PYTHON FUNDAMENTALS: A COMPREHENSIVE TRAINING

Advanced data manipulation with Numpy/Pandas

Python Course -By Binatna Data-¶

- **Sessions of December 2023**
- **Session of 16/12/2023**

1. Initialisation:

➔ List in python:

- A list is a data structure in Python that is a mutable, or changeable, ordered sequence of elements. Each element or value that is inside of a list is called an item.

```
mylist = ["apple", "banana", "orange"]
mylist
```

```
['apple', 'banana', 'orange']
```

```
mylist = [1,2,3,"apple"]
mylist
```

```
[1, 2, 3, 'apple']
```

➔ Dictionary in python:

- Dictionaries are used to store data values in key:value pairs.
- A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

```
my_dict = {"Nom et prenom": "Hassan taroudant", "age": 20}

print(my_dict)
print(my_dict["Nom et prenom"])
print(my_dict["age"])
```

```
{'Nom et prenom': 'Hassan taroudant', 'age': 20}
Hassan taroudant
20
```

- The values in dictionary items can be of any data type.
- It is also possible to use the dict() constructor to make a dictionary.

```
thisdict = dict(name = "John", age = 36, country = "Norway")
print(thisdict)
```

```
{'name': 'John', 'age': 36, 'country': 'Norway'}
```

```python
# creation d'un dictionnaire vide
students = {}
print(students)
print()

# Add
students["g14"]={"Nom et prenom": "Hassan taroudant", "age": 20}
print("Apres l'ajout")
print(students)
print()

# update
students["g14"]={"Nom et prenom": "Yassine Casa", "age": 23} # students["g14"].update()
print("Apres la modification")
print(students)
print()

# Select element
print(students["g14"])
print()

# Initialze
students = {
    "G14": {"Nom et prenom": "Hassan taroudant", "age": 20},
    "G15": {"Nom et prenom": "Yassine Casa", "age": 23},
    "G16": {"Nom et prenom": "Fatima benguerir", "age": 19}
}

print(students)
```

```
{}

Apres l'ajout
{'g14': {'Nom et prenom': 'Hassan taroudant', 'age': 20}}

Apres la modification
{'g14': {'Nom et prenom': 'Yassine Casa', 'age': 23}}

{'Nom et prenom': 'Yassine Casa', 'age': 23}

{'G14': {'Nom et prenom': 'Hassan taroudant', 'age': 20}, 'G15': {'Nom et prenom': 'Yassine Casa', 'age':
23}, 'G16': {'Nom et prenom': 'Fatima benguerir', 'age': 19}}
```

- Here are some methods that can be called from a dictionary object:
  - keys() : Returns a sequence of keys.
  - values() : Returns a sequence of values.
  - items() : Returns a sequence of tuples (key, value).
  - clear() : Deletes all items.
  - get(key): Returns the value of the key.
  - pop(key): Deletes the element from the key and returns its value.
  - popitem() : tuple Returns a randomly selected key/value pair as a tuple and deletes the selected element.

➔ Range(), list(), len() and Type() functions:

- The range() function generates an arithmetic sequence. Its result is converted into a list using the list() function.

```
range(10)
```

```
range(0, 10)
```

```
list(range(10))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
list(range(5, 10, 2))
```

```
[5, 7, 9]
```

- The len() function returns the number of elements.

```
a = list(range(1,10,2))
print(a)
len(a)
```

```
[1, 3, 5, 7, 9]
5
```

- To access an element of a list, we enter the element index between square brackets [].
- The index of the first element of a list is 0.

```
a[3]
```

```
7
```

- To find out the type of an item or the type of a value, simply use the type() function.

```
print(type(15))
print(type("toto"))
print(type(1.2))
```

```
<class 'int'>
<class 'str'>
<class 'float'>
```

➔ Operation in lists

- It is possible to concatenate lists with +.

```
a=[1,3,5]
b=[0,2,4]
c = a+b
print(c)
```

```
[1, 3, 5, 0, 2, 4]
```

- you can add an element to the list in two ways: by using the append() function, or by using the insert() function.

- Insert() takes 2 arguments, the first is the index of the location where you want to add the element, and the second is the value of the element to be added.

- You can remove an element from a list in two ways: using the remove() function, which takes the element to be removed as a parameter, and the pop() function, which takes the index of the element to be removed.

```
a=[1,3,5]

# Add using append
a.append(6)
print(a)

# Add using insert
a.insert(0, 4)
print(a)

# Remove the given element
a.remove(4)
print(a)

# Remove using index
a.pop()
print(a)
```

```
[1, 3, 5, 6]
[4, 1, 3, 5, 6]
[1, 3, 5, 6]
[1, 3, 5]
```

➔ Exercices:

- Create a Python program that performs the following operations on a list of numbers:

    - ❖ Create an empty list called numbers.
    - ❖ Append the numbers 1 to 10 (inclusive) to the list.
    - ❖ Print the original list.
    - ❖ Print the length of the list.
    - ❖ Print the sum of all the numbers in the list (use the sum() function).
    - ❖ Remove the last element from the list.
    - ❖ Print the modified list.
    - ❖ Check if the number 5 is in the list and print the result (use the “in” keyword)
    - ❖ Double the value of each element in the list.
    - ❖ Print the final modified list.

- Create a Python program that performs the following operations on a dictionary representing a person's information:

    - ❖ Create a dictionary called person with keys: 'name', 'age', 'city', and 'country'. Assign appropriate values.
    - ❖ Print the original dictionary.
    - ❖ Add a new key-value pair: 'occupation' with a value representing the person's job.
    - ❖ Print the modified dictionary.
    - ❖ Check if the key 'gender' is in the dictionary and print the result.
    - ❖ Update the value of 'age' in the dictionary.
    - ❖ Print the final modified dictionary.

2. Numpy :

➔ Why numpy ?

- Numpy is a Python package specializing in the manipulation of arrays, for us essentially vectors and matrices.

- Numpy" arrays only handle objects of the same type

- The package offers a large number of routines for rapid access to data (e.g. search, extraction, etc.), for various manipulations (e.g. sorting), for scientific calculations (e.g. statistical calculation, numerical calculation, etc.). calculation, ...)

- Numpy arrays are more powerful (speed, volume management) than the usual Python collections.

➔ Install and Import

To use NumPy, you should install the package via this command: `pip install numpy` .

Once the installation is complete, we can import the package as follows:

```
import numpy as np

mylist = [2,3,4,5]

numpy_list = np.array(mylist)
numpy_list
```

```
array([2, 3, 4, 5])
```

➔ Creating NumPy arrays:

- np.array: Create an array from a Python list or tuple.
- np.arange: Generate an array with regularly spaced values.
- np.linspace: Generate an array with a specified number of evenly spaced values over a specified range.
- np.ones: Create an array filled with ones.
- np.full: Create an array with a specified shape and fill value.

- np.zeros: Create an array filled with zeros.
- np.eye: Create a 2-D identity matrix.

```python
import numpy as np

# Creating arrays
arr_from_list = np.array([1, 2, 3])
arr_range = np.arange(0, 10, 2)  # Start from 0, end at 10 (exclusive), step by 2
arr_linspace = np.linspace(0, 1, 5)  # 5 evenly spaced values from 0 to 1
arr_ones = np.ones((2, 3))  # 2x3 array of ones
arr_full = np.full((3, 3), 7)  # 3x3 array filled with 7
arr_zeros = np.zeros((2, 2))  # 2x2 array of zeros
arr_eye = np.eye(3)  # 3x3 identity matrix

# Displaying arrays
display("Array from list:", arr_from_list)
display("Array range:", arr_range)
display("Array linspace:", arr_linspace)
display("Array of ones:", arr_ones)
display("Array filled with 7:", arr_full)
display("Array of zeros:", arr_zeros)
display("Identity matrix:", arr_eye)
```

```
'Array from list:'
array([1, 2, 3])
'Array range:'
array([0, 2, 4, 6, 8])
'Array linspace:'
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
'Array of ones:'
array([[1., 1., 1.],
       [1., 1., 1.]])
'Array filled with 7:'
array([[7, 7, 7],
       [7, 7, 7],
       [7, 7, 7]])
'Array of zeros:'
array([[0., 0.],
       [0., 0.]])
'Identity matrix:'
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

➔ Array attributes (shape, size, dtype)

Arrays have attributes that provide information about their structure and content:

- shape: Tuple representing the array dimensions.
- size: Total number of elements in the array.
- dtype: Data type of the array elements.

```
print("Shape:", arr_range.shape)
print("Size:", arr_linspace.size)
print("Data type:", arr_ones.dtype)
```

```
Shape: (5,)
Size: 5
Data type: float64
```

➔ Accessing elements and subarrays

Accessing individual elements or subarrays in a NumPy array can be done using indexing and slicing.

```
arr_from_list = np.array([1, 2, 3])
arr_range = np.arange(0, 10, 2)

print("Element at index 2:", arr_from_list[2])

# Slicing subarrays
subarray = arr_range[1:4]  # Elements at index 1, 2, 3
print("Subarray:", subarray)
```

```
Element at index 2: 3
Subarray: [2 4 6]
```

➔ Reshaping arrays

Changing the shape of an array without changing its data. The reshape method is used for this.

```
arr_range = np.arange(2, 10, 2)
print(arr_range)

reshaped_arr = arr_range.reshape((2, 2))  # Reshape to a 2x2 array
display("Reshaped array:", reshaped_arr)
```

```
[2 4 6 8]
'Reshaped array:'
array([[2, 4],
       [6, 8]])
```

➔ Appending and inserting elements

Adding elements to an array using np.append or np.insert.

```
arr_from_list = np.array([1, 2, 3])

# Appending and inserting elements
arr_appended = np.append(arr_from_list, 4)
arr_inserted = np.insert(arr_from_list, 1, [0.5, 1.5])  # Insert elements at index 1

print("Appended array:", arr_appended)
print("Inserted array:", arr_inserted)
```

```
Appended array: [1 2 3 4]
Inserted array: [1 0 1 2 3]
```

➔ Deleting elements

Removing elements from an array using np.delete.

```
arr_range = np.arange(2, 10, 2)
print(arr_range)

arr_deleted = np.delete(arr_range, [0, 2])  # Delete elements at indices 0 and 2
print("\nArray after deletion:", arr_deleted)
```

```
[2 4 6 8]

Array after deletion: [4 8]
```

➔ Slicing and indexing techniques

Various techniques for slicing and indexing arrays based on conditions or intervals.

```
arr_range = np.arange(1, 10, 2)
print(arr_range)

# Slicing and indexing techniques
arr_slice_condition = arr_range[arr_range > 5]  # Elements greater than 5
arr_slice_interval = arr_range[1:4]  # Elements at index 1, 2, 3

print("\nSlice by condition:", arr_slice_condition)
print("Slice by interval:", arr_slice_interval)
```

```
[1 3 5 7 9]

Slice by condition: [7 9]
Slice by interval: [3 5 7]
```

➔ Concatenating arrays

Combining multiple arrays along an existing axis.

```
arr_range = np.arange(1, 10, 2)
arr_linspace = np.linspace(0, 1, 5)
print(arr_range)
print(arr_linspace)

# Concatenating arrays
arr_concatenated = np.concatenate([arr_range, arr_linspace])
print("Concatenated array:", arr_concatenated)
```

```
[1 3 5 7 9]
[0.   0.25 0.5  0.75 1.  ]
Concatenated array: [1.   3.   5.   7.   9.   0.   0.25 0.5  0.75 1.  ]
```

➔ Random sampling from arrays

Generating random samples from arrays using functions like np.random.choice.

```
arr_range = np.arange(1, 10, 2)
print(arr_range)

# Random sampling from arrays
random_samples = np.random.choice(arr_range, size=3)
print("Random samples:", random_samples)
```

```
[1 3 5 7 9]
Random samples: [9 3 9]
```

➔ Loading and saving data using NumPy

NumPy provides functions to read and write data from/to files, such as CSV, TXT, and other common formats.

```
arr_range = np.arange(1, 10, 2)
print(arr_range)

np.savetxt('my_array.txt', arr_range, delimiter=',')  # Save array to text file

loaded_array = np.loadtxt('my_array.txt', delimiter=',')  # Load array from text file

print("Loaded array:", loaded_array)
```

```
[1 3 5 7 9]
Loaded array: [1. 3. 5. 7. 9.]
```

➔ Statistics functions

```
v = np.array([1.2,7.4,4.2,8.5,6.3])


print(np.mean(v)) # 5.52

print(np.median(v)) # 6.3

print(np.var(v)) # 6.6856

print(np.percentile(v,50)) #6.3 (50\% = médiane)

print(np.sum(v)) # 27.6

print(np.cumsum(v)) # [1.2 8.6 12.8 21.3 27.6]
```

```
5.5200000000000005
6.3
6.685599999999999
6.3
27.6
[ 1.2  8.6 12.8 21.3 27.6]
```

➔ Exercice :

1. Create a NumPy array (arr1) with values [1, 2, 3, 4, 5].
2. Create a NumPy array (arr2) with values [10, 20, 30, 40, 50].
3. Perform element-wise multiplication of arr1 and arr2. Save the result in a new array (result_mult).
4. Create a 3x3 identity matrix (identity_matrix).
5. Check the shape, size, and data type of arr1.
6. Reshape arr2 into a 5x1 matrix and name it reshaped_arr2.
7. Append a new element 6 to the end of arr1 and name the new array arr1_appended.
8. Insert a new element 25 at index 2 in arr2 and name the new array arr2_inserted.
9. Delete the element at index 3 in arr1 and name the new array arr1_deleted.
10. Delete the first row in identity_matrix and name the new array identity_matrix_deleted.
11. Slice arr1 to get elements greater than 2 and name the result arr1_slice_condition.
12. Slice arr2 to get elements between index 1 and 3 (inclusive) and name the result arr2_slice_interval.
13. Concatenate arr1 and arr2 along a new axis and name the result concatenated_arrays.
14. Generate an array (random_array) of 10 random integers between 0 and 100.

15. Sample 3 random elements from random_array without replacement.
16. Save random_array to a CSV file named "random_data.csv".
17. Load the data from "random_data.csv" into a new array (loaded_array).
18. Calculate the sum, cumulative sum, and mean of loaded_array.
19. Calculate the mean of each column in identity_matrix_deleted.
20. Create an array called results containing the variables: sum_arr1, arr1_mean as elements, then save the result to a csv file named “results.csv”.

3. Pandas :

The pandas module was designed for data manipulation and analysis. It is particularly powerful for handling structured data in table form.
To import pandas,we use the usual import command: “import pandas”

➔ Series

A Series is a one-dimensional labeled array capable of holding any data type. It is similar to a column in a spreadsheet or a column in a SQL table.

```
import pandas as pd
import numpy as np

# Creating a Series from a list
data = [1, 3, 5, np.nan, 6, 8]
s = pd.Series(data)
print(s)
```

```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

➔ DataFrame

A DataFrame is a two-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled axes (rows and columns). It is similar to a spreadsheet or SQL table.

```python
# Creating a DataFrame from a dictionary
# first way
columns = ["Name", "Age", "City"]
index_list  = [0,1,2]
# data
row1 = ['John', 25, 'New York']
row2 = ['Alice', 28, 'San Francisco']
row3 = ['Bob', 22, 'Los Angeles']

df = pd.DataFrame(columns = columns, index = index_list, data = [row1,row2,row3])
print(df)
```

```
    Name  Age           City
0   John   25       New York
1  Alice   28  San Francisco
2    Bob   22    Los Angeles
```

```python
data = {'Name': ['John', 'Alice', 'Bob'],
        'Age': [25, 28, 22],
        'City': ['New York', 'San Francisco', 'Los Angeles']}
df = pd.DataFrame(data)
print(df)
```

```
    Name  Age           City
0   John   25       New York
1  Alice   28  San Francisco
2    Bob   22    Los Angeles
```

➔ Reading data from different sources (CSV, Excel, SQL, etc.)

Pandas provides functions to read data from various sources like CSV, Excel, SQL databases, etc.

```python
# Reading data from a CSV file
csv_data = pd.read_csv('Greenhouse_Gas_Emissions.csv')

# Reading data from an Excel file
excel_data = pd.read_excel('RuralAtlasData24.xlsx')

# Reading data from a SQL database
sql_data = pd.read_sql('SELECT * FROM table', connection)
```

➔ Displaying basic information (head, tail, info)

Pandas provides methods like head, tail, and info to display basic information about the DataFrame.

```python
# Reading data from a CSV file
df = pd.read_csv('Greenhouse_Gas_Emissions.csv')

# Displaying the first few rows
print("Head of the dataframe")
print(df.head())

# Displaying the last few rows
print("\n\n\nTail of the dataframe")
print(df.tail())

# Displaying information about the DataFrame
print("\n\n\nInfo about the dataframe")
print(df.info())
```

```
Head of the dataframe
  Month/Year  Facilities  Fleet  Total
0   Aug 2020        4617   3946   8563
1   Oct 2020        4337   4140   8477
2   Nov 2020        4681   4017   8686
3   Dec 2020        6195   4319  10514
4   Jan 2021        6011   4192  10203



Tail of the dataframe
   Month/Year  Facilities  Fleet  Total
43   Apr 2023        3843   3899   7742
44   May 2023        4013   3976   7989
45   Jun 2023        3882   4528   8410
46   Jul 2023        4427   4465   8892
47   Aug 2023        4118   4807   8925



Info about the dataframe
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48 entries, 0 to 47
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Month/Year  48 non-null     object
 1   Facilities  48 non-null     int64 
 2   Fleet       48 non-null     int64 
 3   Total       48 non-null     int64 
dtypes: int64(3), object(1)
memory usage: 1.4+ KB
None
```

➔ Descriptive statistics (describe)

The describe method provides descriptive statistics of the DataFrame, including measures like mean, standard deviation, min, max, etc.

```
print(df.describe())
```

```
        Facilities        Fleet         Total
count    48.000000    48.000000     48.000000
mean   4672.541667  4329.354167   8998.354167
std     737.594248   683.400960   1152.077485
min    3486.000000  2410.000000   6400.000000
25%    4101.000000  4064.000000   8308.000000
50%    4506.500000  4282.500000   8886.000000
75%    5160.000000  4639.250000   9886.500000
max    6284.000000  6891.000000  11267.000000
```

➔ Selecting columns and rows

Pandas allows selecting specific columns and rows using various methods, such as loc and iloc.

```
# Selecting a specific column
print(df['Facilities'][:5])

# Selecting specific rows
print("\n\n\n")
print(df.iloc[0])  # Selects the first row
```

```
0    4617
1    4337
2    4681
3    6195
4    6011
Name: Facilities, dtype: int64



Month/Year    Aug 2020
Facilities        4617
Fleet             3946
Total             8563
Name: 0, dtype: object
```

➔ Conditional selection

Conditional selection allows filtering data based on specific conditions.

```
# Selecting rows where Age is greater than 25
print(df[df['Total'] > 8998])
```

```
   Month/Year  Facilities  Fleet  Total
3    Dec 2020        6195   4319  10514
4    Jan 2021        6011   4192  10203
5    Feb 2021        5736   4170   9906
13   Nov 2021        4851   4584   9435
14   Dec 2021        5614   4354   9968
15   Jan 2022        6028   4596  10624
16   Mar 2022        4684   5196   9880
19   Oct 2019        4820   4183   9003
20   Dec 2019        6238   4976  11214
21   Jul 2019        5166   5133  10299
22   Nov 2019        5507   4838  10345
23   Aug 2019        5160   5237  10397
24   Sep 2019        4597   4742   9339
25   Jan 2020        6284   4983  11267
27   Feb 2020        5384   4486   9870
33   Jun 2022        4222   6891  11113
35   Aug 2022        4312   4884   9196
38   Nov 2022        4586   4658   9244
39   Dec 2022        5665   4423  10088
40   Jan 2023        5160   4246   9406
41   Feb 2023        5011   4370   9381
42   Mar 2023        4790   4633   9423
```

➔ Detecting and filling missing values

Pandas provides methods like isnull to detect missing values and fillna to fill them.

```
# Detecting missing values
print(df.isnull().sum())

# Filling missing values with a specific value
df.fillna(0, inplace=True)
```

```
Month/Year    0
Facilities    0
Fleet         0
Total         0
dtype: int64
```

➔ Dropping missing values

The dropna method is used to remove rows with missing values.

```
# Dropping rows with missing values
df.dropna(inplace=True, axis=0)
```

➔ Replacing values

'replace' method is used to replace specific values with other values.

```
# Replacing 'Dec 2020' with 'Dec 2023' in the 'Month/Year' column
df['Month/Year'].replace('Dec 2020', 'Dec 2023', inplace=True)
df
```

	Month/Year	Facilities	Fleet	Total
0	Aug 2020	4617	3946	8563
1	Oct 2020	4337	4140	8477
2	Nov 2020	4681	4017	8686
3	Dec 2023	6195	4319	10514
4	Jan 2021	6011	4192	10203
5	Feb 2021	5736	4170	9906
6	Mar 2021	4774	4106	8880
7	Apr 2021	4273	3978	8251
8	May 2021	3856	4049	7905
9	Jun 2021	4392	4146	8538
10	Jul 2021	4258	4069	8327
11	Sep 2021	4152	4223	8375
12	Oct 2021	3945	4221	8007
13	Nov 2021	4851	4584	9435
14	Dec 2021	5614	4354	9968

➔ Renaming columns

The ‘rename’ method is used to rename columns in a DataFrame.

```
# Renaming the 'Month/Year' column to 'Date'
df.rename(columns={'Month/Year': 'Date'}, inplace=True)
df
```

	Date	Facilities	Fleet	Total
0	Aug 2020	4617	3946	8563
1	Oct 2020	4337	4140	8477
2	Nov 2020	4681	4017	8686
3	Dec 2023	6195	4319	10514

➔ Grouping data with `groupby`

The ‘groupby’ method is used to group data based on one or more columns.

```
# Grouping data by 'Date' and calculating the mean of each group
grouped_data = df.groupby('Date').mean()
print(grouped_data)
```

```
          Facilities   Fleet    Total
Date
Apr 2020      3990.0  2421.5   6411.5
Apr 2021      4273.0  3978.0   8251.0
Apr 2022      4031.0  4004.0   8035.0
Apr 2023      3843.0  3899.0   7742.0
Aug 2019      5160.0  5237.0  10397.0
Aug 2020      4617.0  3946.0   8563.0
Aug 2022      4312.0  4884.0   9196.0
Aug 2023      4118.0  4807.0   8925.0
Dec 2019      6238.0  4976.0  11214.0
Dec 2021      5614.0  4354.0   9968.0
Dec 2022      5665.0  4423.0  10088.0
Dec 2023      6195.0  4319.0  10514.0
Feb 2020      5384.0  4486.0   9870.0
Feb 2021      5736.0  4170.0   9906.0
Feb 2023      5011.0  4370.0   9381.0
```

➔ Aggregating functions (sum, mean, count)

Aggregating functions like sum, mean, and count are used to perform operations on grouped data.

```
# Calculating the sum of Fleets for each Date
sum_fleet = df.groupby('Date')['Fleet'].sum()
sum_fleet
```

```
Date
Apr 2020    4843
Apr 2021    3978
Apr 2022    4004
Apr 2023    3899
Aug 2019    5237
Aug 2020    3946
Aug 2022    4884
Aug 2023    4807
Dec 2019    4976
Dec 2021    4354
Dec 2022    4423
Dec 2023    4319
```

➔ Combining DataFrames using merge and concat

Pandas provides merge and concat methods to combine DataFrames.

```
# Concatenating two DataFrames vertically
df_concatenated = pd.concat([df, grouped_data])

# Merging two DataFrames based on a common column
df_merged = pd.merge(df, grouped_data, on='Total')
```

```
len(df_concatenated)
```

```
95
```

➔ Exporting data as csv, excel ..

Pandas allows exporting data to various formats like CSV, Excel, SQL, etc.

```
# Exporting DataFrame to a CSV file
df.to_csv('output.csv', index=False)

# Exporting DataFrame to an Excel file
df.to_excel('output.xlsx', index=False)
```

➔ Exercise :

1. Develop a script that utilizes the pandas library to read the contents of the csv file named "solar_energy_data_by_binatnaData.csv" into a DataFrame, and assign it the variable name “df”.
2. Display the first and last few rows of the DataFrame 'data' using the 'head()' and 'tail()' functions, and the column names.
3. Rename the columns by the following names : A, E, I, T
4. Determine and print the count of missing (NaN) values for each column in the DataFrame.
5. Eliminate the rows containing NaN values from the DataFrame.
6. Check for duplicate rows in the DataFrame using the 'duplicated()' method.
7. Calculate the Energy for each row by applying the formula: Energy = Solar Panel Area * Solar Panel Efficiency * Solar Irradiance * Time.
8. Calculate the energy for each row by applying the same formula, this time using a function that takes 4 arguments: A (Area), E (Efficiency), I (Irradiance), T (Time), then the function returns the calculated Energy.
9. Introduce a new column in the DataFrame, naming it "Energy," and populate it with the calculated energy values.
10. Save the modified DataFrame with the additional "Energy" column to a new csv file.