMARY KEY, x float)",
19     *[f"ALTER TABLE ideal ADD COLUMN y{i+1} float;" for i in range(0, 50)]
20 ]
21
22 mycursor.execute(sql_queries[0])
23 mycursor.execute(sql_queries[1])
24
25 mydb_1 = mysql.connector.connect(
26     host="localhost",
27     user="root",
28     password="",
29     database="python_assignment_chishti"
30 )
31
32 mycursor_1 = mydb_1.cursor()
33
```



```

31
32     mycursor_1 = mydb_1.cursor()
33
34     for i, table in enumerate(sql_queries):
35         if i > 1:
36             mycursor_1.execute(table)
37
38     print("Database and Tables Created Successfully !")
39

```

Task.py

```

queries.py  task.py  X
task.py
1  import numpy as np
2  import pandas as pd
3  from scipy.optimize import curve_fit
4  import mysql.connector
5  from mysql.connector import Error
6  import sys
7  from bokeh.plotting import figure, show, output_file
8  from bokeh.models import ColumnDataSource, HoverTool
9
10 db_connection = mysql.connector.connect(
11     host="localhost",
12     user="root",
13     password="",
14     database="python_assignment_chishti"
15 )
16
17 mycursor_1 = db_connection.cursor()
18 mycursor_1.execute("show databases")
19 databases = mycursor_1.fetchall()
20 database_name = [i[0] for i in databases]
21
22 if 'python_assignment_chishti' in database_name:
23     print('true')
24 else:
25     print('Database not found, First run queries.py on terminal !')
26     sys.exit()
27

```

```

queries.py task.py x
task.py
26 sys.exit()
27
28 # CSV files to variables
29 def load_data(file_path):
30     data = pd.read_csv(file_path)
31     x_val = data['x'].values
32     y_cols = [col for col in data.columns if col.startswith('y')]
33     y_val = data[y_cols].values
34     return x_val, y_val
35
36 # ideal function
37 def ideal_function(x, a, b):
38     return a * x + b
39
40 # Fit ideal function to training data
41 def fit_ideal_functions(training_data, ideal_functions):
42     best_function_val = []
43     min_deviation = float('inf')
44     min_sum_squared_deviations_array = []
45     x_train, y_train = training_data
46
47     j = 0
48     for y in y_train:
49         if j < 4:
50             for y_ideal in ideal_functions:
51                 # Fit only the first column of y_train
52                 params, _ = curve_fit(ideal_function, x_train, y_train[:, j])
53                 y_pred = ideal_function(x_train, *params)
54                 # Calculate deviation only for the first column
55                 sum_squared_deviations = np.sum((y_train[:, j] - y_pred) ** 2)
56
57                 if sum_squared_deviations < min_deviation:
58                     min_deviation = sum_squared_deviations

```

```

queries.py task.py x
task.py
57         if sum_squared_deviations < min_deviation:
58             min_deviation = sum_squared_deviations
59             best_function_params = params
60         j = j+1
61         best_function_val.append(best_function_params)
62         min_sum_squared_deviations_array.append(min_deviation)
63     return best_function_val, min_sum_squared_deviations_array
64     # put these values into the best_fit_function table
65
66 # Map test data to chosen ideal functions
67 def map_test_data(test_data, chosen_functions):
68     x_test, y_test = test_data
69     mappings = []
70
71     for x, y in zip(x_test, y_test):
72         deviations = []
73         for params in chosen_functions:
74             y_pred = ideal_function(x, *params)
75             deviation = np.abs(y - y_pred)
76             deviations.append(deviation)
77         min_deviation = min(deviations)
78         if min_deviation < np.sqrt(2) * np.max(deviations):
79             best_fit_index = deviations.index(min_deviation)
80             best_fit_params = chosen_functions[best_fit_index]
81             mappings.append((x, y, best_fit_params, min_deviation))
82         else:
83             mappings.append((x, y, None, None))
84     # print(mappings[0])
85     # sys.exit()
86     return mappings
87
88

```

```

queries.py task.py X
task.py
88
89 x_train, y_train = load_data("train.csv") # training data (A)
90 ideal_functions = load_data("ideal.csv") # ideal data (C)
91 x_test, y_test = load_data("test.csv") # test data
92
93 try:
94     train_data_list = [
95         (x, y1, y2, y3, y4)
96         for x, y1, y2, y3, y4 in zip(x_train, y_train[:, 0], y_train[:, 1], y_train[:, 2], y_train[:, 3])
97     ]
98     mycursor_1.executemany(
99         "INSERT INTO train (x, y1, y2, y3, y4) VALUES (%s, %s, %s, %s, %s)",
100         train_data_list
101     )
102     db_connection.commit()
103
104     test_data_list = [
105         (x, y)
106         for x, y in zip(x_test, y_test[:, 0])
107     ]
108     mycursor_1.executemany(
109         "INSERT INTO test (x, y) VALUES (%s, %s)",
110         test_data_list
111     )
112     db_connection.commit()
113
114     x_ideal, y_ideal = ideal_functions
115     ideal_data_list = [
116         (x, *y_row)
117         for x, y_row in zip(x_ideal, y_ideal)
118     ]
119     mycursor_1.executemany(

```

```

queries.py task.py X
task.py
119 mycursor_1.executemany(
120     "INSERT INTO ideal (x, " + ", ".join(
121         [f"y{i+1}" for i in range(50)]) + ") VALUES (" + ", ".join(["%s"] * 51) + ")",
122     ideal_data_list
123 )
124 db_connection.commit()
125
126 print("Data insertion completed successfully.")
127 except Error as e:
128     # raise Exception("Sorry")
129     print(f"Something Went Wrong or maybe database got crash !!! {e}")
130 finally:
131     print("Enjoy !")
132
133 # Choose the four best fitting ideal functions
134 chosen_functions = fit_ideal_functions((x_train, y_train), ideal_functions)
135
136 # Map test data to chosen ideal functions
137 mappings = map_test_data((x_test, y_test), chosen_functions[0])
138
139 print("Chosen ideal function parameters:")
140 print("S.No      x      y      chosen_val")
141 > for i, params in enumerate(chosen_functions[0]): ...
146 db_connection.commit()
147
148 print()
149 print("Mappings:")
150 print()
151 print(" x      y      Best fit (A)      Best fit (B)      Deviation")
152 for i, (x, y, best_fit_params, deviation) in enumerate(mappings):
153     if best_fit_params is not None:
154         print(x, y[0], best_fit_params[0], best_fit_params[1], deviation[0])

```

```

queries.py task.py x
task.py
154     print(x, y[0], best_fit_params[0], best_fit_params[1], deviation[0])
155     data_to_insert = [(x, y[0], best_fit_params[0],
156                       best_fit_params[1], deviation[0])]
157     mycursor_1.executemany(
158         "INSERT INTO mapping (x,y,ideal_x,ideal_y,deviation) VALUES (%s, %s, %s, %s, %s)", data_to_insert)
159     else:
160         print(f>Data point {
161             i+1}: x={x}, y={y}, No best fit found within the deviation threshold")
162     db_connection.commit()
163
164     # Graphing
165     def plot_data_with_bokeh(x_train, y_train, x_test, y_test, chosen_functions, mappings):
166         try:
167             p = figure(title="Training Data, Ideal Functions, Test Data, and Mappings",
168                       x_axis_label='x', y_axis_label='y', width=800, height=600)
169
170             # Plot training data
171             for i in range(y_train.shape[1]):
172                 p.scatter(x_train, y_train[:, i], legend_label=f'Training y{
173                     i+1}', color='green', size=6, alpha=0.6)
174
175             # Plot ideal functions
176             x_range = np.linspace(min(x_train), max(x_train), 500)
177             for i, params in enumerate(chosen_functions[0]):
178                 y_range = ideal_function(x_range, *params)
179                 p.line(x_range, y_range, legend_label=f'Ideal Function {
180                     i+1}', color='pink', line_width=2)
181
182             # Plot test data
183             for i in range(y_test.shape[1]):
184                 p.scatter(x_test, y_test[:, i], legend_label=f'Test Data y{
185                     i+1}', color='black', size=6, alpha=0.6)

```

```

queries.py task.py x
task.py
186
187     # Plot mappings with deviation
188     mapped_x = [x for x, y, function_idx,
189                 deviation in mappings if function_idx is not None]
190
191     mapped_y = [y for x, y, function_idx,
192                 deviation in mappings if function_idx is not None]
193     deviations = [deviation for x, y, function_idx,
194                  deviation in mappings if function_idx is not None]
195     source = ColumnDataSource(
196         data=dict(x=mapped_x, y=mapped_y, deviation=deviations))
197
198     p.scatter('x', 'y', source=source, color='red', size=10,
199             alpha=0.6, legend_label='Mapped Test Data')
200
201     hover = HoverTool()
202     hover.tooltips = [("@x", "@x"), ("y", "@y"),
203                       ("deviation", "@deviation")]
204     p.add_tools(hover)
205
206     p.legend.location = "top_left"
207     output_file("mapping.html")
208     show(p)
209     except Error as e:
210         print(f"Something Went Wrong !!! {e}")
211
212     plot_data_with_bokeh(x_train, y_train, x_test, y_test,
213                         chosen_functions, mappings)
214
215     def plot_data_xtrain_ytrain(x,y):
216         p = figure(title="Training Dataset", x_axis_label="x", y_axis_label="y")
217
218         p.line(x, y[0], legend_label="x y1", color="yellow", line_width=2)

```

```

queries.py task.py x
task.py
217
218 p.line(x, y[:, 0], legend_label="x y1", color="yellow", line_width=2)
219 p.line(x, y[:, 1], legend_label="x y2", color="green", line_width=2)
220 p.line(x, y[:, 2], legend_label="x y3", color="blue", line_width=2)
221 p.line(x, y[:, 3], legend_label="x y4", color="red", line_width=2)
222
223 output_file("train.html")
224 show(p)
225 plot_data_xtrain_ytrain(x_train, y_train)
226
227 def plot_data_xtest_ytest(x, y):
228
229     p = figure(title="Test Dataset", x_axis_label="x", y_axis_label="y")
230
231     p.line(x, y[:, 0], legend_label="x y", color="yellow", line_width=2)
232
233     output_file("test.html")
234     show(p)
235 plot_data_xtest_ytest(x_test, y_test)
236
237 print("Enjoy & Have A Nice Day <3")

```

8.3 RESULT

```

C:\Windows\System32\cmd.exe
D:\IU University of applied science\1st semester\quarter 1\programming with python\assignment\1st solution>queries.py
Database and Tables Created Successfully !

D:\IU University of applied science\1st semester\quarter 1\programming with python\assignment\1st solution>task.py
true
Data insertion completed successfully.
Enjoy !
Chosen ideal function parameters:
S.No      x              y              chosen_val
1: -1.9978981440449517, 0.007771132716573925, 87.11627242584481
2: 1.998638834500751, 0.00569289893183722, 86.13857933247337
3: 0.9996282494243194, -0.0016468939684690653, 29.853399197190416
4: 0.9996282494243194, -0.0016468939684690653, 29.853399197190416

Mappings:
x      y      Best fit (A)      Best fit (B)      Deviation
17.5  34.16104  1.998638834500751  0.00569289893183722  0.8208325026949836
0.3    1.2151024  1.998638834500751  0.00569289893183722  0.6098178507179374
-8.7   -16.843908  1.998638834500751  0.00569289893183722  0.5385569612246961
-19.2  -37.17087  1.998638834500751  0.00569289893183722  1.1973027234825864
-11.0  -20.263054  1.998638834500751  0.00569289893183722  1.7162802805764237
0.8    1.4264555  1.998638834500751  0.00569289893183722  0.178148466532438
14.0   -0.06650608  0.9996282494243194  -0.0016468939684690653  14.059654677972002
-10.4  -2.007094  0.9996282494243194  -0.0016468939684690653  8.390686687981392
-15.0  -0.20536347  0.9996282494243194  -0.0016468939684690653  14.790707165333261
5.8    10.711373  1.998638834500751  0.00569289893183722  0.8864251390361932
-7.6   -39.4954  1.998638834500751  0.00569289893183722  24.31143775672613
-19.8  -19.915014  0.9996282494243194  -0.0016468939684690653  0.12072776743000446

```

8.4 VISUALIZATION

