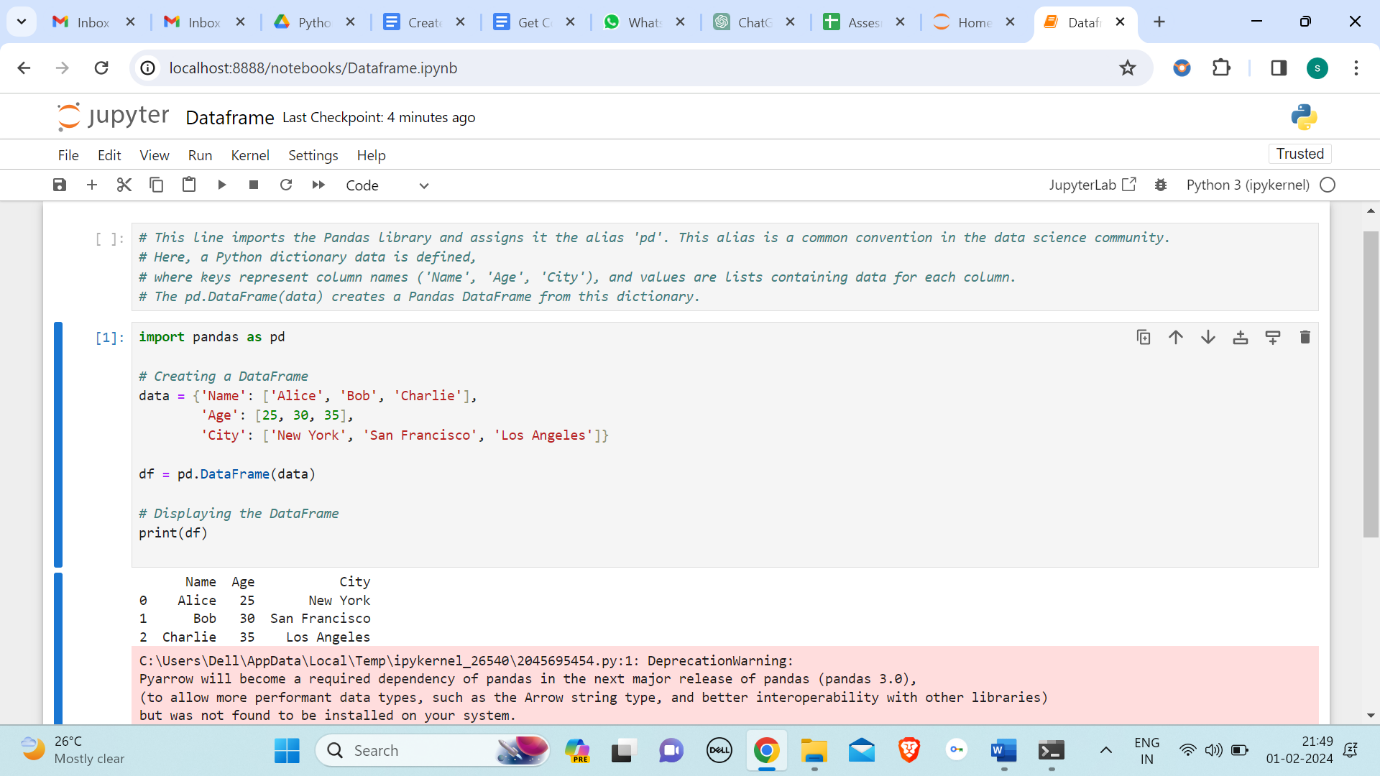
**Pandas for Data Processing**

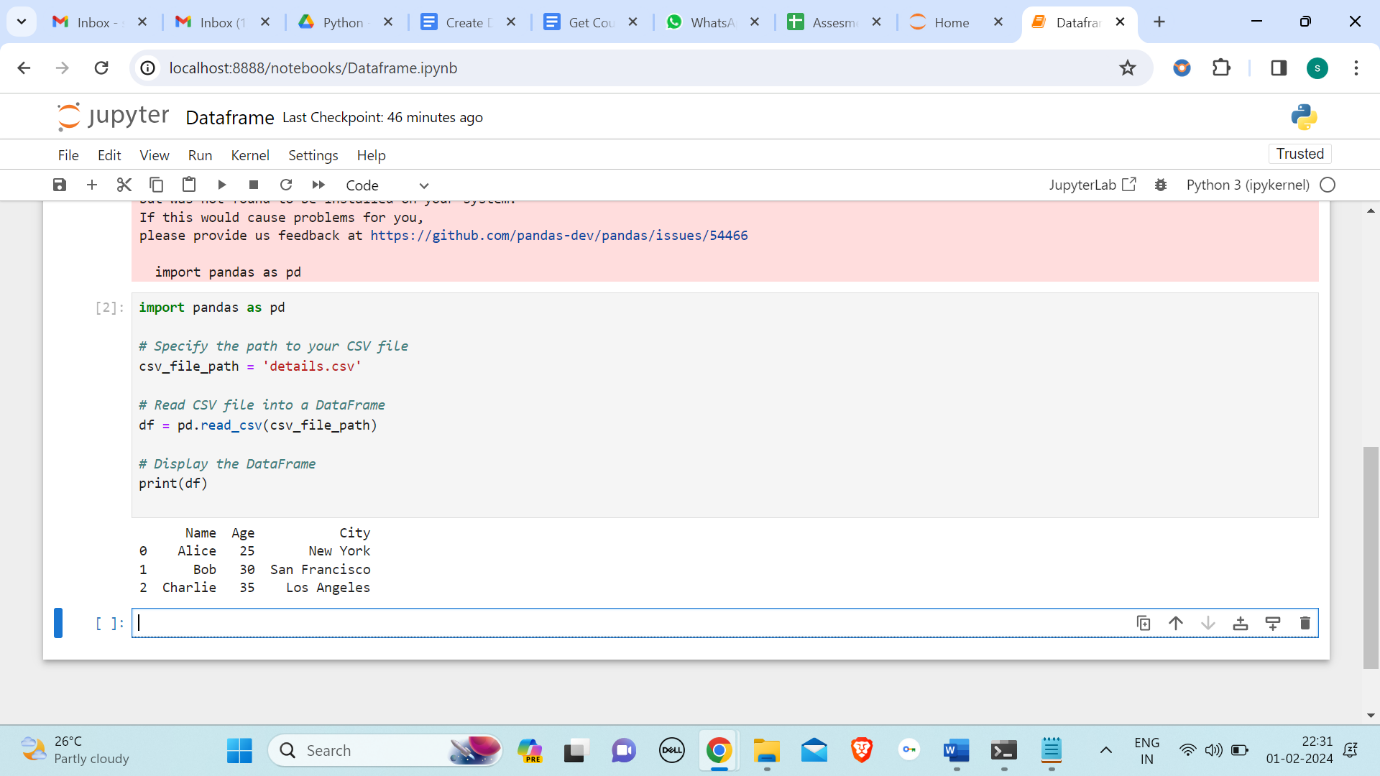
Pandas is a powerful open-source data manipulation and analysis library for Python. It provides easy-to-use data structures, such as Series and DataFrame, along with a wide variety of functions to facilitate data manipulation, cleaning, exploration, and analysis.

DataFrame: The central data structure in Pandas is the DataFrame, a two-dimensional table with labeled axes (rows and columns). It is similar to a spreadsheet or SQL table and is highly efficient for working with structured data.

Here's a simple example of using Pandas to create a DataFrame:



Reading CSV data using using pandas



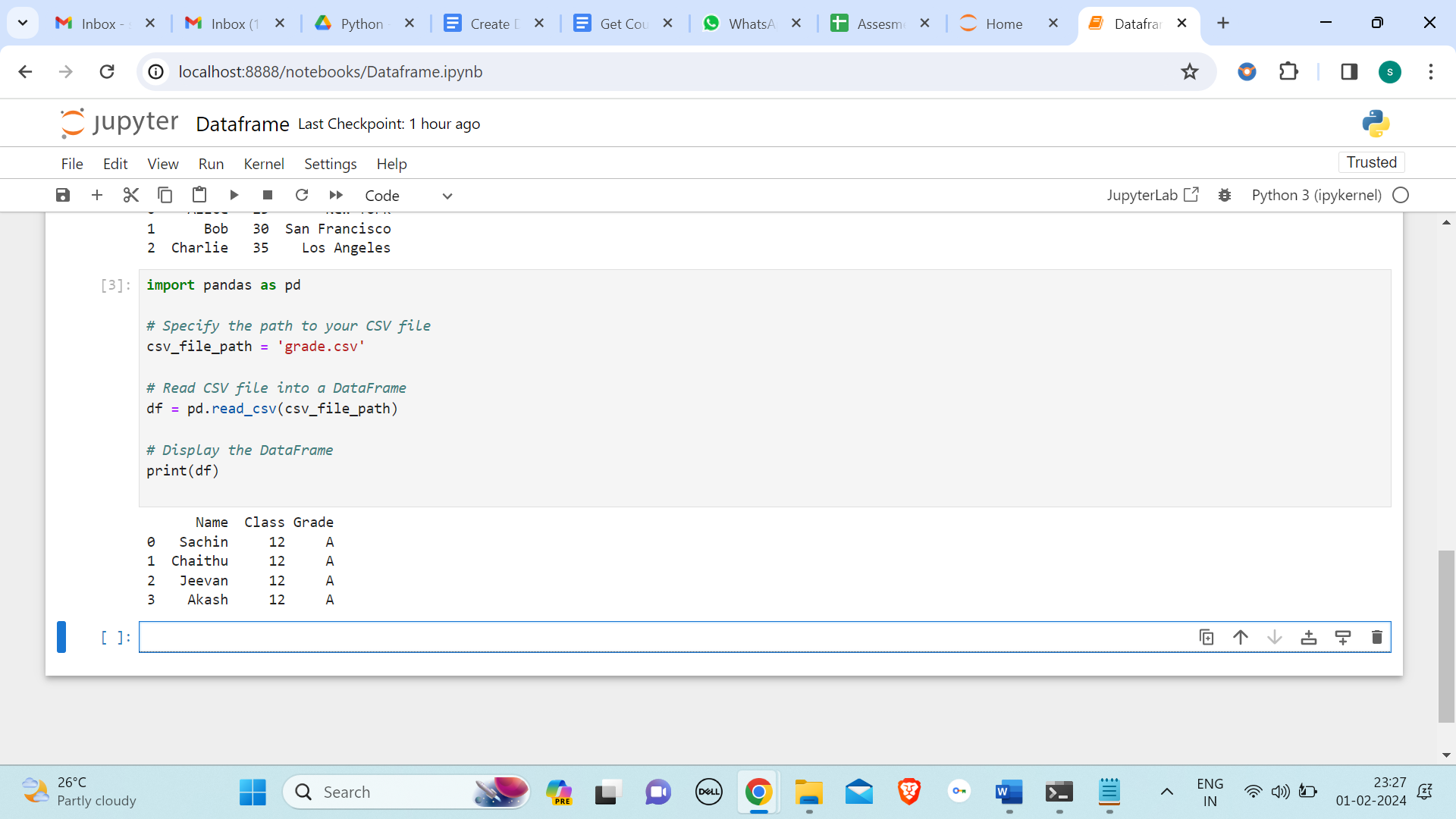
Pandas provides a function called **read\_csv()** to read data from a CSV file and create a DataFrame. In this example:

1. **pd.read\_csv(csv\_file\_path)** reads the CSV file specified by csv\_file\_path and creates a DataFrame (df) from its contents.
2. The **print(df)** statement displays the resulting DataFrame.

The read\_csv() function is versatile and can handle various CSV formats, including those with different delimiters, custom headers, and missing values. You can customize the behavior of read\_csv() by passing additional parameters based on your specific CSV file's characteristics.

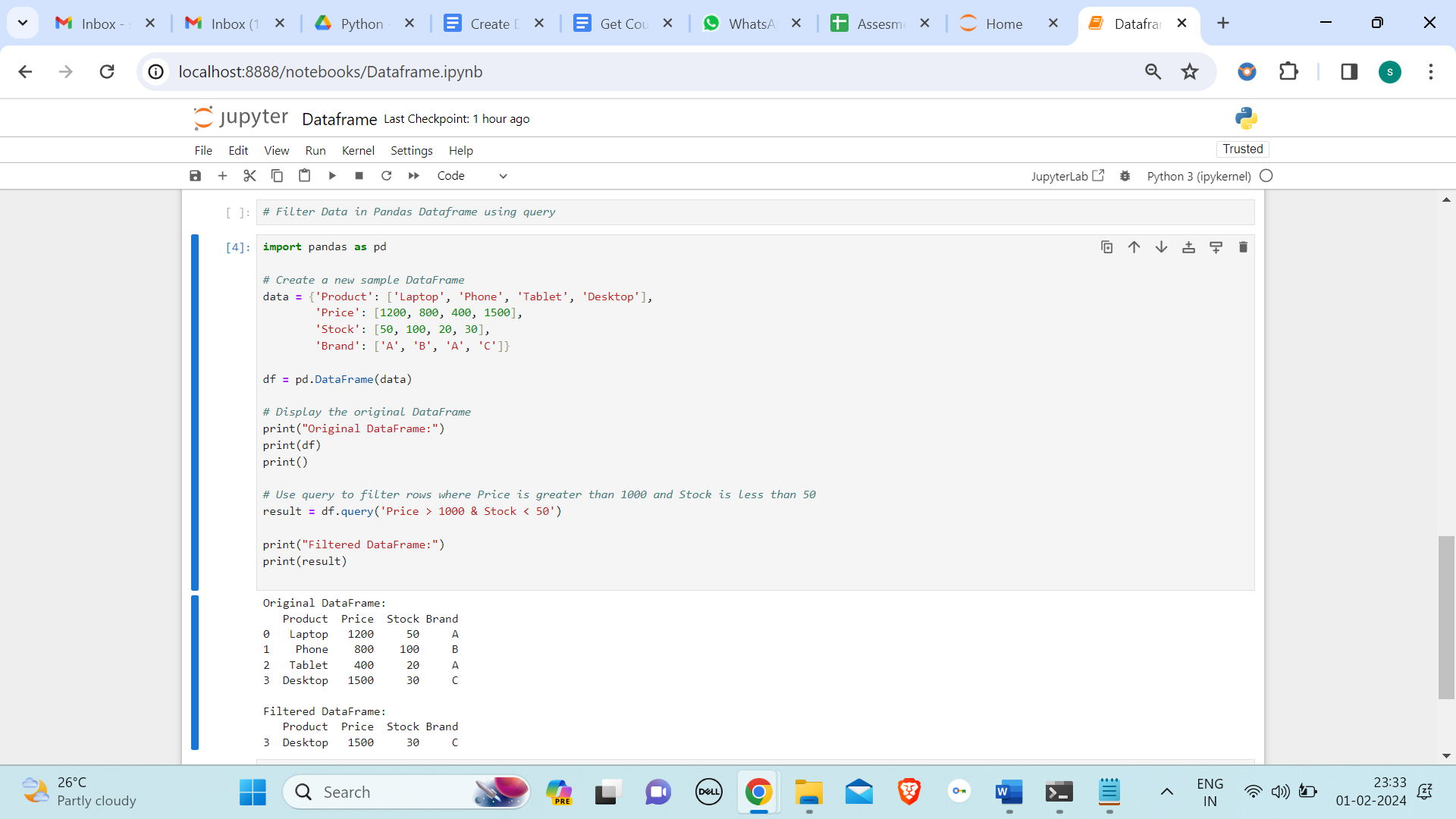
Read Data from CSV Files to Pandas Dataframes

"Reading Data from CSV Files to Pandas DataFrames" and "Reading CSV Data using Pandas" essentially refer to the same process, but they may be worded differently in different contexts.

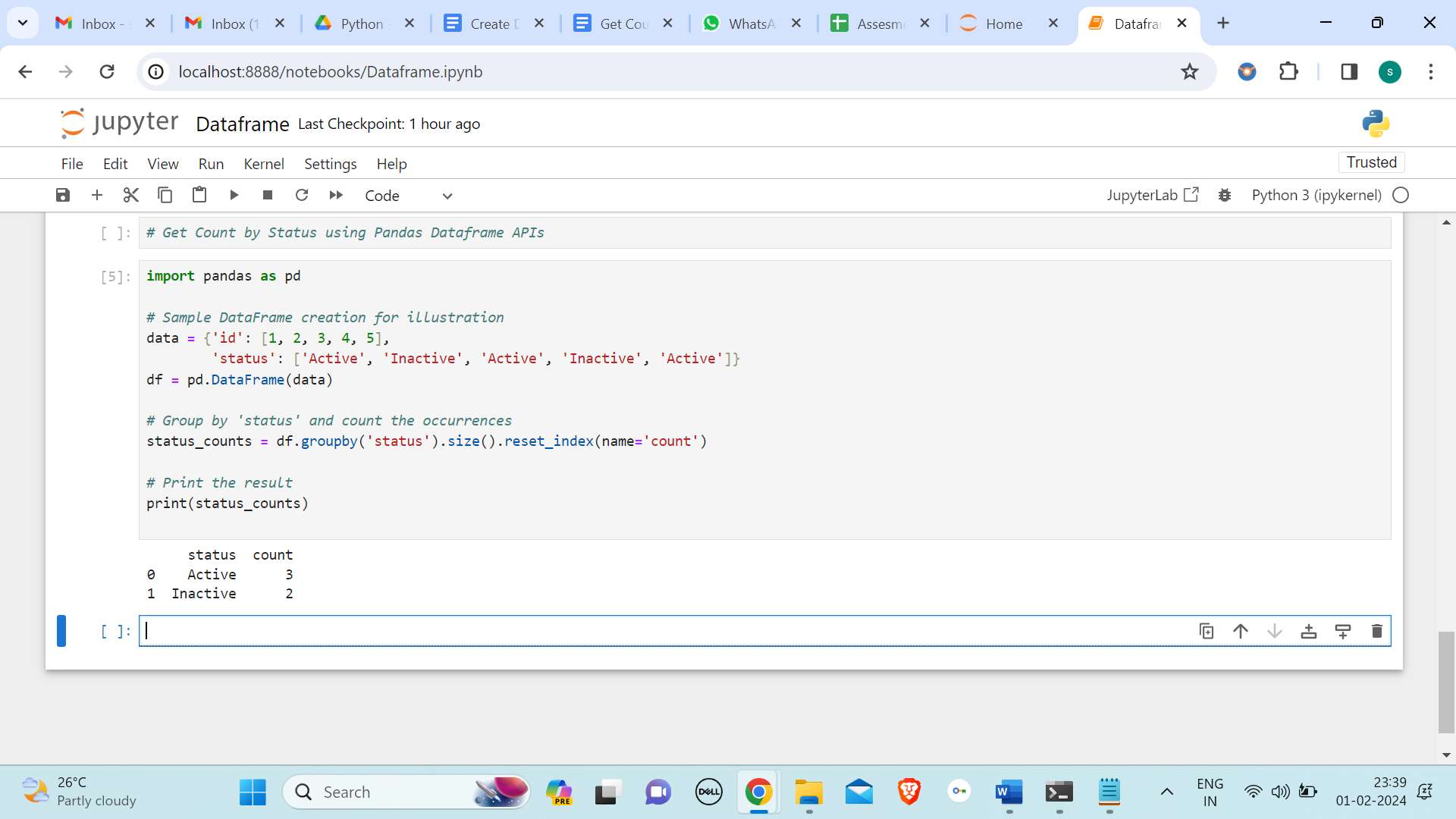


Filter Data in Pandas Dataframe using query

In pandas, you can use the query method to filter data in a DataFrame based on a specified condition. The query method allows you to write expressions that resemble SQL queries to filter rows of data. The below is the example for filter data in pandas dataframe using query. In this example, the condition in the query method is 'Price > 1000 & Stock < 50', which filters rows where the 'Price' is greater than 1000 and the 'Stock' is less than 50.



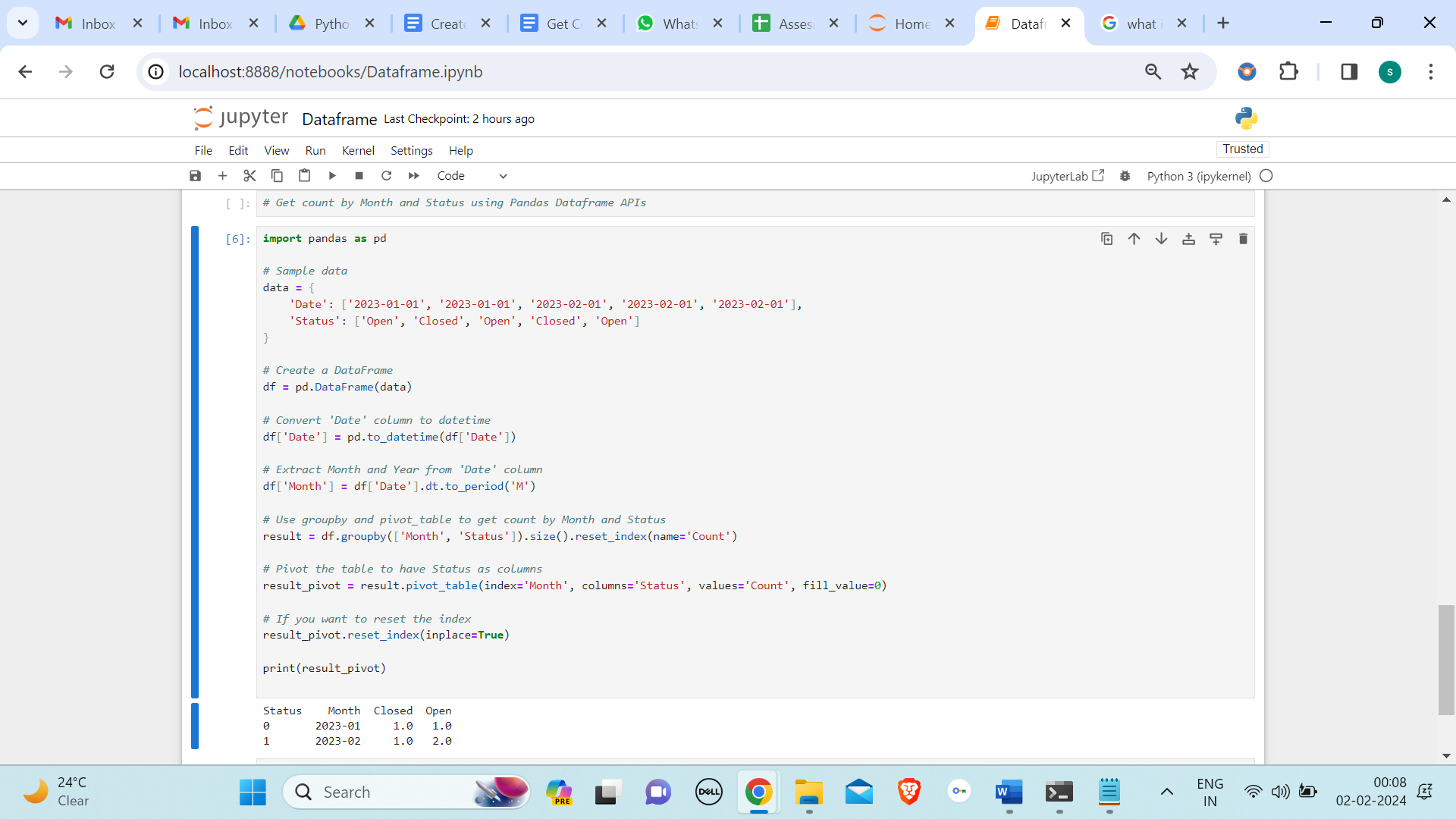
Get Count by Status using Pandas Dataframe APIs



To get the count by status using Pandas DataFrame APIs, you can use the groupby and count functions. Here's an example assuming you have a DataFrame called df with a column named 'status'. In this example, the **groupby('status')** groups the DataFrame by the 'status' column, and **size()** is used to get the count for each group**. reset\_index(name='count')** is then used to reset the index and rename the count column.

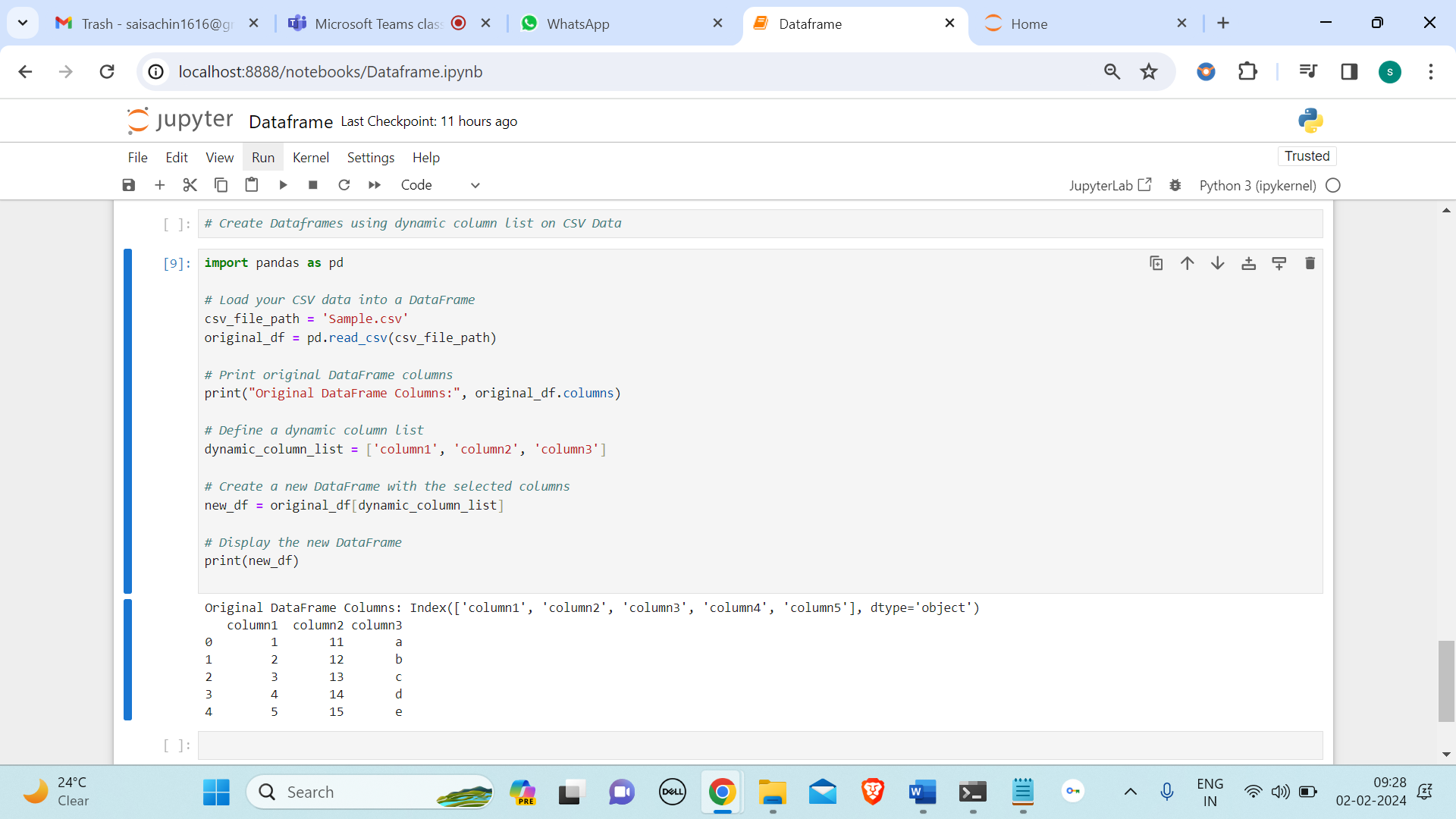
Get count by Month and Status using Pandas Dataframe APIs

If you have a DataFrame in Python using the Pandas library and you want to get the count of rows by month and status, you can use the groupby and pivot\_table functions. Here's an example assuming you have a DataFrame called df with columns 'Date', 'Status', and any other relevant columns. This will give you a DataFrame where each row represents a month, each column represents a status, and the values represent the count of occurrences for that combination of month and status.



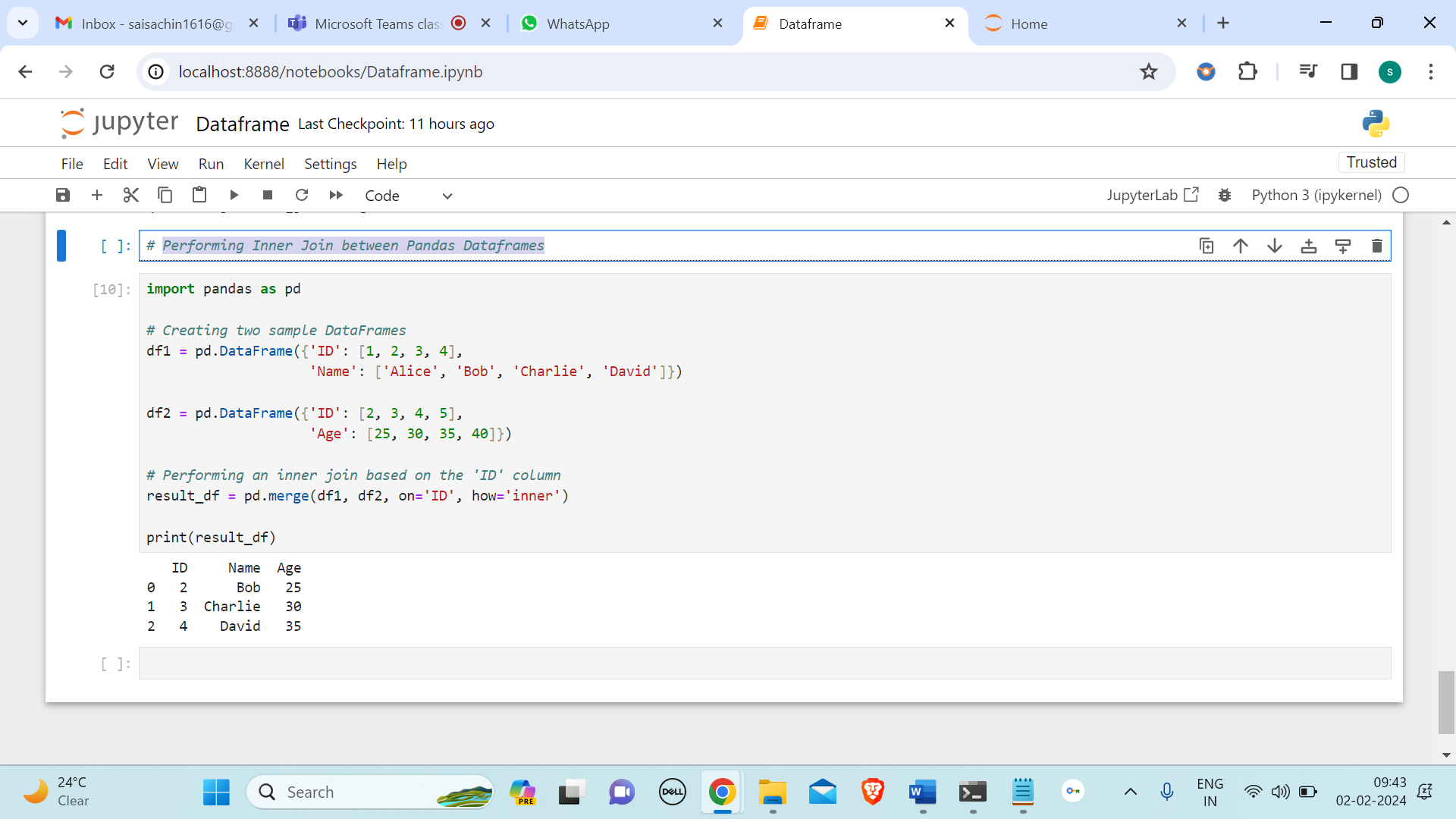
This program creates a simple DataFrame with 'Date' and 'Status' columns, converts the 'Date' column to datetime format, extracts the month, and then uses groupby and pivot\_table to get the count by month and status. Finally, it prints the resulting DataFrame.

Create Dataframes using dynamic column list on CSV Data



This script reads the CSV file into a DataFrame and then creates a new DataFrame (new\_df) with the columns specified in the **dynamic\_column\_list**. You can customize the column list based on your specific use case.

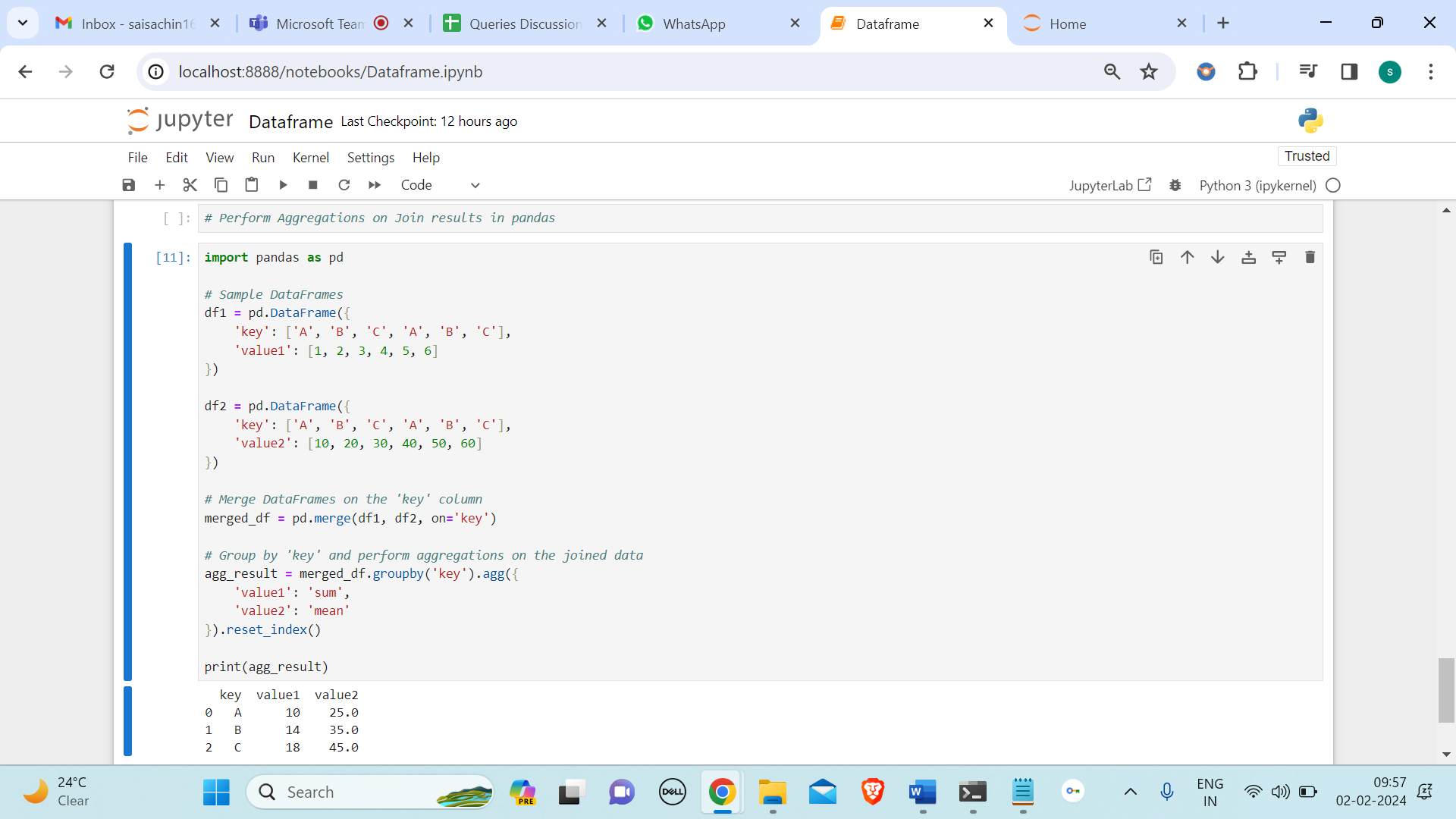
Performing Inner Join between Pandas Dataframes



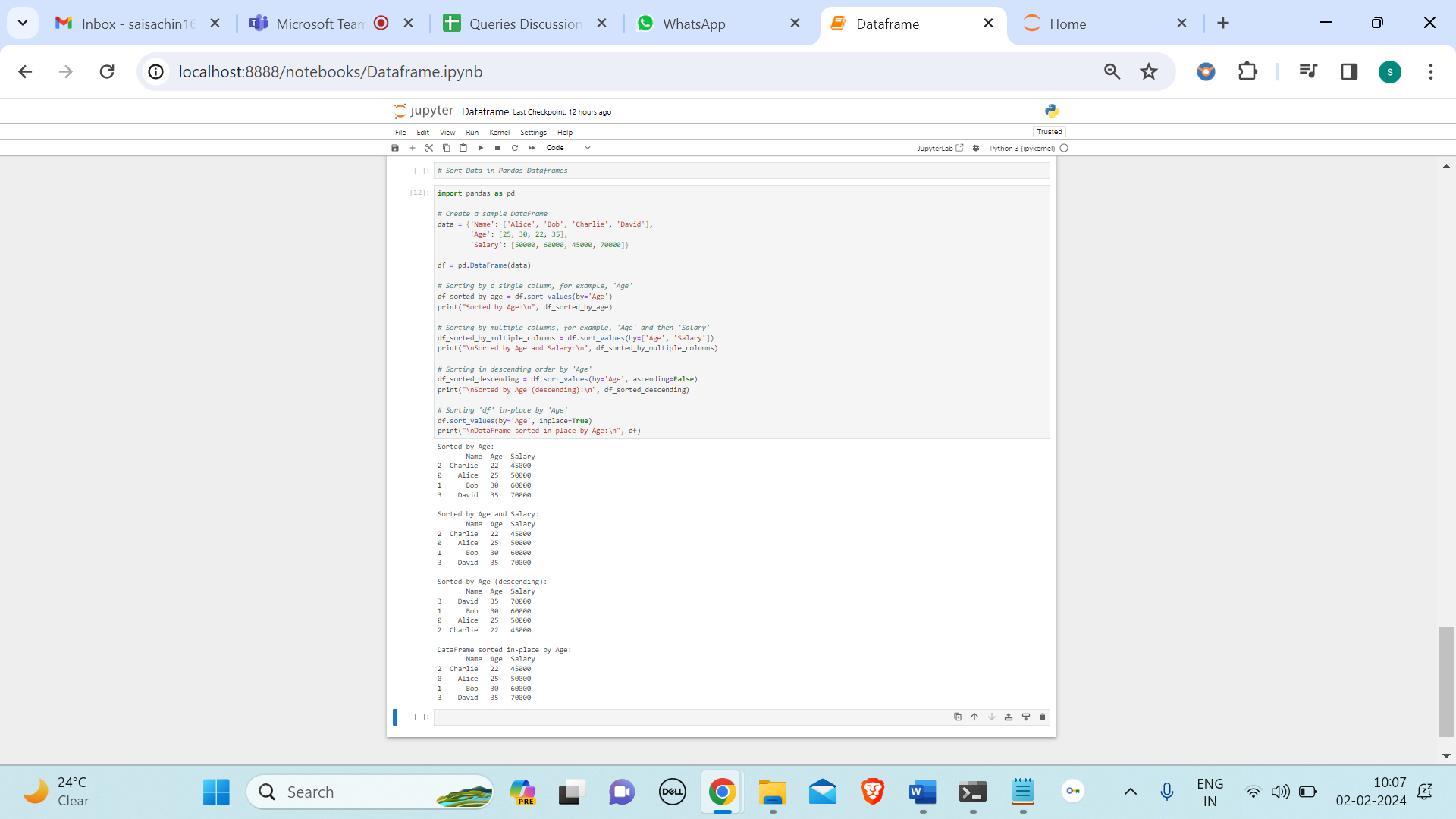
In Pandas, you can perform an inner join between two DataFrames using the **merge()** function. The merge() function allows you to combine two DataFrames based on a common column. In this example**, pd.merge()** is used to perform an inner join on the 'ID' column of both DataFrames (df1 and df2). The on parameter specifies the common column to join on, and the how parameter is set to 'inner' to perform an inner join. The resulting result\_df will contain only the rows where the 'ID' values are common in both DataFrames, and columns from both DataFrames will be included in the result.

Perform Aggregations on Join results in pandas

Performing aggregations on join results in Pandas involves using the **groupby** function to group the joined DataFrame by a certain key and then applying aggregation functions like sum, mean, etc. to the grouped data. **pd.merge** is used to join df1 and df2 based on the 'key' column. Then, **groupby** is applied on the 'key' column, and the agg function is used to specify the aggregation functions for each column. Finally, reset\_index is used to flatten the resulting DataFrame.



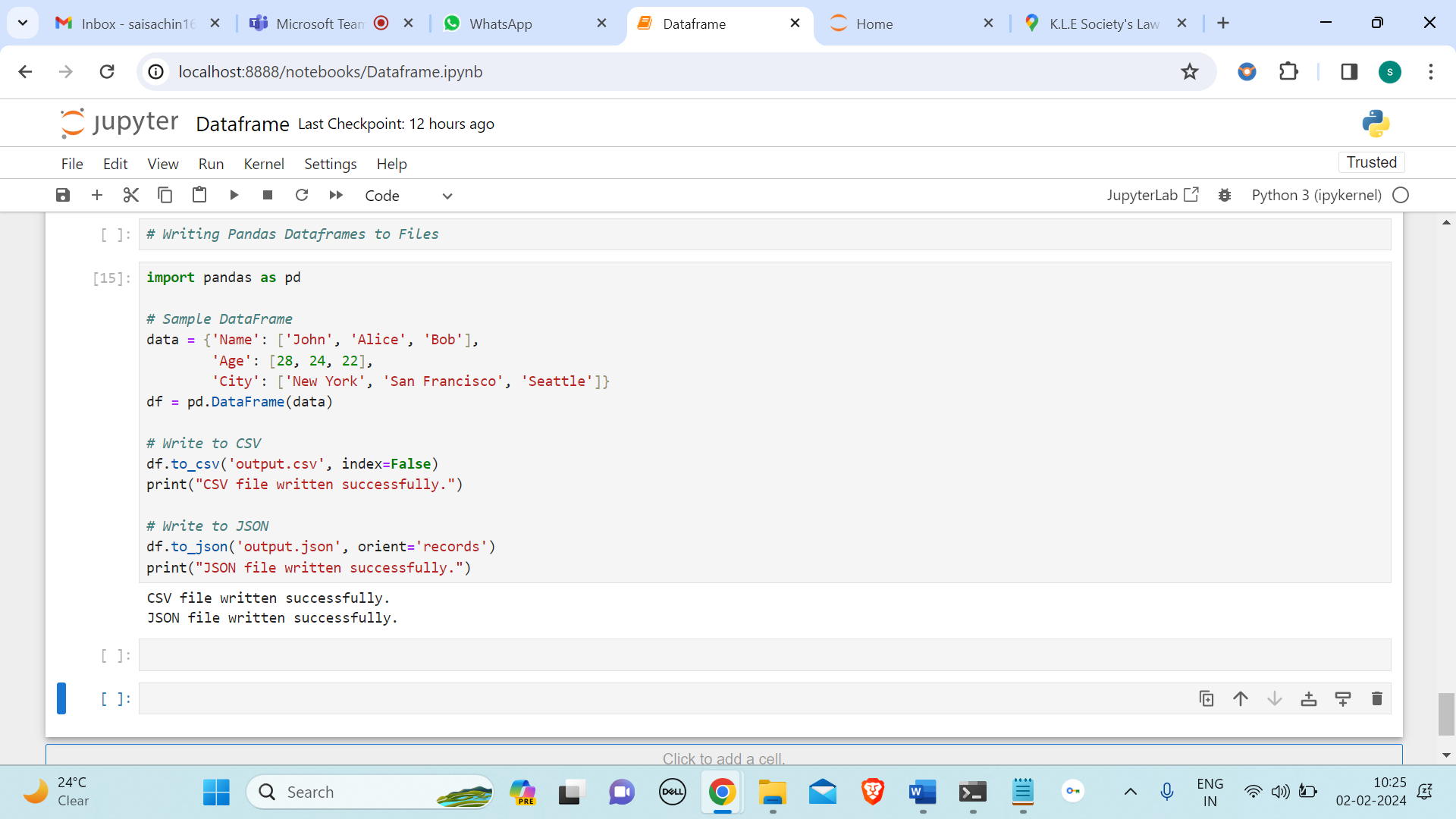
Sort Data in Pandas Dataframes



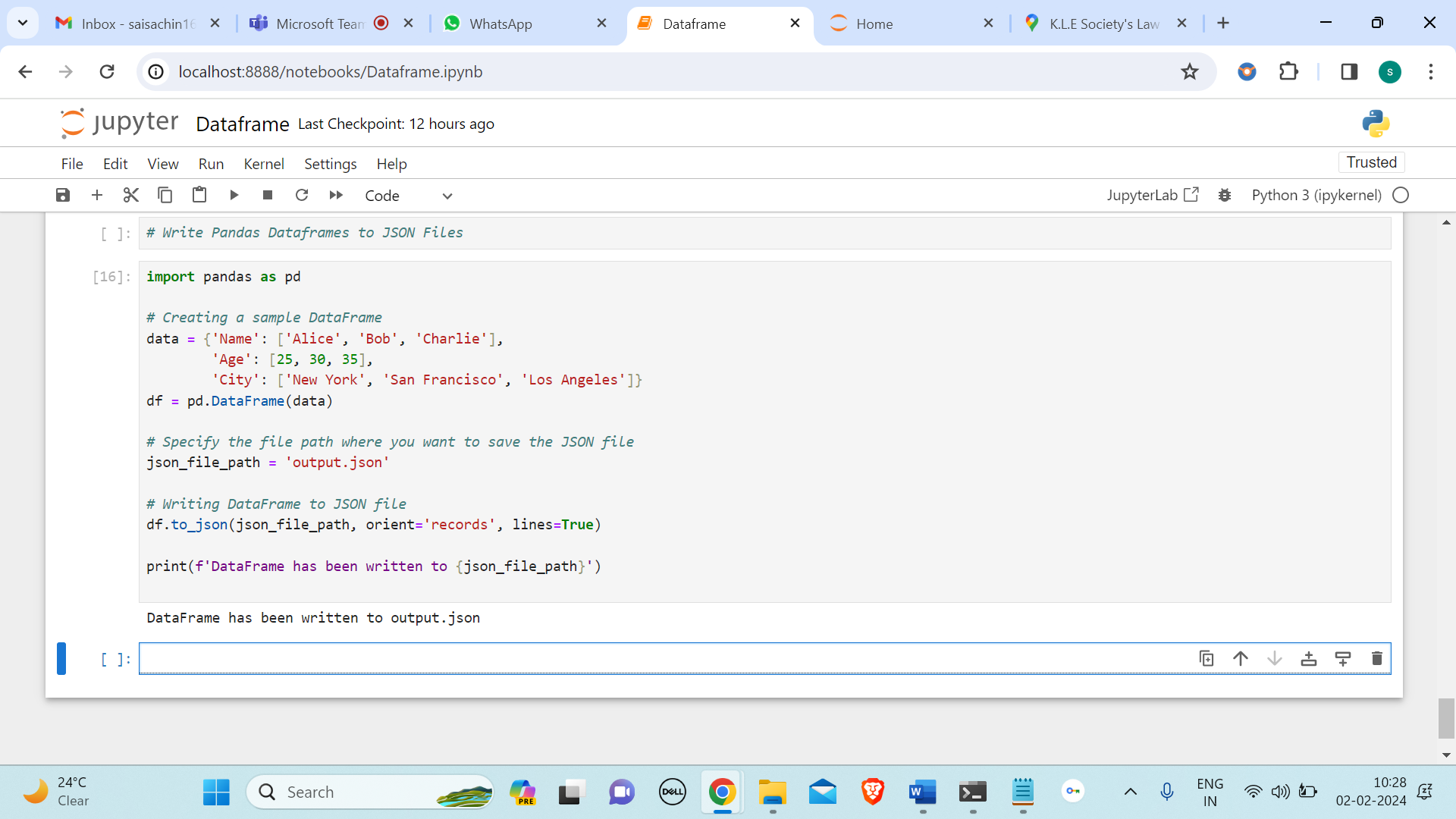
In the example above, **sort\_values()** is used to sort the DataFrame based on one or more columns. The **by** parameter specifies the column(s) to sort by. The resulting DataFrame will be sorted in ascending order by default. If you want to sort in descending order, you can use the **ascending** parameter. Remember to assign the sorted DataFrame to a variable or use the **inplace=True** argument to modify the existing DataFrame.

Writing Pandas Dataframes to Files

Pandas provides several methods to write DataFrames to various file formats. Here are some common file formats and examples of how to write Pandas DataFrames to them.



Write Pandas Dataframes to JSON Files



Pandas library to work with DataFrames and the to\_json method to write DataFrames to JSON files.

* **orient='records'** specifies that the DataFrame should be serialized as a list of dictionaries, where each dictionary represents a row in the DataFrame.
* **lines=True** specifies that each record (row) should be written as a separate line in the JSON file.