Basic and Advanced Data Manipulation with NumPy

Author: Basel Amr Barakat
Email: baselamr52@gmail.com
Date: 2024-12-19
Task Name: Data Manipulation with Pandas and NumPy **Task Number:** 21
Part: Advanced Pandas
Module: Python Programming Language for AI / ML
Submit Number: 1

Description:

In this task, students will work with both Pandas and NumPy together. They will create and manipulate Data Frames using Pandas, handle missing data, merge/join Data Frames, and apply NumPy functions for efficient data analysis.

Requirements:

1. **Requirement 1: Creating and Manipulating Pandas DataFrames**
   - **Description:**
     Students will create DataFrames from lists, dictionaries, and NumPy arrays. They will also practice selecting and filtering data in DataFrames.
2. **Requirement 2: Handling Missing Data and Merging DataFrames**
   - **Description:**
     Students will learn techniques for handling missing data using .fillna() and .dropna(), and perform DataFrame merging and joining.
3. **Requirement 3: Combining Pandas and NumPy for Analysis* *
   - **Description:**
     Students will use NumPy functions to perform mathematical operations on Pandas DataFrames, such as aggregation, statistical analysis, and transforming data.

📝 1. Introduction & Objective

In this notebook, we will:

- Learn how to create and manipulate Pandas DataFrames.

- Handle missing data using `.fillna()` and `.dropna()`.

- Merge and join DataFrames efficiently.

- Utilize NumPy functions for mathematical operations on DataFrames.

🚀 2. Setting Up the Environment

Importing necessary libraries

```
import pandas as pd
import numpy as np
```

📌 3. Requirement 1: Creating and Manipulating Pandas DataFrames

Description : Students will create DataFrames from lists, dictionaries, and NumPy arrays. They will also practice selecting and filtering data in DataFrames.

3.1 Create DataFrame from Lists

```
# Creating a simple DataFrame from lists
data = [['Basel', 26], ['Omar', 25], ['Mohamed', 22],['Omar',26]]
data_frame = pd.DataFrame(data, columns=['Name', 'Age'])
data_frame.head(5)
```

	Name	Age
0	Basel	26
1	Omar	25
2	Mohamed	22
3	Omar	26

3.2 Create DataFrame from Dictionary

```
# Creating a DataFrame from a dictionary
data = {
    'Name': ['Basel', 'Omar', 'Mohamed','Abanoub','Aya'],
    'Age': [24, 27, 22,26,25],
    'City': ['Cairo', 'Assuit', 'Alexandria','Cairo','Cairo']
}
df_dict = pd.DataFrame(data)
print(df_dict)
```

```
      Name  Age        City
0    Basel   24       Cairo
1     Omar   27      Assuit
2  Mohamed   22  Alexandria
3  Abanoub   26       Cairo
4      Aya   25       Cairo
```

3.3 Create DataFrame from NumPy Array

```
# Creating a DataFrame from a NumPy array
array = np.random.rand(5, 3)             # --->  Build a random numpy array 5x3 (5 rows and 3 columns)
df_array = pd.DataFrame(array, columns=['Column1', 'Column2', 'Column3'])
print(df_array)
```

```
    Column1   Column2   Column3
0  0.979463  0.381122  0.353856
1  0.626203  0.680800  0.938583
2  0.180458  0.101420  0.934608
3  0.223734  0.747711  0.447831
4  0.197296  0.572358  0.355112
```

3.4 Data Selection and Filtering

```
# Selecting a single column
print("Filtering the names only of the dataframe")
print(df_dict['Name'])

# Filtering rows based on conditions
filtered_df = df_dict[df_dict['Age'] > 25]
print("Filtering the rows of age >25")
print(filtered_df)
```

```
Filtering the names only of the dataframe
0      Basel
1       Omar
2    Mohamed
3    Abanoub
4        Aya
Name: Name, dtype: object
```

```
Filtering the rows of age >25
      Name  Age    City
1     Omar   27  Assuit
3  Abanoub   26   Cairo
```

📌 4. Requirement 2: Handling Missing Data and Merging DataFrames

Description : Students will learn techniques for handling missing data using .fillna() and .dropna(), and perform DataFrame merging and joining.

4.1 Handling Missing Data

```
# Create a DataFrame with missing values
data = {
    'A': [1, 2, np.nan, 4,5,2,1,2,4,10],
    'B': [5, np.nan, 7, 8,np.nan,1,3,2,2,1],
    'C': [9, 10, 11, np.nan,np.nan,2,1,3,5,np.nan],
    'D':[1,2,3,4,5,6,7,8,9,10],
    'E':[11,22,31,41,51,61,71,81,91,101],
    'F':([1,2,3,4,5,6,7,8,9,10]),
    'G':[11,22,np.nan,41,51,np.nan,71,81,91,101]
}

df_missing = pd.DataFrame(data)
print("Original DataFrame with missing values:")
print(df_missing)

# Fill missing values
df_filled = df_missing.fillna(0,inplace=False)
print("After filling missing values:")
print(df_filled)

# Drop missing values
df_dropped = df_missing.dropna(axis=0,inplace=False)
print("After dropping rows with missing values:")
print(df_dropped)

df_dropped = df_missing.dropna(axis=1,inplace=False)
print("After dropping columns with missing values:")
print(df_dropped)
```

```
Original DataFrame with missing values:
      A    B     C   D    E   F      G
0   1.0  5.0   9.0   1   11   1   11.0
1   2.0  NaN  10.0   2   22   2   22.0
2   NaN  7.0  11.0   3   31   3    NaN
3   4.0  8.0   NaN   4   41   4   41.0
4   5.0  NaN   NaN   5   51   5   51.0
5   2.0  1.0   2.0   6   61   6    NaN
6   1.0  3.0   1.0   7   71   7   71.0
7   2.0  2.0   3.0   8   81   8   81.0
8   4.0  2.0   5.0   9   91   9   91.0
9  10.0  1.0   NaN  10  101  10  101.0
After filling missing values:
      A    B     C   D    E   F      G
0   1.0  5.0   9.0   1   11   1   11.0
1   2.0  0.0  10.0   2   22   2   22.0
2   0.0  7.0  11.0   3   31   3    0.0
3   4.0  8.0   0.0   4   41   4   41.0
4   5.0  0.0   0.0   5   51   5   51.0
5   2.0  1.0   2.0   6   61   6    0.0
6   1.0  3.0   1.0   7   71   7   71.0
7   2.0  2.0   3.0   8   81   8   81.0
8   4.0  2.0   5.0   9   91   9   91.0
9  10.0  1.0   0.0  10  101  10  101.0
After dropping rows with missing values:
     A    B    C  D   E  F     G
0  1.0  5.0  9.0  1  11  1  11.0
6  1.0  3.0  1.0  7  71  7  71.0
7  2.0  2.0  3.0  8  81  8  81.0
8  4.0  2.0  5.0  9  91  9  91.0
After dropping columns with missing values:
    D    E   F
0   1   11   1
1   2   22   2
2   3   31   3
3   4   41   4
```

```
4   5   51   5
5   6   61   6
6   7   71   7
7   8   81   8
8   9   91   9
9  10  101  10
```

4.2 Merging DataFrames

```python
# Create two DataFrames for merging
data1 = {'ID': [1, 2, 3], 'Name': ['Basel', 'Amr', 'Barakat']}
data2 = {'ID': [1, 2, 4], 'Score': [85, 90, 95]}
df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)
print("Before Merging the two dataframes")
print(df1)
print(df2)
# Merging on ID
merged_df = pd.merge(df1, df2, on='ID', how='inner')
print("Merged DataFrame:")
print(merged_df)
print("We found that the two dataframes are merged together depending on col `ID` and that the ID 3 is not in the second dataframe")
print("We found that the two dataframes are merged together depending on col `ID` and that the ID 4 is not in the first dataframe")
```

```
Before Merging the two dataframes
   ID     Name
0   1    Basel
1   2      Amr
2   3  Barakat
   ID  Score
0   1     85
1   2     90
2   4     95
Merged DataFrame:
   ID   Name  Score
0   1  Basel     85
1   2    Amr     90
We found that the two dataframes are merged together depending on col `ID` and that the ID 3 is not in the second dataframe
We found that the two dataframes are merged together depending on col `ID` and that the ID 4 is not in the first dataframe
```

📌 5. Requirement 3: Combining Pandas and NumPy for Analysis

Description : Students will use NumPy functions to perform mathematical operations on Pandas DataFrames, such as aggregation, statistical analysis, and transforming data.

5.1 Aggregations and Statistical Analysis

```python
# Creating a Sample DataFrame for Analysis
data_analysis = {
    'Product': ['A', 'B', 'C', 'D'],
    'Sales_Price': [1000, 2500, 30000, 4000],
    'Cost': [50, 80, 120, 150]
}
df_analysis = pd.DataFrame(data_analysis)
print("\nAnalysis DataFrame:\n", df_analysis)

# Using NumPy for Mathematical Operations
df_analysis['Profit'] = np.array(df_analysis['Sales_Price']) - np.array(df_analysis['Cost'])
print("\nDataFrame with Profit Calculated:\n", df_analysis)

# Aggregation and Statistical Analysis
mean_sales = np.mean(df_analysis['Sales_Price'])
print("\nMean Sales:", mean_sales)

total_cost = np.sum(df_analysis['Cost'])
print("Total Cost:", total_cost)

max_profit = np.max(df_analysis['Profit'])
print("Maximum Profit:", max_profit)

min_profit = np.min(df_analysis['Profit'])
```

```
print("Minimum Profit:", min_profit)

median_profit = np.median(df_analysis['Profit'])
print("Median Profit:", median_profit)

std_profit = np.std(df_analysis['Profit'])
print("Standard Deviation of Profit:", std_profit)

df_analysis.describe()
print("We have noticed that the std equation of NumPy is different from the equation of describe() ")
```

```
Analysis DataFrame:
   Product  Sales_Price  Cost
0       A         1000    50
1       B         2500    80
2       C        30000   120
3       D         4000   150

DataFrame with Profit Calculated:
   Product  Sales_Price  Cost  Profit
0       A         1000    50     950
1       B         2500    80    2420
2       C        30000   120   29880
3       D         4000   150    3850

Mean Sales: 9375.0
Total Cost: 400
Maximum Profit: 29880
Minimum Profit: 950
Median Profit: 3135.0
Standard Deviation of Profit: 11940.407237611287
We have noticed that the std equation of NumPy is different from the equation of describe()
```

5.2 Data Transformation

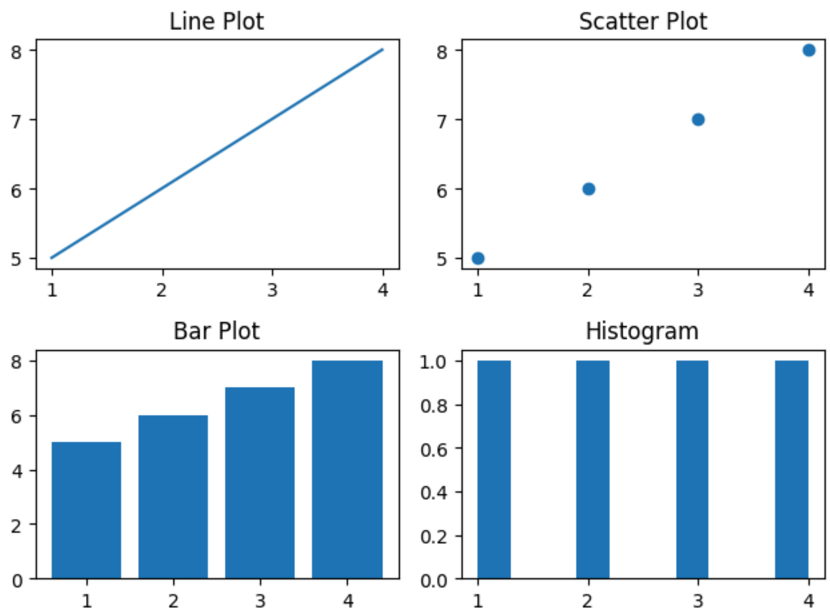
```
# Apply NumPy transformations
data_analysis = {
    'Sales': [1000, 2500, 30000, 4000],
    'Cost': [50, 80, 120, 150]
}
df_analysis = pd.DataFrame(data_analysis)
df_transformed_log = df_analysis.apply(np.log)
print("Log Transformed DataFrame:")
print(df_transformed_log)

df_transformed_sqrt = df_analysis.apply(np.sqrt)
print("Square Root Transformed DataFrame:")
print(df_transformed_sqrt)

df_tansformed_exp = df_analysis.apply(np.exp)
print("Exponential Transformed DataFrame:")
print(df_tansformed_exp)

df_transformed_sin = df_analysis.apply(np.sin)
print("Sin Transformed DataFrame:")
print(df_transformed_sin)

df_transformed_cos = df_analysis.apply(np.cos)
print("Cos Transformed DataFrame:")
print(df_transformed_cos)

df_transformed_tan = df_analysis.apply(np.tan)
print("Tan Transformed DataFrame:")
print(df_transformed_tan)
```

```
Log Transformed DataFrame:
       Sales      Cost
0   6.907755  3.912023
1   7.824046  4.382027
2  10.308953  4.787492
3   8.294050  5.010635
Square Root Transformed DataFrame:
        Sales       Cost
0   31.622777   7.071068
1   50.000000   8.944272
2  173.205081  10.954451
3   63.245553  12.247449
```

```
Exponential Transformed DataFrame:
   Sales          Cost
0    inf  5.184706e+21
1    inf  5.540622e+34
2    inf  1.304181e+52
3    inf  1.393710e+65
Sin Transformed DataFrame:
      Sales      Cost
0  0.826880 -0.262375
1 -0.650128 -0.993889
2 -0.802665  0.580611
3 -0.683504 -0.714876
Cos Transformed DataFrame:
      Sales      Cost
0  0.562379  0.964966
1  0.759825 -0.110387
2 -0.596430  0.814181
3 -0.729947  0.699251
Tan Transformed DataFrame:
      Sales      Cost
0  1.470324 -0.271901
1 -0.855628  9.003655
2  1.345784  0.713123
3  0.936375 -1.022346
```

📌 6. Requirement 4: Plotting

```python
# Plot data
import matplotlib.pyplot as plt
data = {'A': [1, 2, 3, 4], 'B': [5, 6, 7, 8]}
df_analysis = pd.DataFrame(data)

plt.subplot(2, 2, 1)
plt.plot(df_analysis['A'], df_analysis['B'])
plt.title('Line Plot')

plt.subplot(2, 2, 2)
plt.scatter(df_analysis['A'], df_analysis['B'])
plt.title('Scatter Plot')

plt.subplot(2, 2, 3)
plt.bar(df_analysis['A'], df_analysis['B'])
plt.title('Bar Plot')

plt.subplot(2, 2, 4)
plt.hist(df_analysis['A'])
plt.title('Histogram')

plt.tight_layout()
plt.show()
```



🎓 7. Conclusion

- We created DataFrames from multiple sources.
- Handled missing data effectively.
- Performed merging and analysis using Pandas and NumPy.
- Added visualizations for better insights.

📚 8. References

- Pandas Documentation: https://pandas.pydata.org/docs/
- NumPy Documentation: https://numpy.org/doc/
- Matplotlib Documentation: https://matplotlib.org/stable/index.html