Implementation of Data Management using Object-Oriented Programming (OOP) in Python

Ahmad Farhan AlShammari
Department of Computer and Information Systems
College of Business Studies, PAAET
Kuwait

ABSTRACT
The goal of this research is to implement data management using object-oriented programming (OOP) in Python. Data management is the process of handling data efficiently by performing the basic operations on data. It helps to keep data organized, accessible, accurate, and secure. It also provides a solid foundation for data analysis and decision making.

The basic operations of data management are explained: defining table, creating table, displaying table, displaying shape, displaying field names, displaying data types, displaying row, displaying column, adding row, adding column, updating row, updating column, deleting row, deleting column, getting values, counting values, computing statistics (count, min, max, mean, and std), searching by value, sorting by column, grouping by column, and clearing table.

The developed program was tested on an experimental dataset. The program has successfully performed the basic operations of data management using object-oriented programming and provided the required results.

Keywords
Computer Science, Artificial Intelligence, Machine Learning, Data Science, Data Management, Object-Oriented Programming, OOP, Python, Programming.

1. INTRODUCTION
In the recent years, machine learning has played a major role in the development of computer systems. Machine learning (ML) is a branch of Artificial Intelligence (AI) that focuses on developing models and algorithms to improve the performance and efficiency of computer programs [1-12].

Data management is a fundamental concept in the field of machine learning. It is also sharing knowledge with other related fields like: programming, data science, mathematics, statistics, and numerical methods [13-17].

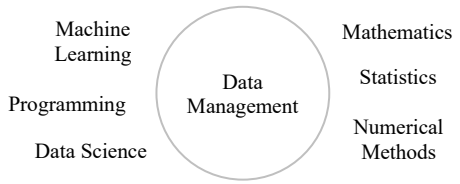

Fig 1: Area of Data Management

Simply, data management is about managing data. It is done by performing the basic operations of data management: defining, creating, displaying, adding, updating, deleting, searching, sorting, grouping, and clearing. It helps to keep data organized, accessible, accurate, and secure.

2. LITERATURE REVIEW
The literature was reviewed to understand the fundamental concepts, methods, and applications of data management [18-23] using object-oriented programming [24-36].

Data management is the "cornerstone" of the information age. It is strongly connected to the evolution of computing technology. The early practices of data management trace back to the 1950s, where data was stored in punch cards and processed manually.

In the 1970s, the database management systems (DBMS) were introduced. The relational model was proposed by Codd [37]. Then, the structured query language (SQL) was developed by Chamberlin and Boyce at IBM [38]. Actually, they still form the "backbone" of data management systems today.

Now, with the evolution of internet and web applications, the volume and complexity of data have increased dramatically. This led to the emergence of new concepts like: NoSQL, big data, cloud computing, data mining, and machine learning.

In fact, data management is the "core" concept of computer systems because data is the "most valuable" asset in the organization (in both the operational and strategic levels).

The fundamental concepts of data management using object-oriented programming are explained in the following section.

Data Management:
Data management is the process of storing, organizing, and manipulating data efficiently. This includes performing the basic operations on data: defining, creating, displaying, adding, updating, deleting, searching, sorting, grouping, and clearing.

The goal of data management is to make sure that data is organized, accessible, accurate, and secure. It also provides support for data analysis and decision making.

The concept of data management is illustrated in the following diagram:

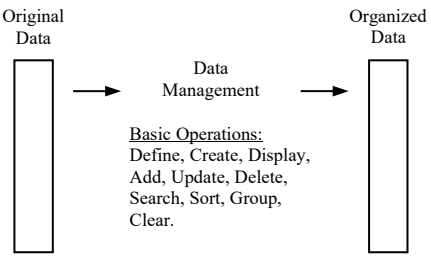

Fig 2: Concept of Data Management

Table:

Table is the most familiar form in data management. It is widely used in documents, files, spreadsheets, and databases. Simply, table is a two-dimensional data structure that consists of rows and columns. It can be represented as shown in the following diagram:

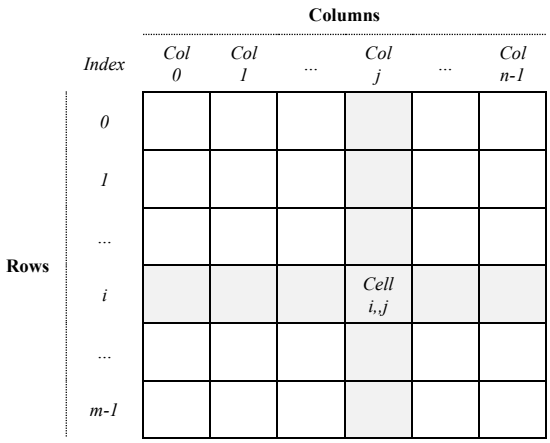

Fig 3: Representation of Table

The rows are identified by index numbers, and the columns by column names. The data values are stored in the cells of the table, where each cell (*i*,*j*) is the intersection of row (*i*) and column (*j*).

Table Operations:

The basic operations of table are shown in the following list:

- Defining table
- Creating table
- Displaying table
- Displaying shape
- Displaying field names
- Displaying data types
- Displaying row
- Displaying column
- Adding row
- Adding column
- Updating row
- Updating column
- Deleting row
- Deleting column
- Computing statistics
- Searching by value
- Sorting by column
- Grouping by column
- Clearing table

Object-Oriented Programming:

Object-Oriented Programming (OOP) is an advanced approach of programming which is based on the concept of "objects". They are similar to the entities in the real-world. The object consists of attributes and methods, where the attributes are used to store data and the methods are used to process data.

Class and Object:

To create an "object", the "class" should be defined first. Then, the object is created exactly as specified in the class definition.

For example, the table class (*Table*) is defined first, then the table object (*tbl*) is created based on the class definition.

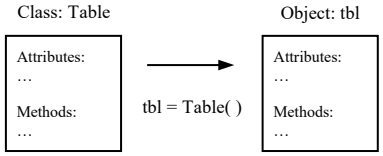

Fig 4: Explanation of Class and Object

Implementation in Python:

The table is defined and created using object-oriented programming in Python as shown in the following code:

```
# define class
class Table:
    def __init__(self, fields, rows):
        self.fields = fields
        self.rows = rows
# data: field names and rows
fields = [field0, field1, …, fieldn-1]
rows = [[v0,0, v0,1, …, v0,n-1],
        [v1,0, v1,1, …, v1,n-1],
        [v2,0, v2,1, …, v2,n-1],
         …,
        [vm-1,0, vm-1,1, …, vm-1,n-1]]
# create object
tbl = Table(fields, rows)
```

Where: (*Table*) is the table class, (*fields*) is the column names, (*rows*) is the data rows, (*tbl*) is the table object.

For example, the table object (*tbl*) is created from the table class (*Table*) as shown in the following code:

```
class Table:
    def __init__(self, fields, rows):
        self.fields = fields
        self.rows = rows
fields = ['A', 'B', 'C', 'D']
rows = [[1, 2, 3, 4],
        [5, 6, 7, 8],
        [9,10,11,12],
        [13,14,15,16]]
tbl = Table(fields, rows)
```

The table object (*tbl*) is represented by the following form:

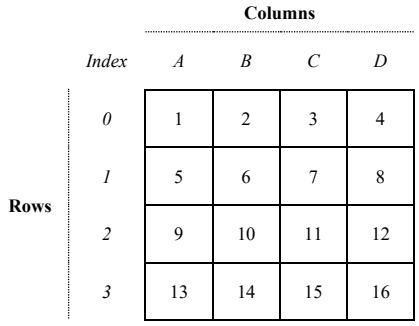

Fig 5: Representation of Table Object

Data Management System:

The data management system is briefly described in the following summary:

Input: Original data.
Output: Organized data.
Processing: First, the table is defined and created. Then, the data is stored in the table. Next, the table is displayed. After that, the basic operations of data management are performed: displaying row, displaying column, adding row, adding column, updating row, updating column, deleting row, deleting column, computing statistics, searching by value, sorting by column, grouping by column, and clearing table.

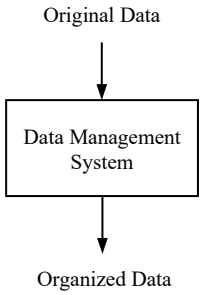

Fig 6: Data Management System

Python:

Python [39] is an open source, object-oriented, and general-purpose programming language. It is simple, easy to learn, and powerful. It is the most popular programming language especially for the development of machine learning applications.

Python provides many additional libraries for different purposes. For example: Numpy [40], Pandas [41], Matplotlib [42], Seaborn [43], NLTK [44], SciPy [45], and SK Learn [46].

3. RESEARCH METHODOLOGY

The basic operations of data management are: (1) defining table, (2) creating table, (3) displaying table, (4) displaying shape, (5) displaying column names, (6) displaying data types, (7) displaying row, (8) displaying column, (9) adding row, (10) adding column, (11) updating row, (12) updating column, (13) deleting row, (14) deleting column, (15) getting values, (16) counting values, (17) computing statistics (count, min, max, mean, and std), (18) searching by value, (19) sorting by column, (20) grouping by column, and (21) clearing table.

- Defining table
- Creating table
- Displaying table
- Displaying shape
- Displaying field names
- Displaying data types
- Displaying row
- Displaying column
- Adding row
- Adding column
- Updating row
- Updating column
- Deleting row
- Deleting column
- Getting values
- Counting values
- Computing statistics
- Searching by value
- Sorting by column
- Grouping by column
- Clearing table

Fig 7: Basic Operations of Data Management

The basic operations of data management using object-oriented programming are explained in the following section.

Note: The program is developed using the standard functions of Python, without any additional library like Numpy or Pandas.

1. Defining Table:

The table class (*Table*) is defined by the following code:

```
class Table:
    def __init__(self, fields, rows):
        self.fields = fields
        self.rows = rows
```

2. Creating Table:

The table object (*tbl*) is created by the following code:

```
fields = [field0, field1, …, fieldm-1]
rows = [[v0,0, v0,1, …, v0,n-1],
        [v1,0, v1,1, …, v1,n-1],
        [v2,0, v2,1, …, v2,n-1],
         …,
        [vm-1,0, vm-1,1, …, vm-1,n-1]]
tbl = Table(fields, rows)
```

3. Displaying Table:

Displaying the table is performed by the following code:

```
def display(self):
    print(end="\t")
    for field in self.fields:
        print(field, end="\t")
    print()
    nf = len(self.fields)
    print("-"*8*(nf+1))
    for i, row in enumerate(self.rows):
        print(i, end="\t")
        for item in row:
            print(item, end="\t")
        print()
```

4. Displaying Shape:

Displaying the shape of table (number of rows and columns) is performed by the following code:

```
def shape(self):
    nr = len(self.rows)
    nc = len(self.rows[0])
    print("Shape:", (nr, nc))
```

5. Displaying Field Names:

Displaying the field names is performed by the following code:

```
def display_fields(self):
    print("Field Names:")
    print("-"*16)
    for i, field in enumerate(self.fields):
        print(i, "\t", field)
```

6. Displaying Data Types:

Displaying the data types of columns (int, float, boolean, or string) is performed by the following code:

```
def display_types(self):
    print("Data Types:")
    print("-"*24)
```

```
    for i, field in enumerate(self.fields):
        item = self.rows[0][i]
        print(field, "\t", type(item))
```

7. Displaying Row:

Displaying a row in the table by row index is performed by the following code:

```
def display_row(self, index):
    print(end="\t")
    for field in self.fields:
        print(field, end="\t")
    print()
    nf = len(self.fields)
    print("-"*8*(nf+1))
    print(index, end="\t")
    for item in self.rows[index]:
        print(item, end="\t")
    print()
```

8. Displaying Column:

Displaying a column in the table by field name is performed by the following code:

```
def display_col(self, field):
    index_col = self.fields.index(field)
    print("\t", field)
    print("-"*16)
    for i, row in enumerate(self.rows):
        print(i, "\t", row[index_col])
```

9. Adding Row:

Adding a new row to the table is performed by the following code:

```
def add_row(self, row):
    self.rows.append(row)
```

10. Adding Column:

Adding a new column to the table with a specific value is performed by the following code:

```
def add_col(self, field, value):
    self.fields.append(field)
    for row in self.rows:
        row.append(value)
```

11. Updating Row:

Updating a row in the table by row index and field name with a specific value is performed by the following code:

```
def update_row(self, index, field, value):
    index_col = self.fields.index(field)
    self.rows[index][index_col] = value
```

12. Updating Column:

Updating a column in the table by field name with a specific value is performed by the following code:

```
def update_col(self, field, value):
    index_col = self.fields.index(field)
    for row in self.rows:
        row[index_col] = value
```

13. Deleting Row:

Deleting a row from the table by row index is performed by the following code:

```
def del_row(self, index):
    del self.rows[index]
```

14. Deleting Column:

Deleting a column from the table by field name is performed by the following code:

```
def del_col(self, field):
    index_col = self.fields.index(field)
    for row in self.rows:
        del row[index_col]
    del self.fields[index_col]
```

15. Getting Values:

Getting the unique values of a column by field name is performed by the following code:

```
def get_values(self, field):
    index_col = self.fields.index(field)
    # compute values
    values = []
    for row in self.rows:
        item = row[index_col]
        values.append(item)
    # sort values
    values = sorted(list(set(values)))
    # print values
    print("\tValue")
    print("-"*16)
    for i, value in enumerate(values):
        print(i, "\t", value)
```

16. Counting Values:

Counting the unique values of a column by field name is performed by the following code:

```
def count_values(self, field):
    index_col = self.fields.index(field)
    # compute values
    values = {}
    for row in self.rows:
        item = row[index_col]
        if (item not in values):
            values[item] = 0
        values[item] += 1
    # sort values
    values = dict(sorted(values.items()))
    # print values
    print("Value\tCount")
    print("-"*16)
    for value, count in values.items():
        print(value, "\t", count)
```

17. Computing Statistics:

Computing the descriptive statistics (count, min, max, mean, and std) for a specific column is performed by the following code:

```
def compute_stats(self, field):
    funcs = ['Count','Min','Max','Mean','Std']
    # compute stats
    index_col = self.fields.index(field)
    items = []
    for row in self.rows:
        item = row[index_col]
        items.append(item)
```

```
    stats = [len(items),
             min(items),
             max(items),
             mean(items),
             std(items)]
    # print stats
    print(end="\t")
    for func in funcs:
        print(func, end="\t")
    print()
    nf = len(funcs)
    print("-"*8*(nf+1))
    print(field, end="\t")
    for stat in stats:
        print(stat, end="\t")
    print()
```

18. Searching by Value:

Searching a column by value is performed by the following code:

```
def search(self, field, value)
    index_col = self.fields.index(field)
    # print fields
    print(end="\t")
    for field in self.fields:
        print(field, end="\t")
    print()
    nf = len(self.fields)
    print("-"*8*(nf+1))
    # print search result
    for i, row in enumerate(self.rows):
        if (row[index_col] == value):
            print(i, end="\t")
            for item in row:
                print(item, end="\t")
            print()
```

19. Sorting by Column:

Sorting data by column (in ascending or descending order) is performed by the following code:

```
def sortby(self, field, rev=False):
    index_col = self.fields.index(field)
    self.rows.sort(reverse=rev,
                   key=lambda x: x[index_col])
```

20. Grouping by Column:

Grouping data by column and aggregating by another column using an aggregation function (count, min, max, sum, mean, and std) is performed by the following code:

```
def groupby(self, field, field_agg, func_agg):
    index_col = self.fields.index(field)
    index_agg = self.fields.index(field_agg)
    # compute groups
    groups = {}
    for row in self.rows:
        group = row[index_col]
        item = row[index_agg]
        if (group not in groups):
            groups[group] = []
        groups[group].append(item)
    # sort groups
    groups = dict(sorted(groups.items()))
    # compute results
    results = {}
    for group, items in groups.items():
        if (func_agg == "count"):
            results[group] = len(items)
        elif (func_agg == "min"):
            results[group] = min(items)
        elif (func_agg == "max"):
            results[group] = max(items)
        elif (func_agg == "sum"):
            results[group] = sum(items)
        elif (func_agg == "mean"):
            results[group] = mean(items)
        elif (func_agg == "std"):
            results[group] = std(items)
    # print results
    print("Group\tResult")
    print("-"*16)
    for group, result in results.items():
        print(group, "\t", result)
```

21. Clearing Table:

Clearing the table from data (field names and rows) is performed by the following code:

```
def clear(self):
    self.fields = []
    self.rows = []
```

4. RESULTS AND DISCUSSION

The developed program was tested on an experimental dataset. The program has successfully performed the basic operations of data management using object-oriented programming and provided the required results. The program output is shown and explained in the following section.

Displaying Table:

The table is displayed as shown in the following view:

```
tbl.display()

   Id   Name    Major          Score
--------------------------------------
0  101  Adam    Computer       79
1  102  Sarah   Science        83
2  103  John    Math           75
3  104  Mary    Computer       91
4  105  Sally   English        74
```

Displaying Shape:

The shape of table is displayed as shown in the following view:

```
tbl.shape()

Shape: (5, 4)
```

Displaying Field Names:

The field names of table are displayed as shown in the following view:

```
tbl.display_fields()

Field Names:
----------------
0  Id
1  Name
2  Major
3  Score
```

Displaying Data Types:

The data types of columns are displayed as shown in the following view:

```
tbl.display_types()
```

```
Data Types:
-----------------------
Id      <class 'int'>
Name    <class 'str'>
Major   <class 'str'>
Score   <class 'int'>
```

Displaying Row:

The row of index (3) is displayed as shown in the following view:

```
tbl.display_row(3)

   Id   Name    Job            Score
-------------------------------------
3  104  Mary    Computer       91
```

Displaying Column:

The column ('Name') is displayed as shown in the following view:

```
tbl.display_col('Name')

   Name
----------------
0  Adam
1  Sarah
2  John
3  Mary
4  Sally
```

Adding Row:

The new row [106, 'Peter', 'Computer', 82] is added to the table. Then, the table is displayed as shown in the following view:

```
tbl.add_row([106, 'Peter', 'Computer', 82])
tbl.display()

   Id   Name    Major          Score
-------------------------------------
0  101  Adam    Computer       79
1  102  Sarah   Science        83
2  103  John    Math           75
3  104  Mary    Computer       91
4  105  Sally   English        74
5  106  Peter   Computer       82
```

Adding Column:

The new column ('Bonus') of value (0) is added to the table. Then, the table is displayed as shown in the following view:

```
tbl.add_col('Bonus', 0)
tbl.display()

   Id   Name    Job            Score   Bonus
---------------------------------------------
0  101  Adam    Computer       79      0
1  102  Sarah   Science        83      0
2  103  John    Math           75      0
3  104  Mary    Computer       91      0
4  105  Sally   English        74      0
5  106  Peter   Computer       82      0
```

Updating Row:

The row of index (5) and column ('Score') is updated with a new value (84). Then, the table is displayed as shown in the following view:

```
tbl.update_row(5, 'Score', 84)
tbl.display()

   Id   Name    Job            Score   Bonus
---------------------------------------------
0  101  Adam    Computer       79      0
1  102  Sarah   Science        83      0
2  103  John    Math           75      0
3  104  Mary    Computer       91      0
4  105  Sally   English        74      0
5  106  Peter   Computer       84      0
```

Updating Column:

The column ('Bonus') is updated with a new value (10). Then, the table is displayed as shown in the following view:

```
tbl.update_col('Bonus', 10)
tbl.display()

   Id   Name    Job            Score   Bonus
---------------------------------------------
0  101  Adam    Computer       79      10
1  102  Sarah   Science        83      10
2  103  John    Math           75      10
3  104  Mary    Computer       91      10
4  105  Sally   English        74      10
5  106  Peter   Computer       84      10
```

Deleting Row:

The row of index (5) is deleted from the table. Then, the table is displayed as shown in the following view:

```
tbl.del_row(5)
tbl.display()

   Id   Name    Job            Score   Bonus
---------------------------------------------
0  101  Adam    Computer       79      10
1  102  Sarah   Science        83      10
2  103  John    Math           75      10
3  104  Mary    Computer       91      10
4  105  Sally   English        74      10
```

Deleting Column:

The column ('Bonus') is deleted from the table. Then, the table is displayed as shown in the following view:

```
tbl.del_col('Bonus')
tbl.display()

   Id   Name    Major          Score
-------------------------------------
0  101  Adam    Computer       79
1  102  Sarah   Science        83
2  103  John    Math           75
3  104  Mary    Computer       91
4  105  Sally   English        74
```

Getting Values:

The unique values of column ('Major') are obtained and displayed as shown in the following view:

```
tbl.get_values('Major')

   Value
----------------
0  Computer
1  English
2  Math
3  Science
```

Counting Values:

The unique values of column ('Major') are counted and displayed as shown in the following view:

```
tbl.count_values('Major')

Value      Count
----------------
Computer   2
English    1
Math       1
Science    1
```

Computing Statistics:

The descriptive statistics (count, min, max, mean, and std) for column ('Score') are computed and displayed as shown in the following view:

```
tbl.compute_stats('Score')

        Count   Min     Max     Mean    Std
-------------------------------------------
Score   5       74      91      80.4    6.18
```

Searching by Value:

Searching column ('Id') by value (103) is performed and displayed as shown in the following view:

```
tbl.search('Id', 103)

   Id   Name    Major           Score
-------------------------------------
2  103  John    Math            75
```

Sorting by Column:

Sorting data by column ('Name') (in ascending order) is performed and displayed as shown in the following view:

```
tbl.sortby('Name')
tbl.display()

   Id   Name    Major           Score
-------------------------------------
0  101  Adam    Computer        79
1  103  John    Math            75
2  104  Mary    Computer        91
3  105  Sally   English         74
4  102  Sarah   Science         83
```

For sorting data in descending order, the parameter (*rev*) is set to (*True*). For example: sorting data by column ('Score') in reverse order is performed and displayed as shown in the following view:

```
tbl.sortby('Score', rev=True)
tbl.display()

   Id   Name    Major           Score
-------------------------------------
0  104  Mary    Computer        91
1  102  Sarah   Science         83
2  101  Adam    Computer        79
3  103  John    Math            75
4  105  Sally   English         74
```

Grouping by Column:

Grouping data by column ('Major') and aggregating by column ('Score') using the aggregation function ('mean') is performed and displayed as shown in the following view:

```
tbl.groupby('Major', 'Score', 'mean')

Group           Result
------------------------
Computer        85.0
English         74.0
Math            75.0
Science         83.0
```

Clearing Table:

The table is cleared from data (field names and rows). Then, the table is displayed empty as shown in the following view.

```
tbl.clear()
```

Now, it is obviously clear that the program has successfully performed the basic operations of data management using object-oriented programming and provided the required results.

5. CONCLUSION

In this research, the goal was to implement data management using object-oriented programming (OOP) in Python. The literature was reviewed to explore the fundamental concepts of data management using object-oriented programming: data management, table, table operations, object-oriented programming, class and object, and implementation in Python.

The author developed a program in Python to perform the basic operations of data management using object-oriented programming: defining table, creating table, displaying table, displaying shape, displaying field names, displaying data types, displaying row, displaying column, adding row, adding column, updating row, updating column, deleting row, deleting column, getting values, counting values, computing statistics (count, min, max, mean, and std), searching by value, sorting by column, grouping by column, and clearing table.

The developed program was tested on an experimental dataset. The program has successfully performed the basic operations of data management using object-oriented programming and provided the required results.

In the future, more work is needed to improve the current methods and add new methods to fully perform the operations of data management using object-oriented programming. In addition, they should be more investigated on different fields, domains, and datasets.

6. REFERENCES

[1] Sammut, C., & Webb, G. I. (2011). "Encyclopedia of Machine Learning". Springer.

[2] Jung, A. (2022). "Machine Learning: The Basics". Springer.

[3] Kubat, M. (2021). "An Introduction to Machine Learning". Springer.

[4] Li, H. (2023). "Machine Learning Methods". Springer.

[5] Zollanvari, A. (2023). " Machine Learning with Python". Springer.

[6] Chopra, D., & Khurana, R. (2023). "Introduction to Machine Learning with Python". Bentham Science Publishers.

[7] Müller, A. C., & Guido, S. (2016). "Introduction to Machine Learning with Python: A Guide for Data Scientists". O'Reilly Media.

[8] Raschka, S. (2015). "Python Machine Learning". Packt Publishing.

[9] Forsyth, D. (2019). "Applied Machine Learning". Springer.

[10] Sarkar, D., Bali, R., & Sharma, T. (2018). "Practical Machine Learning with Python". Apress.

[11] Bonaccorso, G. (2018). "Machine Learning Algorithms: Popular Algorithms for Data Science and Machine Learning". Packt Publishing.

[12] Teoh, T., & Rong, Z. (2022). "Artificial Intelligence with Python". Springer.

[13] Igual, L., & Seguí, S. (2017). "Introduction to Data Science: A Python Approach to Concepts, Techniques and Applications". Springer.

[14] VanderPlas, J. (2017). "Python Data Science Handbook: Essential Tools for Working with Data". O'Reilly Media.

[15] Muddana, A., & Vinayakam, S. (2024). "Python for Data Science". Springer.

[16] Unpingco, J. (2022). "Python for Probability, Statistics, and Machine Learning". Springer.

[17] Unpingco, J. (2021). "Python Programming for Data Analysis". Springer.

[18] Blazewicz, J., Kubiak, W., Morzy, T., & Rusinkiewicz, M. (2003). "Handbook on Data Management in Information Systems". Springer.

[19] Purba, S. (2019). "Handbook of Data Management". CRC Press.

[20] Gray, J. (1996). "Data Management: Past, Present, and Future". IEEE Computer 29(10), 38-46.

[21] Gordon, K. (2022). "Principles of Data Management: Facilitating Information Sharing". BCS.

[22] Bressoud, T., & White, D. (2020). "Introduction to Data Systems: Building from Python". Springer.

[23] Cao, J. (2023). "E-Commerce Big Data Mining and Analytics". Springer.

[24] Zelle, J. (2017). "Python Programming: An Introduction to Computer Science". Franklin, Beedle & Associates.

[25] Xanthidis, D., Manolas, C., Xanthidou, O. K., & Wang, H. I. (2022). "Handbook of Computer Programming with Python". CRC Press.

[26] Chun, W. (2001). "Core Python Programming". Prentice Hall Professional.

[27] Padmanabhan, T. (2016). "Programming with Python". Springer.

[28] Beazley, D., & Jones, B. K. (2013). "Python Cookbook: Recipes for Mastering Python 3". O'Reilly Media.

[29] Lott, S. (2014). "Mastering Object-Oriented Python". Packt Publishing.

[30] Phillips, D. (2015). "Python 3 Object-Oriented Programming: Harness the Power of Python 3 Objects". Packt Publishing.

[31] Lott, S., & Phillips, D. (2021). "Python Object-Oriented Programming: Build Robust and Maintainable Object-Oriented Python Applications and Libraries". Packt Publishing.

[32] Goldwasser, M. H., & Letscher, D. (2008). "Object-Oriented Programming in Python". Pearson Prentice Hall.

[33] Downey, A. (2012). "Think Python". O'Reilly Media

[34] Lutz, M. (2013). "Learning Python: Powerful Object-Oriented Programming". O'Reilly Media.

[35] Rangisetti, A. (2024). "Hands-On Object-Oriented Programming: Mastering OOP Features for Real-World Software Systems Development". Apress.

[36] Hillar, G. (2015). "Learning Object-Oriented Programming". Packt Publishing.

[37] Codd, E. (1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM. 13 (6), 377–87.

[38] Chamberlin, D., Boyce, R. (1974). "SEQUEL: A Structured English Query Language". Proceedings of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control, 249–64.

[39] Python: http://www.python.org

[40] Numpy: http://www.numpy.org

[41] Pandas: http://pandas.pydata.org

[42] Matplotlib: http://www. matplotlib.org

[43] Seaborn: http://seaborn.pydata.org

[44] NLTK: http://www.nltk.org

[45] SciPy: http://scipy.org

[46] SK Learn: http://scikit-learn.org