

Data Mining and Data Warehousing Lab

CSEL-4108

Assignment on Data Analysis Techniques Using Pandas – 2

Submitted By

NISHAT MAHMUD
ID: B190305003

MD. WALIUL ISLAM RAYHAN
ID: B190305034

Submitted To

MD. MANOWARUL ISLAM, PHD
Associate Professor
Dept. of CSE
Jagannath University, Dhaka - 1100

This assignment explores various data analysis techniques using the Pandas library in Python. Below are detailed explanations, code examples, and expected outputs for different operations.

1. Finding Maximum Values

Method: `df.max()`

This method finds the maximum value for each column in a DataFrame.



The screenshot shows a Jupyter Notebook interface with a code cell and its output. The code cell contains the following Python code:

```
import pandas as pd
data = {'Student': ['Arif', 'Tania', 'Nabil'], 'Physics': [68, 74, 85], 'Chemistry': [72, 69, 88]}
df = pd.DataFrame(data)
print(df.max())
```

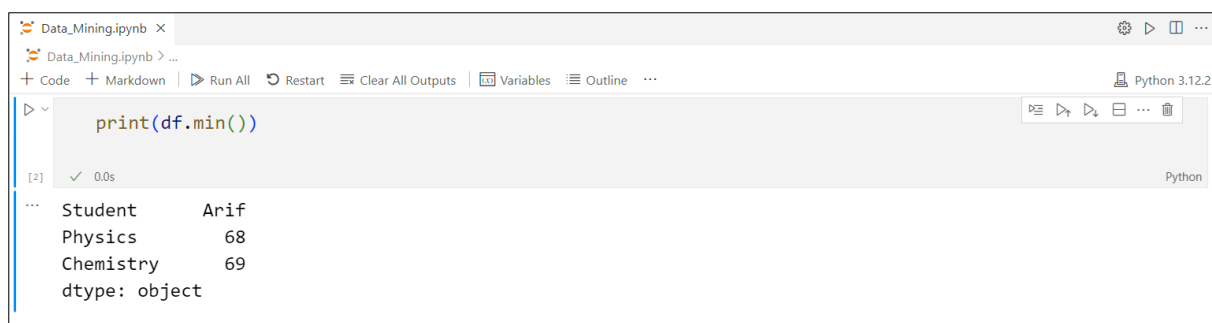
The output cell shows the result of the `df.max()` operation:

```
[1] ✓ 1.4s Python
*** Student      Tania
    Physics      85
    Chemistry    88
    dtype: object
```

2. Finding Minimum Values

Method: `df.min()`

This method finds the minimum value for each column in a DataFrame.



The screenshot shows a Jupyter Notebook interface with a code cell and its output. The code cell contains the following Python code:

```
print(df.min())
```

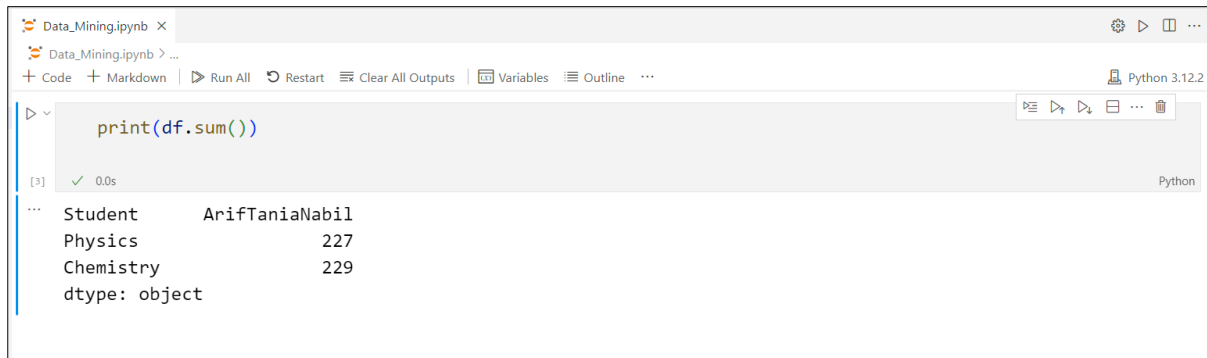
The output cell shows the result of the `df.min()` operation:

```
[2] ✓ 0.0s Python
*** Student      Arif
    Physics      68
    Chemistry    69
    dtype: object
```

3. Summing Values

Method: `df.sum()`

This method adds up the values for each column in a DataFrame. For non-numeric columns, it concatenates the values.



A screenshot of a Jupyter Notebook interface. The top bar shows the file name 'Data_Mining.ipynb' and various controls like 'Run All', 'Restart', and 'Clear All Outputs'. The code cell contains `print(df.sum())`. The output shows the sum for each column: 'Student' is 'ArifTaniaNabil', 'Physics' is 227, and 'Chemistry' is 229. The dtype is 'object'.

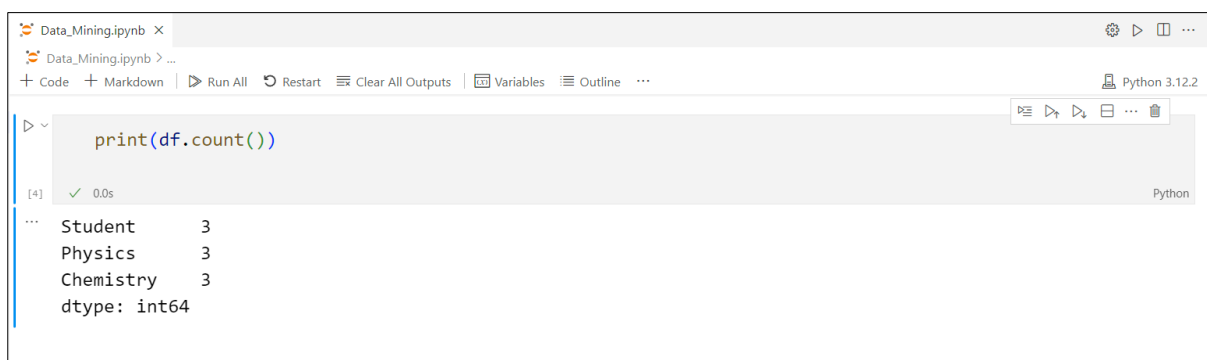
```
print(df.sum())
```

```
Student      ArifTaniaNabil
Physics              227
Chemistry             229
dtype: object
```

4. Counting Values

Method: `df.count()`

This method counts the number of non-null values in each column.



A screenshot of a Jupyter Notebook interface. The code cell contains `print(df.count())`. The output shows the count of non-null values for each column: 'Student', 'Physics', and 'Chemistry' all have a count of 3. The dtype is 'int64'.

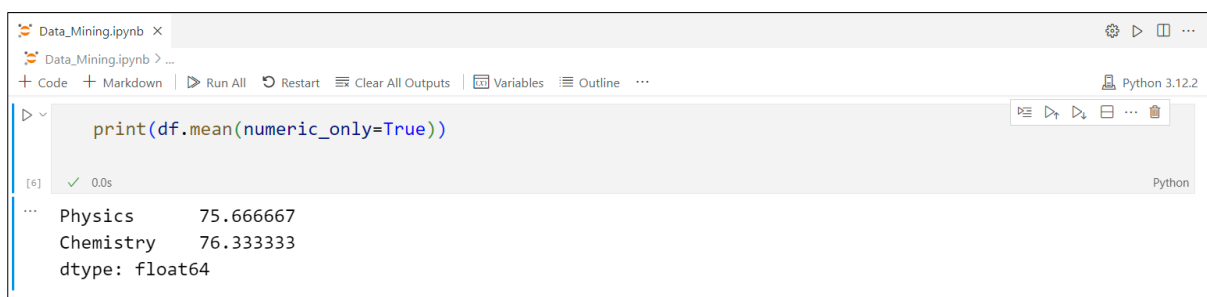
```
print(df.count())
```

```
Student      3
Physics      3
Chemistry    3
dtype: int64
```

5. Calculating Mean

Method: `df.mean()`

This method calculates the mean (average) of numeric values for each column.



A screenshot of a Jupyter Notebook interface. The code cell contains `print(df.mean(numeric_only=True))`. The output shows the mean for the numeric columns: 'Physics' is 75.666667 and 'Chemistry' is 76.333333. The dtype is 'float64'.

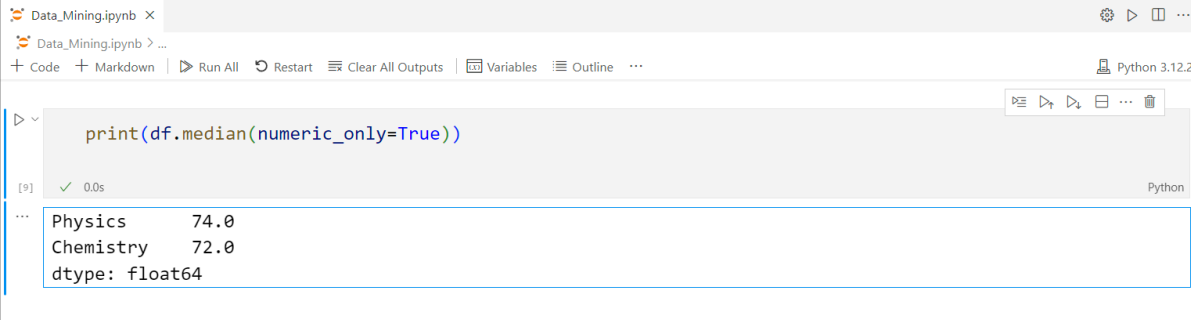
```
print(df.mean(numeric_only=True))
```

```
Physics      75.666667
Chemistry    76.333333
dtype: float64
```

6. Calculating Median

Method: `df.median()`

This method finds the median value for each column in a DataFrame.



The screenshot shows a Jupyter Notebook interface with a single code cell. The code cell contains the command `print(df.median(numeric_only=True))`. Below the code cell, the output is displayed as a text box containing the median values for the 'Physics' and 'Chemistry' columns, both being 74.0 and 72.0 respectively, with a data type of float64.

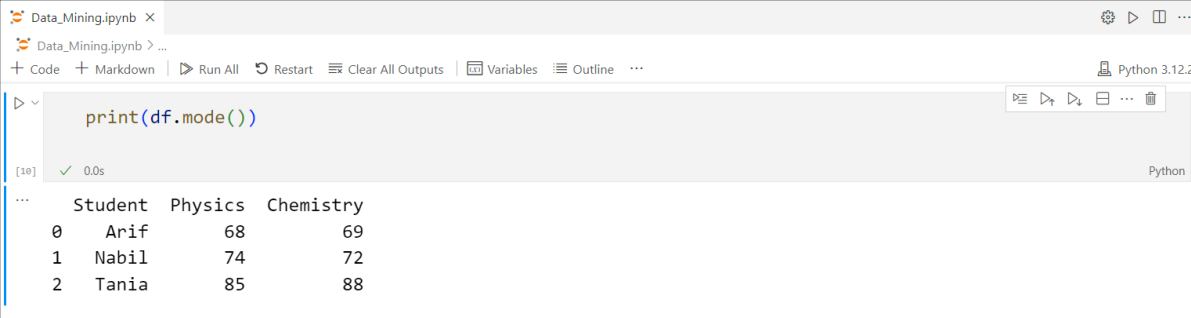
```
print(df.median(numeric_only=True))
```

```
Physics    74.0
Chemistry   72.0
dtype: float64
```

7. Calculating Mode

Method: `df.mode()`

This method finds the mode (most frequent value) for each column.



The screenshot shows a Jupyter Notebook interface with a single code cell. The code cell contains the command `print(df.mode())`. Below the code cell, the output is displayed as a text box containing the mode for each column. The 'Student' column has three unique values (Arif, Nabil, Tania) and the 'Physics' and 'Chemistry' columns have three unique values (68, 74, 85 and 69, 72, 88 respectively).

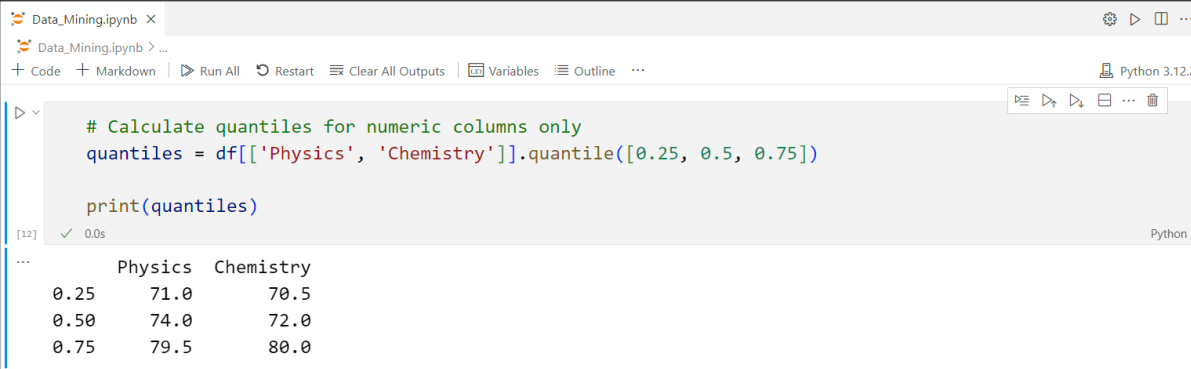
```
print(df.mode())
```

```
   Student  Physics  Chemistry
0    Arif      68      69
1   Nabil      74      72
2   Tania      85      88
```

8. Calculating Quartiles

Method: `df.quantile([0.25, 0.5, 0.75])`

This method calculates the quartiles of numeric values in each column.



The screenshot shows a Jupyter Notebook interface with a single code cell. The code cell contains two lines of code: a comment `# Calculate quartiles for numeric columns only` followed by `quantiles = df[['Physics', 'Chemistry']].quantile([0.25, 0.5, 0.75])` and `print(quantiles)`. Below the code cell, the output is displayed as a text box containing the quartiles for the 'Physics' and 'Chemistry' columns. The output shows the 0.25, 0.50, and 0.75 quartiles for each column.

```
# Calculate quartiles for numeric columns only
quantiles = df[['Physics', 'Chemistry']].quantile([0.25, 0.5, 0.75])

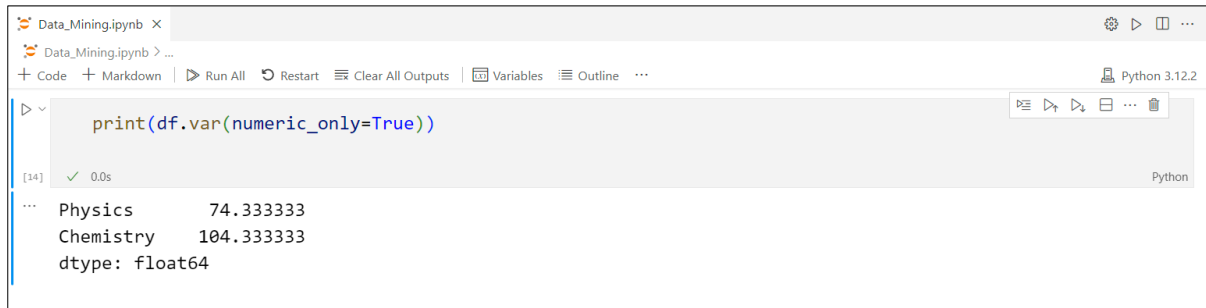
print(quantiles)
```

```
   Physics  Chemistry
0.25    71.0     70.5
0.50    74.0     72.0
0.75    79.5     80.0
```

9. Calculating Variance

Method: `df.var()`

This method calculates the variance of numeric values in each column.



A Jupyter Notebook interface with a single code cell. The code cell contains the command `print(df.var(numeric_only=True))`. Below the code cell, the output is displayed, showing the variance for 'Physics' (74.333333) and 'Chemistry' (104.333333), along with the data type `dtype: float64`.

```
print(df.var(numeric_only=True))
```

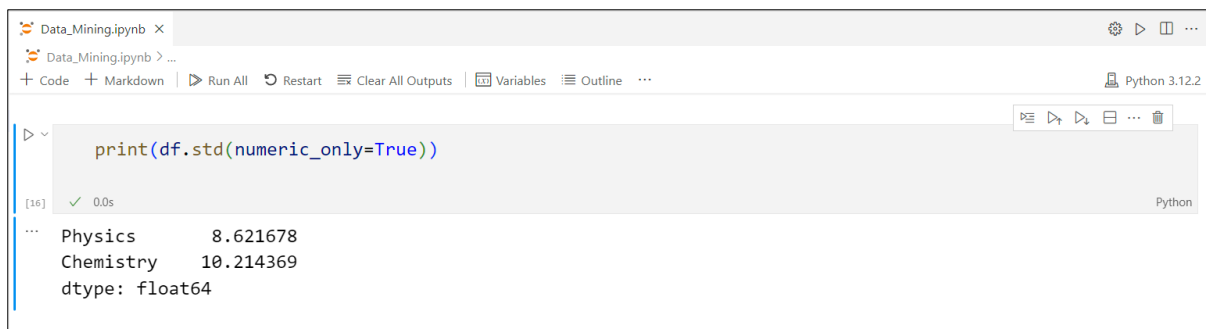
```
[14] ✓ 0.0s
```

```
Physics      74.333333
Chemistry    104.333333
dtype: float64
```

10. Calculating Standard Deviation

Method: `df.std()`

This method calculates the standard deviation of numeric values in each column.



A Jupyter Notebook interface with a single code cell. The code cell contains the command `print(df.std(numeric_only=True))`. Below the code cell, the output is displayed, showing the standard deviation for 'Physics' (8.621678) and 'Chemistry' (10.214369), along with the data type `dtype: float64`.

```
print(df.std(numeric_only=True))
```

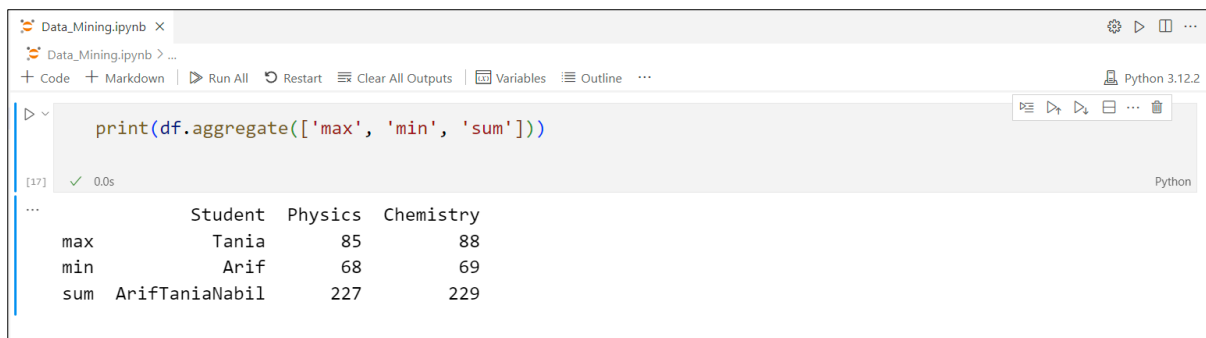
```
[16] ✓ 0.0s
```

```
Physics      8.621678
Chemistry    10.214369
dtype: float64
```

11. Performing Aggregation

Method: `df.aggregate(['max', 'min', 'sum'])`

This method applies multiple aggregation functions to the columns.



A Jupyter Notebook interface with a single code cell. The code cell contains the command `print(df.aggregate(['max', 'min', 'sum']))`. Below the code cell, the output is displayed as a table with columns 'Student', 'Physics', and 'Chemistry'. The rows represent the aggregation functions: 'max' (Tania, 85, 88), 'min' (Arif, 68, 69), and 'sum' (ArifTaniaNabil, 227, 229).

```
print(df.aggregate(['max', 'min', 'sum']))
```

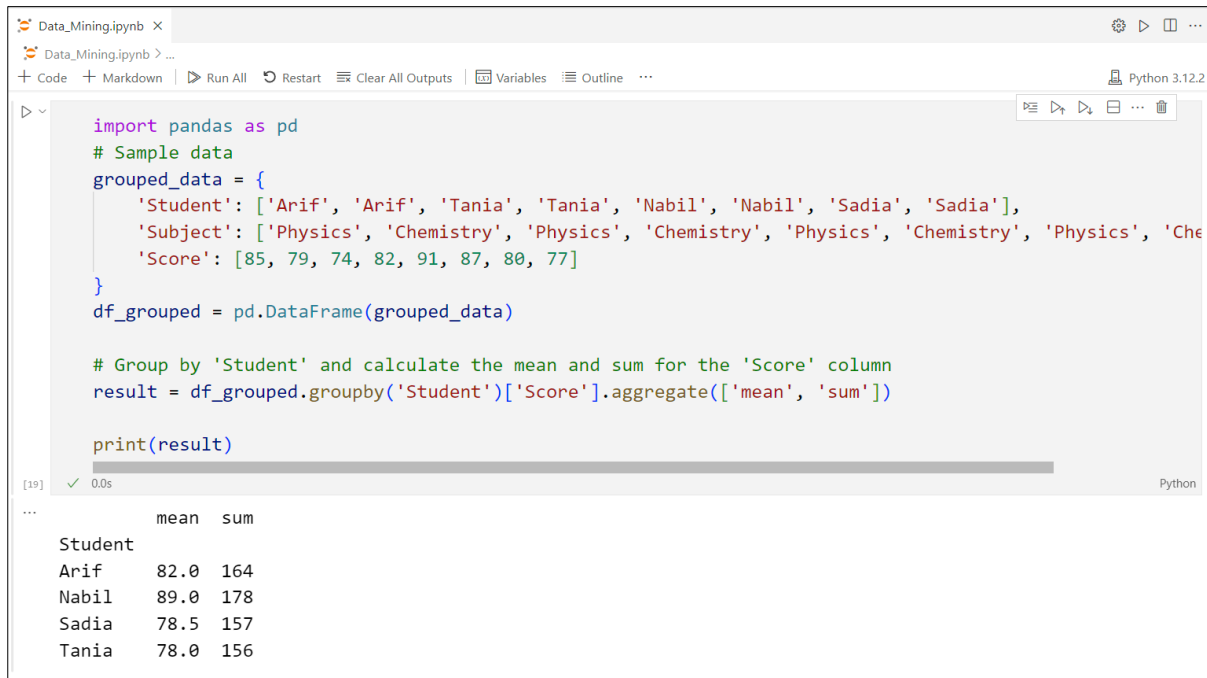
```
[17] ✓ 0.0s
```

```
Student  Physics  Chemistry
max      Tania    85         88
min      Arif     68         69
sum      ArifTaniaNabil  227      229
```

12. Grouping and Aggregation

Method: `df.groupby('Student').aggregate(['mean', 'sum'])`

This method groups the DataFrame by a column and applies aggregation functions.



The screenshot shows a Jupyter Notebook interface with a code cell containing the following Python code:

```
import pandas as pd
# Sample data
grouped_data = {
    'Student': ['Arif', 'Arif', 'Tania', 'Tania', 'Nabil', 'Nabil', 'Sadia', 'Sadia'],
    'Subject': ['Physics', 'Chemistry', 'Physics', 'Chemistry', 'Physics', 'Chemistry', 'Physics', 'Chemistry'],
    'Score': [85, 79, 74, 82, 91, 87, 80, 77]
}
df_grouped = pd.DataFrame(grouped_data)

# Group by 'Student' and calculate the mean and sum for the 'Score' column
result = df_grouped.groupby('Student')['Score'].aggregate(['mean', 'sum'])

print(result)
```

The output of the code is displayed below the cell:

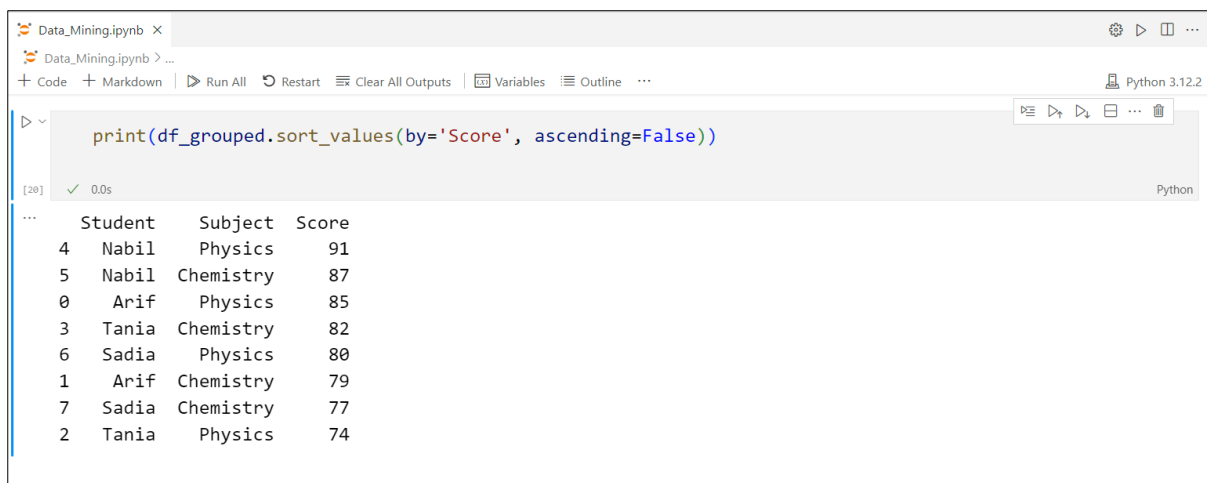
```
[19] ✓ 0.0s
```

	mean	sum
Student		
Arif	82.0	164
Nabil	89.0	178
Sadia	78.5	157
Tania	78.0	156

13. Sorting Values

Method: `df.sort_values(by='Score', ascending=False)`

This method sorts the DataFrame by a specified column.



The screenshot shows a Jupyter Notebook interface with a code cell containing the following Python code:

```
print(df_grouped.sort_values(by='Score', ascending=False))
```

The output of the code is displayed below the cell:

```
[20] ✓ 0.0s
```

	Student	Subject	Score
4	Nabil	Physics	91
5	Nabil	Chemistry	87
0	Arif	Physics	85
3	Tania	Chemistry	82
6	Sadia	Physics	80
1	Arif	Chemistry	79
7	Sadia	Chemistry	77
2	Tania	Physics	74

14. Handling Missing Values

Method: `df.fillna(0)`

This method replaces missing values in the DataFrame with a specified value.



The screenshot shows a Jupyter Notebook interface with a code cell and its output. The code cell contains the following Python code:

```
import numpy as np
missing_data = {
    'Student': ['Aisha', 'Kamal', 'Maya', 'Rafiq'],
    'Physics': [np.nan, 65, 78, 82],
    'Chemistry': [76, np.nan, 89, 92]
}
df_missing = pd.DataFrame(missing_data)
print(df_missing.fillna(0))
```

The output of the code is a DataFrame with 4 rows and 3 columns. The first row is the header, and the subsequent rows show the data for each student, with missing values replaced by 0.0.

	Student	Physics	Chemistry
0	Aisha	0.0	76.0
1	Kamal	65.0	0.0
2	Maya	78.0	89.0
3	Rafiq	82.0	92.0