

Data Mining and Data Warehousing Lab

CSEL-4108

Assignment on Data Analysis Techniques Using Pandas – 1

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

pd.Series()

The Series method creates a one-dimensional array-like object, akin to a Python list or array, that can hold any data type and is indexed by labels. For example:

A screenshot of a Jupyter Notebook interface. The top bar shows the file name 'Assignment.ipynb' and the Python version 'Python 3.12.2'. The code area contains the following Python code:

```
import pandas as pd

# pd.Series() Example
scores = pd.Series([85, 92, 78], index=['Alice', 'Bob', 'Charlie'])
scores
```

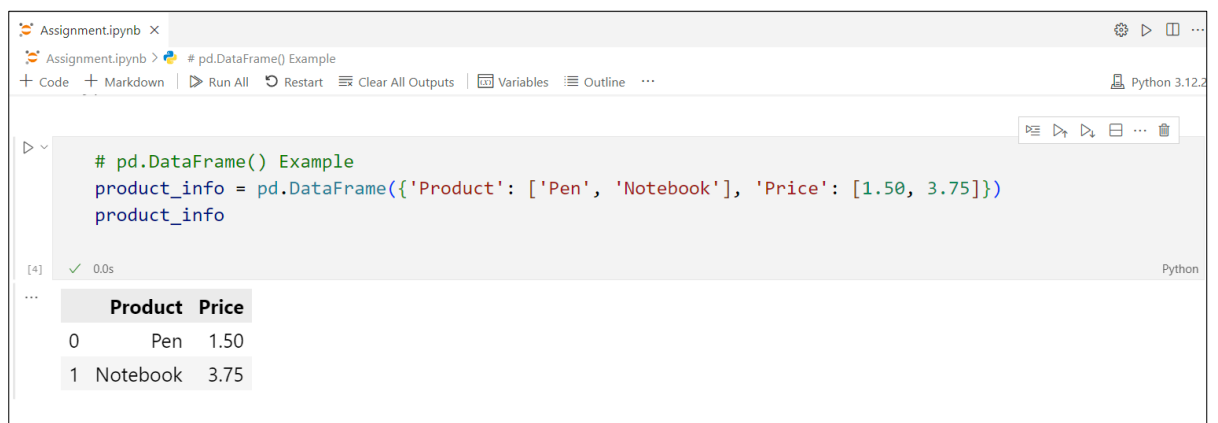
The output area shows the result of the code execution, which is a pandas Series object. The output is displayed as a table with two columns: the index labels and the corresponding values. The dtype is shown as 'int64'.

Alice	85
Bob	92
Charlie	78

dtype: int64

pd.DataFrame()

The DataFrame method constructs a two-dimensional, mutable, and potentially heterogeneous data structure, similar to a table with labeled axes (rows and columns). For instance:

A screenshot of a Jupyter Notebook interface. The top bar shows the file name 'Assignment.ipynb' and the Python version 'Python 3.12.2'. The code area contains the following Python code:

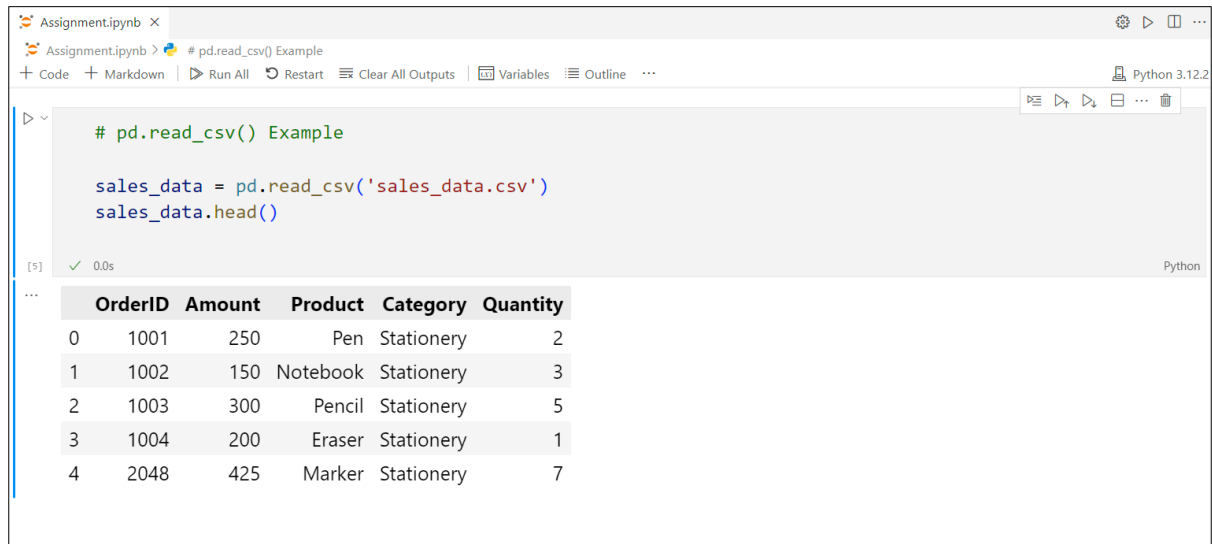
```
# pd.DataFrame() Example
product_info = pd.DataFrame({'Product': ['Pen', 'Notebook'], 'Price': [1.50, 3.75]})
product_info
```

The output area shows the result of the code execution, which is a pandas DataFrame object. The output is displayed as a table with two columns: 'Product' and 'Price'. The index is shown as 0 and 1.

	Product	Price
0	Pen	1.50
1	Notebook	3.75

pd.read_csv()

The `read_csv` method imports data from a CSV file into a DataFrame. It's an essential method for loading data from external sources. Example:



```
Assignment.ipynb x
Assignment.ipynb > # pd.read_csv() Example
+ Code + Markdown | Run All Restart Clear All Outputs Variables Outline ... Python 3.12.2

# pd.read_csv() Example

sales_data = pd.read_csv('sales_data.csv')
sales_data.head()
```

[5] ✓ 0.0s Python

	OrderID	Amount	Product	Category	Quantity
0	1001	250	Pen	Stationery	2
1	1002	150	Notebook	Stationery	3
2	1003	300	Pencil	Stationery	5
3	1004	200	Eraser	Stationery	1
4	2048	425	Marker	Stationery	7

DataFrame.head()

The `head` method returns the first `n` rows of a DataFrame, defaulting to 5 rows if `n` isn't specified. Example:



```
Assignment.ipynb x
Assignment.ipynb > # DataFrame.head() Example
+ Code + Markdown | Run All Restart Clear All Outputs Variables Outline ... Python 3.12.2

# DataFrame.head() Example
top_entries = product_info.head(1)
top_entries
```

[6] ✓ 0.0s Python

	Product	Price
0	Pen	1.5

DataFrame.tail()

The tail method provides the last n rows of a DataFrame, with 5 rows as the default when n is not given. Example:



A screenshot of a Jupyter Notebook interface. The top bar shows 'Assignment.ipynb' and 'Python 3.12.2'. The code cell contains the following Python code:

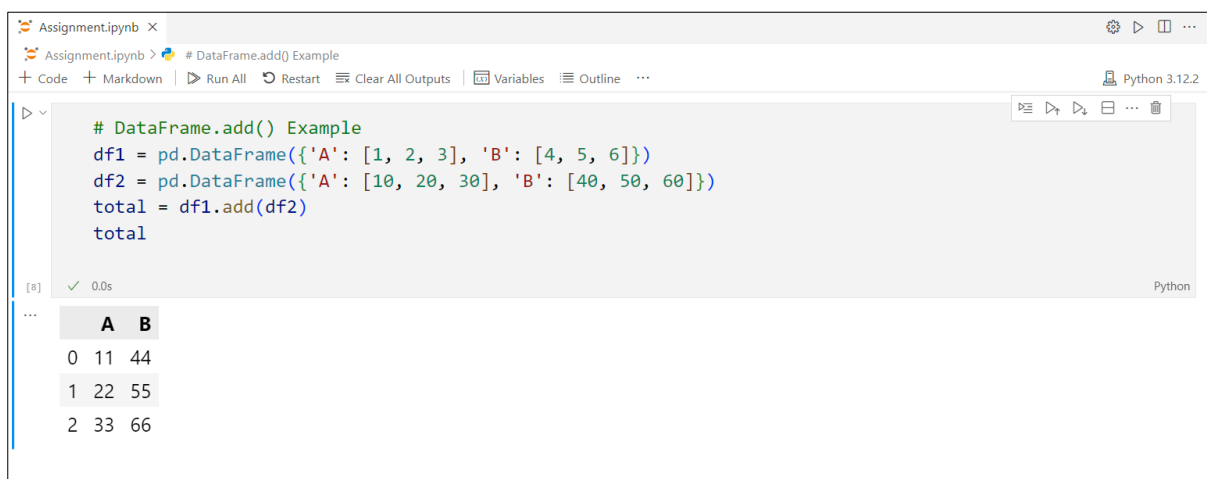
```
# DataFrame.tail() Example
recent_sales = product_info.tail(1)
recent_sales
```

The output cell shows a single row of data with columns 'Product' and 'Price'.

	Product	Price
1	Notebook	3.75

DataFrame.add()

The add method performs element-wise addition between two DataFrames or between a DataFrame and a Series. Missing values are replaced with NaN unless specified otherwise. Example:



A screenshot of a Jupyter Notebook interface. The top bar shows 'Assignment.ipynb' and 'Python 3.12.2'. The code cell contains the following Python code:

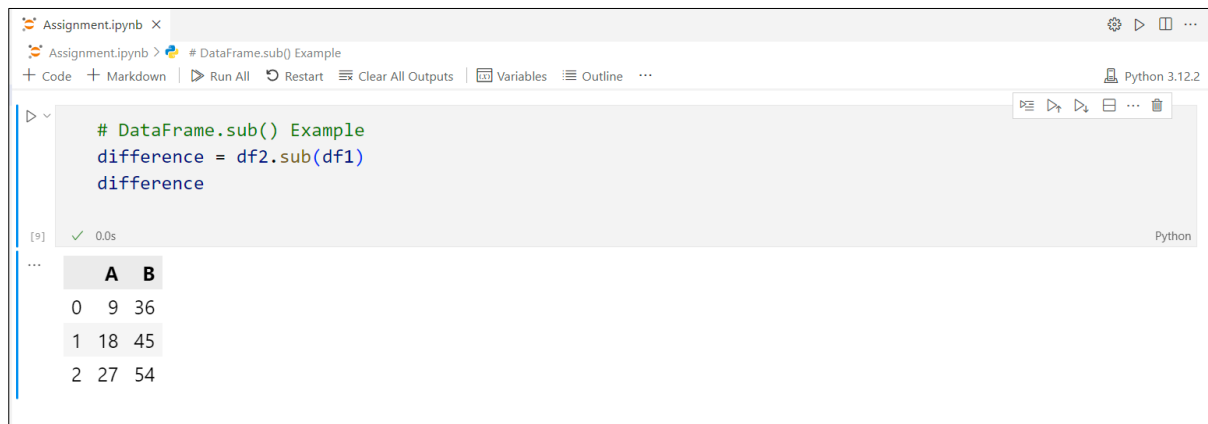
```
# DataFrame.add() Example
df1 = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
df2 = pd.DataFrame({'A': [10, 20, 30], 'B': [40, 50, 60]})
total = df1.add(df2)
total
```

The output cell shows the result of the element-wise addition of df1 and df2.

	A	B
0	11	44
1	22	55
2	33	66

DataFrame.sub()

The sub method allows for element-wise subtraction between two DataFrames or between a DataFrame and a Series. Missing values can be substituted using the fill_value argument. Example:



A screenshot of a Jupyter Notebook interface. The top bar shows the file name 'Assignment.ipynb' and the current cell's title '# DataFrame.sub() Example'. The code area contains the following Python code:

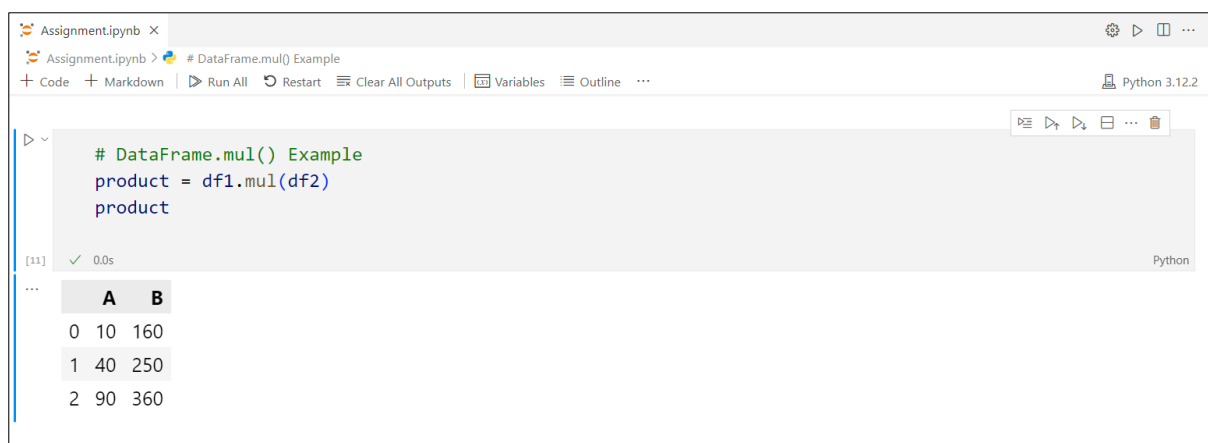
```
# DataFrame.sub() Example
difference = df2.sub(df1)
difference
```

Below the code, the output is displayed as a table with two columns, 'A' and 'B', and three rows of data:

	A	B
0	9	36
1	18	45
2	27	54

DataFrame.mul()

The mul method is used for element-wise multiplication between two DataFrames or between a DataFrame and a Series. It also allows for filling in missing values. Example:



A screenshot of a Jupyter Notebook interface. The top bar shows the file name 'Assignment.ipynb' and the current cell's title '# DataFrame.mul() Example'. The code area contains the following Python code:

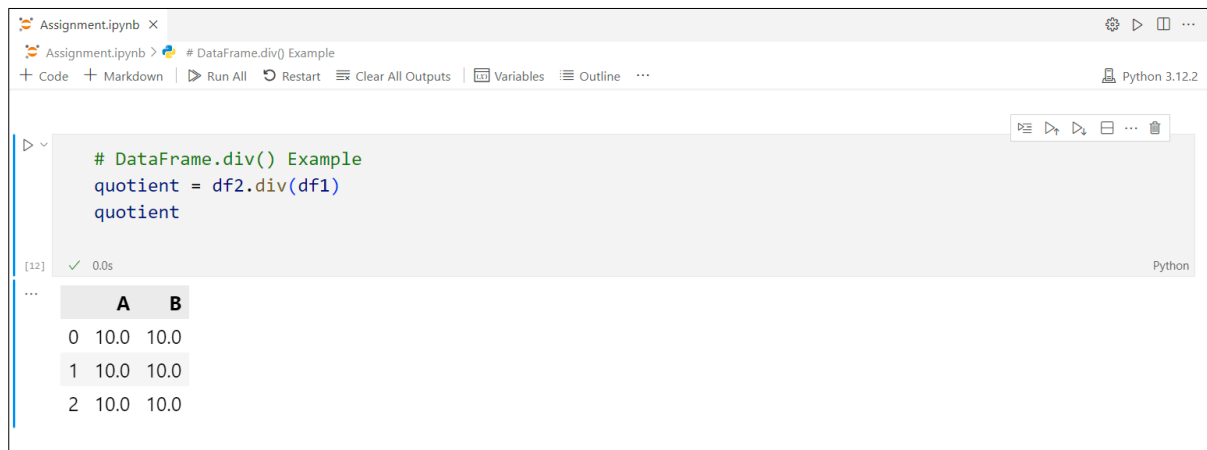
```
# DataFrame.mul() Example
product = df1.mul(df2)
product
```

Below the code, the output is displayed as a table with two columns, 'A' and 'B', and three rows of data:

	A	B
0	10	160
1	40	250
2	90	360

DataFrame.div()

The `div` method performs element-wise division between a `DataFrame` and another `DataFrame` or `Series`. It also supports handling missing data through the `fill_value` parameter. Example:



A screenshot of a Jupyter Notebook interface. The top bar shows the file name 'Assignment.ipynb' and the current cell's title '# DataFrame.div() Example'. The code cell contains the following Python code:

```
# DataFrame.div() Example
quotient = df2.div(df1)
quotient
```

The output of the code is displayed below the cell, showing a `DataFrame` with two columns, 'A' and 'B', and three rows (index 0 to 2). The values in both columns are 10.0.

	A	B
0	10.0	10.0
1	10.0	10.0
2	10.0	10.0

DataFrame.loc[]

The `loc` method provides label-based indexing to select specific rows or columns within a `DataFrame`. It allows for slicing, boolean arrays, and lists of labels. Example:



A screenshot of a Jupyter Notebook interface. The top bar shows the file name 'Assignment.ipynb' and the current cell's title '# DataFrame.loc[] Example'. The code cell contains the following Python code:


```
# DataFrame.loc[] Example
selected_rows = product_info.loc[0:1, ['Product', 'Price']]
selected_rows
```

The output of the code is displayed below the cell, showing a `DataFrame` with two columns, 'Product' and 'Price', and two rows (index 0 to 1). The values are 'Pen' and 1.50 for index 0, and 'Notebook' and 3.75 for index 1.

	Product	Price
0	Pen	1.50
1	Notebook	3.75

DataFrame.drop()

The drop method removes specified rows or columns from a DataFrame by their label names. Example:



A screenshot of a Jupyter Notebook interface. The top bar shows the file name 'Assignment.ipynb' and the current cell's title '# DataFrame.drop() Example'. The code area contains the following Python code:


```
# DataFrame.drop() Example
cleaned_data = product_info.drop('Price', axis=1)
cleaned_data
```

Below the code, the output is displayed as a table with two rows and one column labeled 'Product'.

	Product
0	Pen
1	Notebook

DataFrame.rename()

The rename method is used to change the labels of rows or columns within a DataFrame. It allows for renaming multiple labels simultaneously. Example:



A screenshot of a Jupyter Notebook interface. The top bar shows the file name 'Assignment.ipynb' and the current cell's title '# DataFrame.rename() Example'. The code area contains the following Python code:

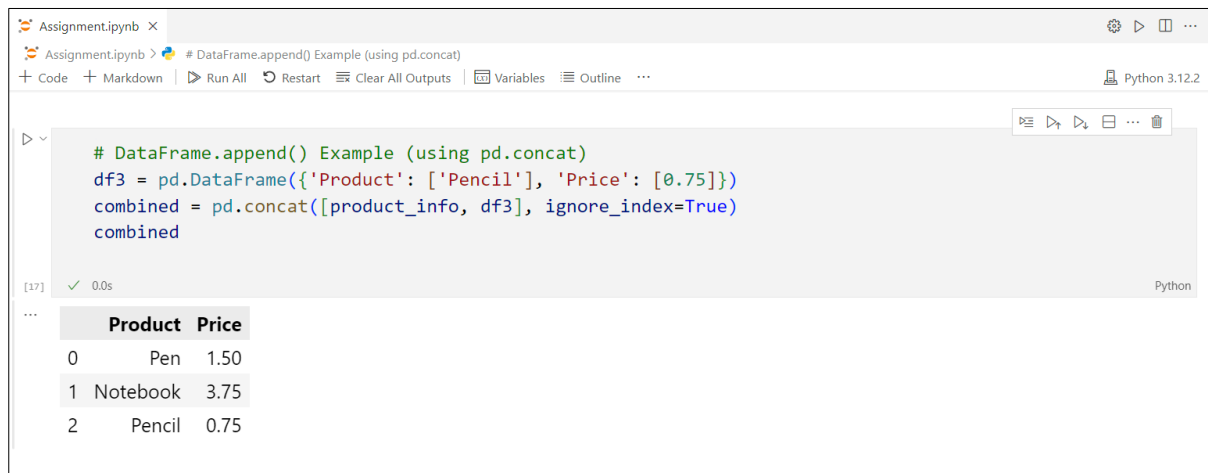
```
# DataFrame.rename() Example
renamed_data = product_info.rename(columns={'Product': 'Item'})
renamed_data
```

Below the code, the output is displayed as a table with two rows and two columns labeled 'Item' and 'Price'.

	Item	Price
0	Pen	1.50
1	Notebook	3.75

DataFrame.append()

The append method allows you to add rows from another DataFrame to the end of the first DataFrame. If the columns in the second DataFrame are not present in the first, they will be added as new columns. Example:

A screenshot of a Jupyter Notebook interface. The top bar shows the file name 'Assignment.ipynb' and the current cell's title '# DataFrame.append() Example (using pd.concat)'. The toolbar includes options for Code, Markdown, Run All, Restart, Clear All Outputs, Variables, Outline, and a Python version indicator 'Python 3.12.2'. The code cell contains the following Python code:

```
# DataFrame.append() Example (using pd.concat)
df3 = pd.DataFrame({'Product': ['Pencil'], 'Price': [0.75]})
combined = pd.concat([product_info, df3], ignore_index=True)
combined
```

The output cell shows the result of the code execution, which is a DataFrame with three rows. The first two rows are from the 'product_info' DataFrame, and the third row is from 'df3'. The DataFrame has two columns: 'Product' and 'Price'. The output is displayed as a table with the following data:

	Product	Price
0	Pen	1.50
1	Notebook	3.75
2	Pencil	0.75