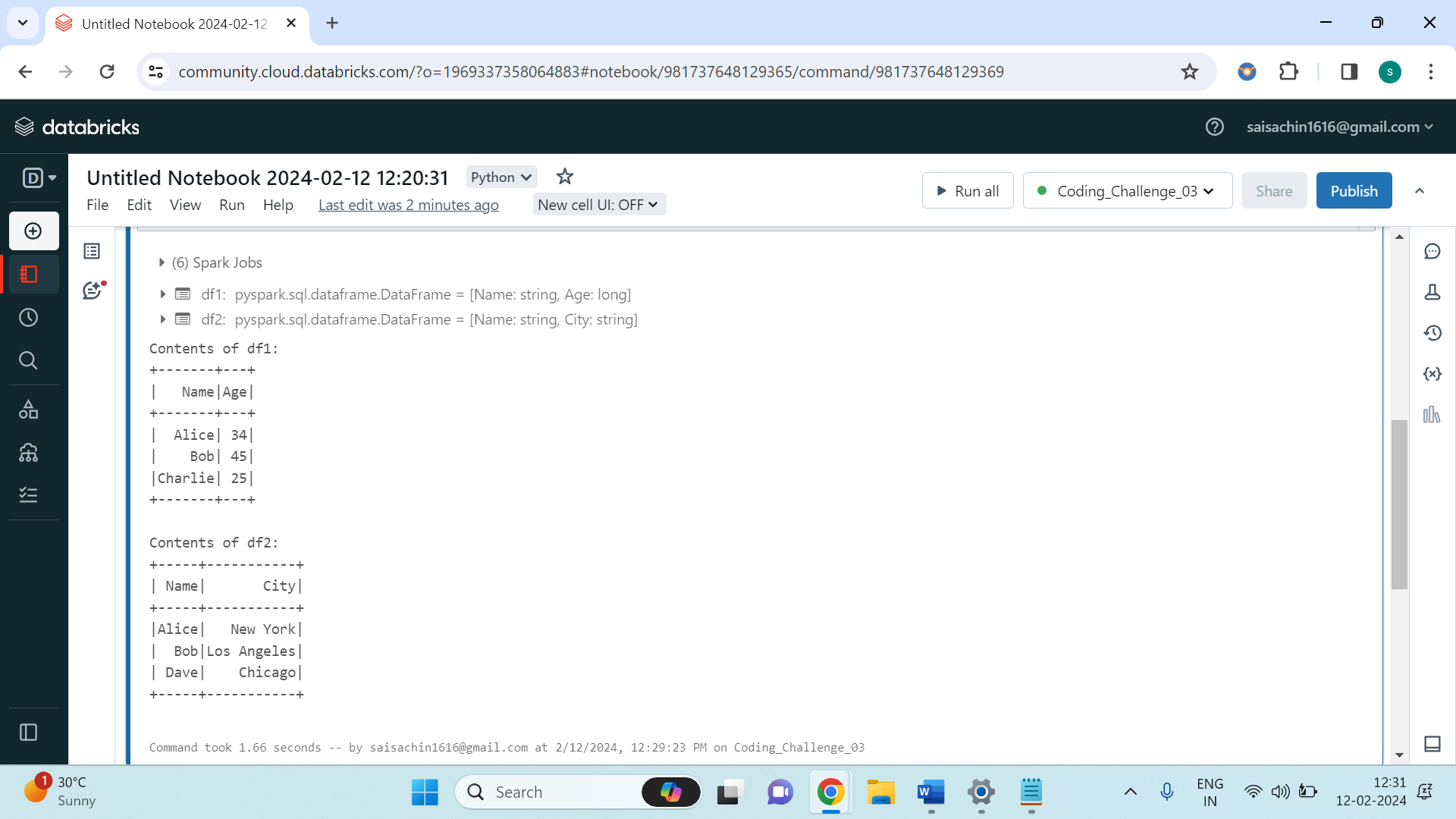
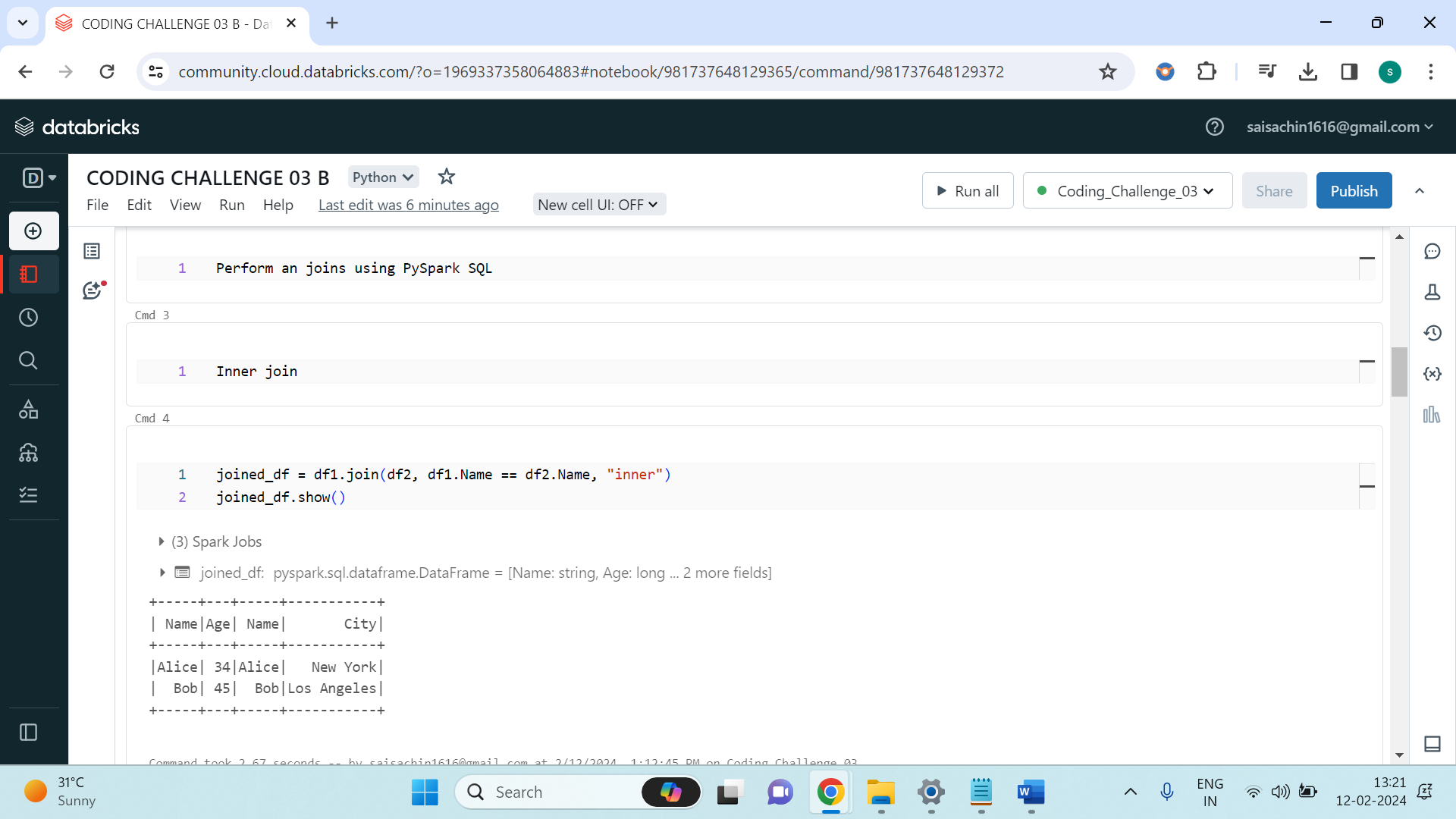


The provided code initializes a SparkSession, a crucial component for managing Spark functionality. This session enables the creation and manipulation of distributed datasets. Next, it creates two sample DataFrames, **df1** and **df2**, representing individual details like names, ages, and cities. By calling the **show()** method, the contents of both DataFrames are displayed, allowing for a quick review of the data structure and contents before further processing.

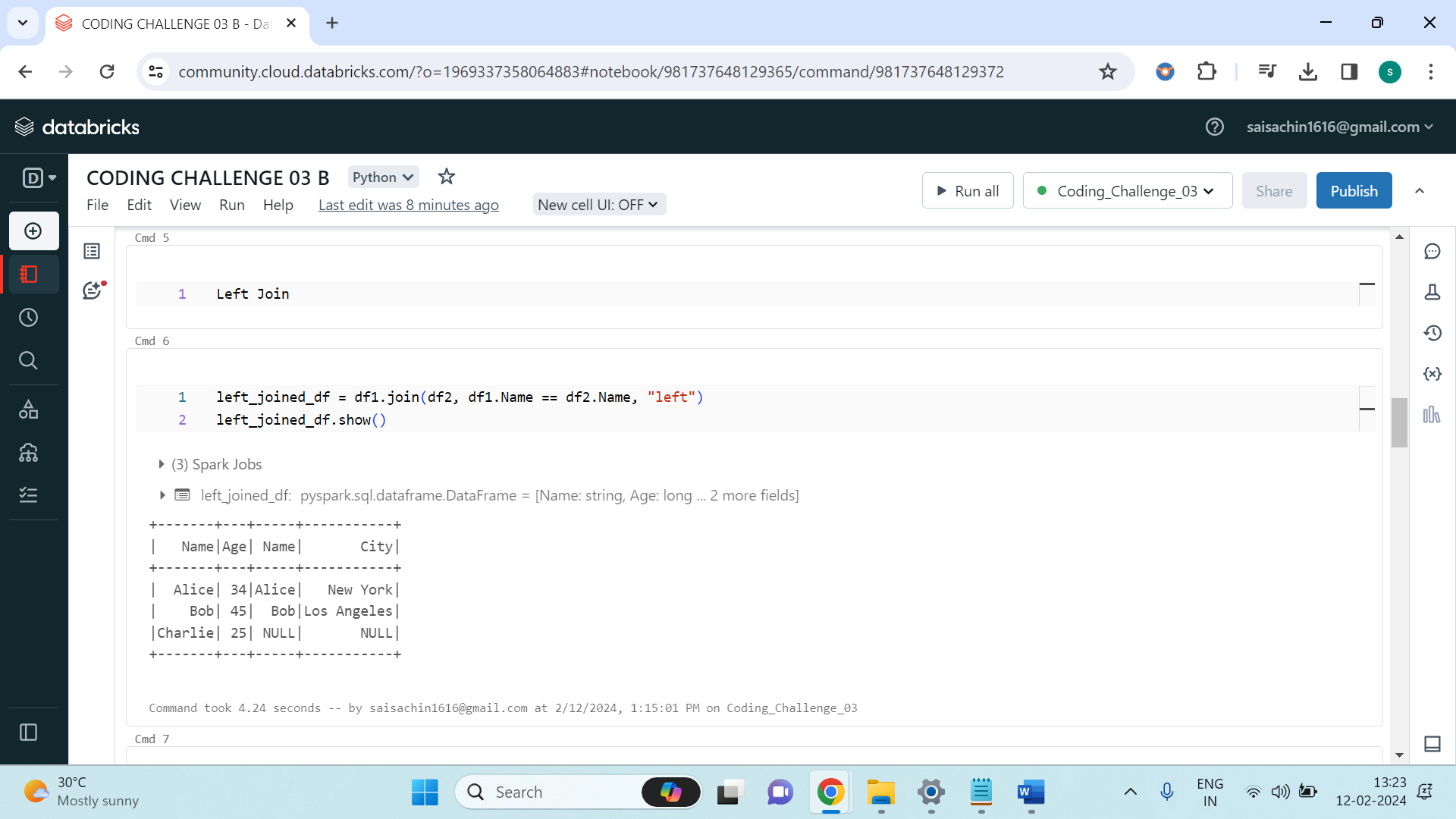


The code proceeds to perform different types of joins: **inner**, **left**, **right**, and **full outer joins**, along with **left semi-join** and **left anti-join**, showcasing the versatility of PySpark SQL in handling data merging operations. After each join operation, the resulting DataFrame is displayed using the **show()** method.

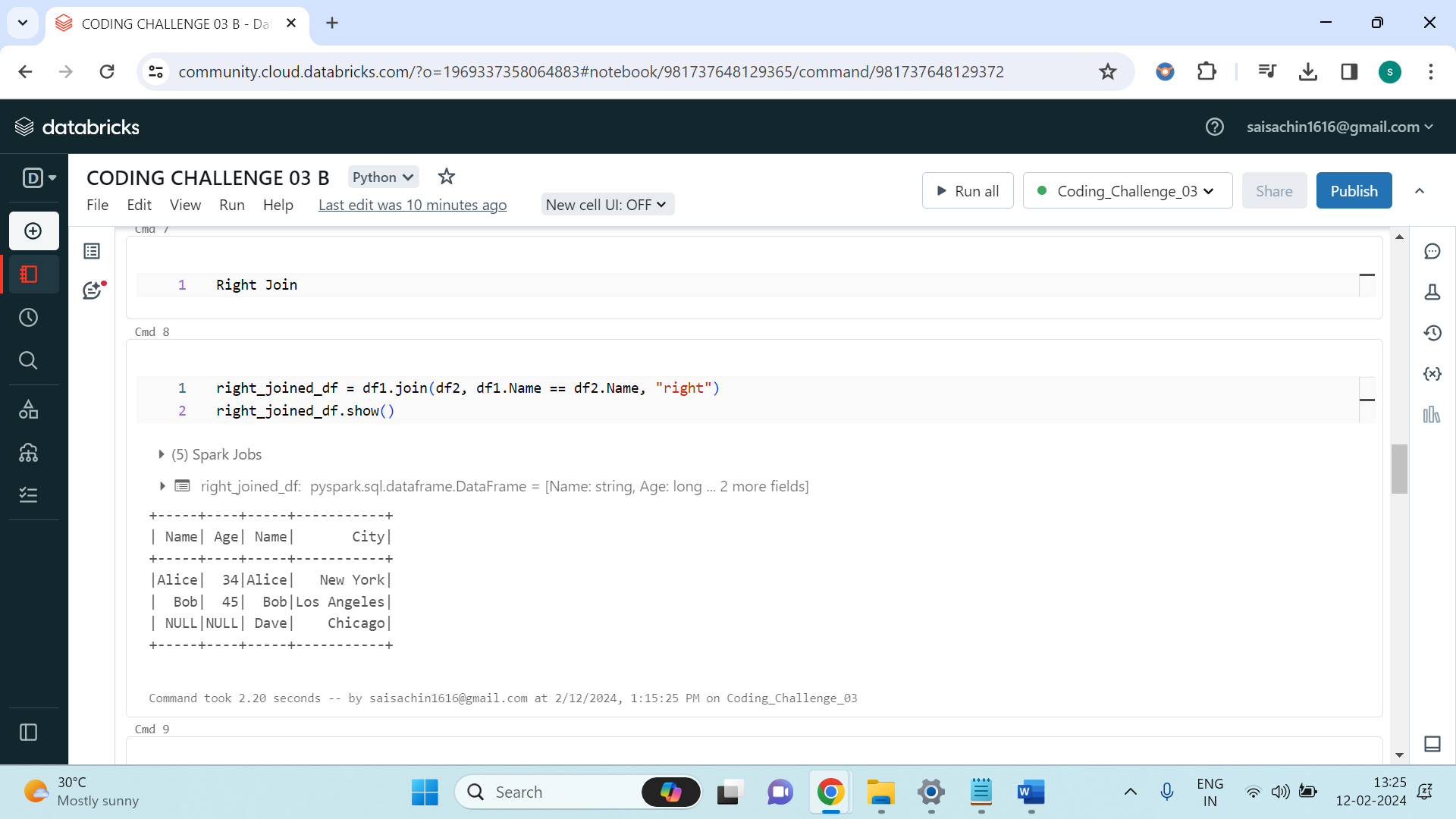
**Inner Join**: An inner join returns only the rows that have matching values in both DataFrames based on the specified condition (**df1.Name == df2.Name**). In the example, only "Alice" and "Bob" have matching names in both DataFrames, so their corresponding information is combined in the resulting DataFrame.



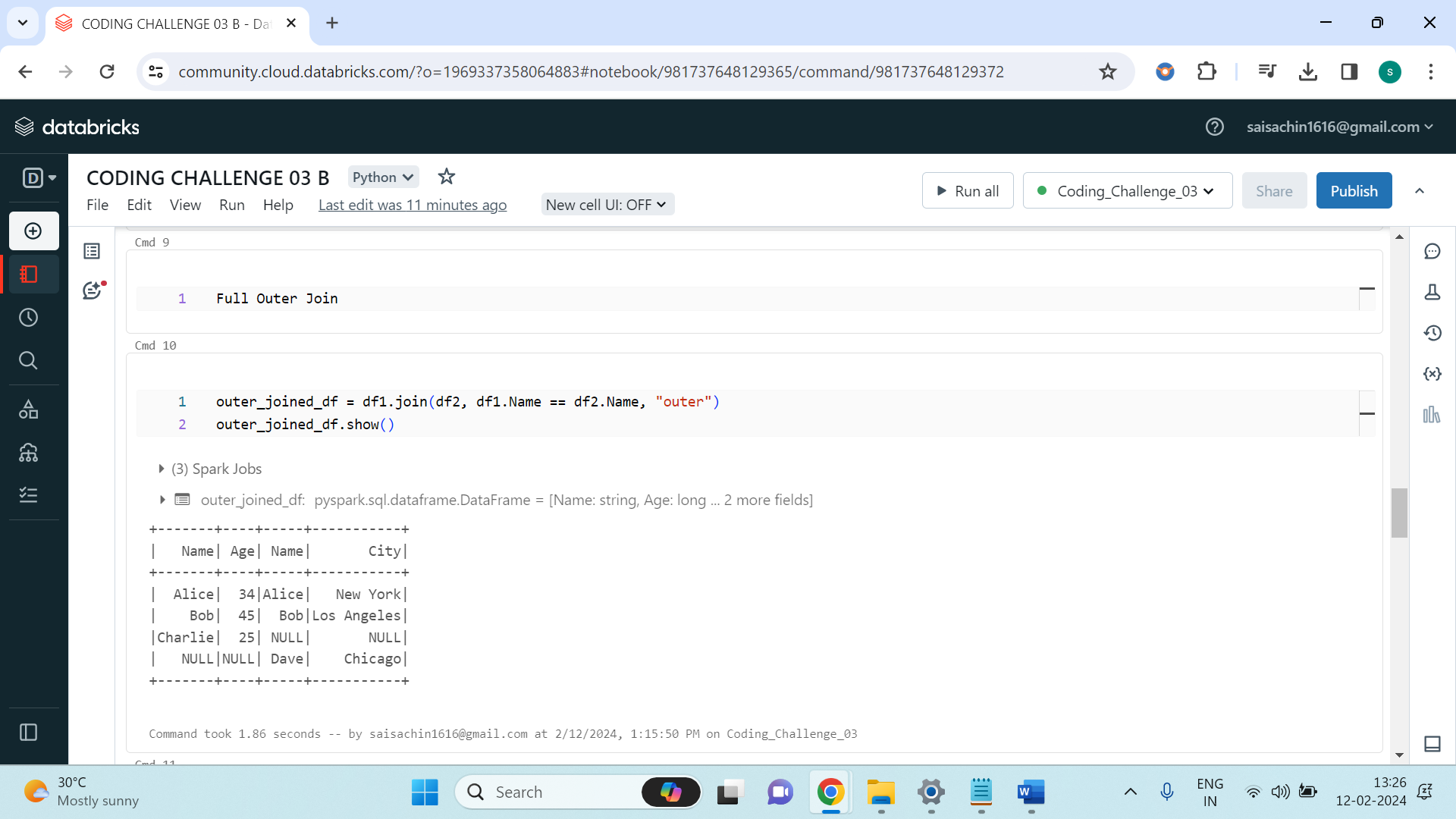
**Left Join**: A left join returns all rows from the left DataFrame (**df1**) and the matching rows from the right DataFrame (**df2**). If there are no matching rows, it fills the missing values in the right DataFrame with nulls. This is useful when you want to keep all records from the left DataFrame, regardless of whether they have matches in the right DataFrame. For instance, "Charlie" from **df1** doesn't have a matching record in **df2**, so "City" will be null for "Charlie" in the resulting DataFrame.



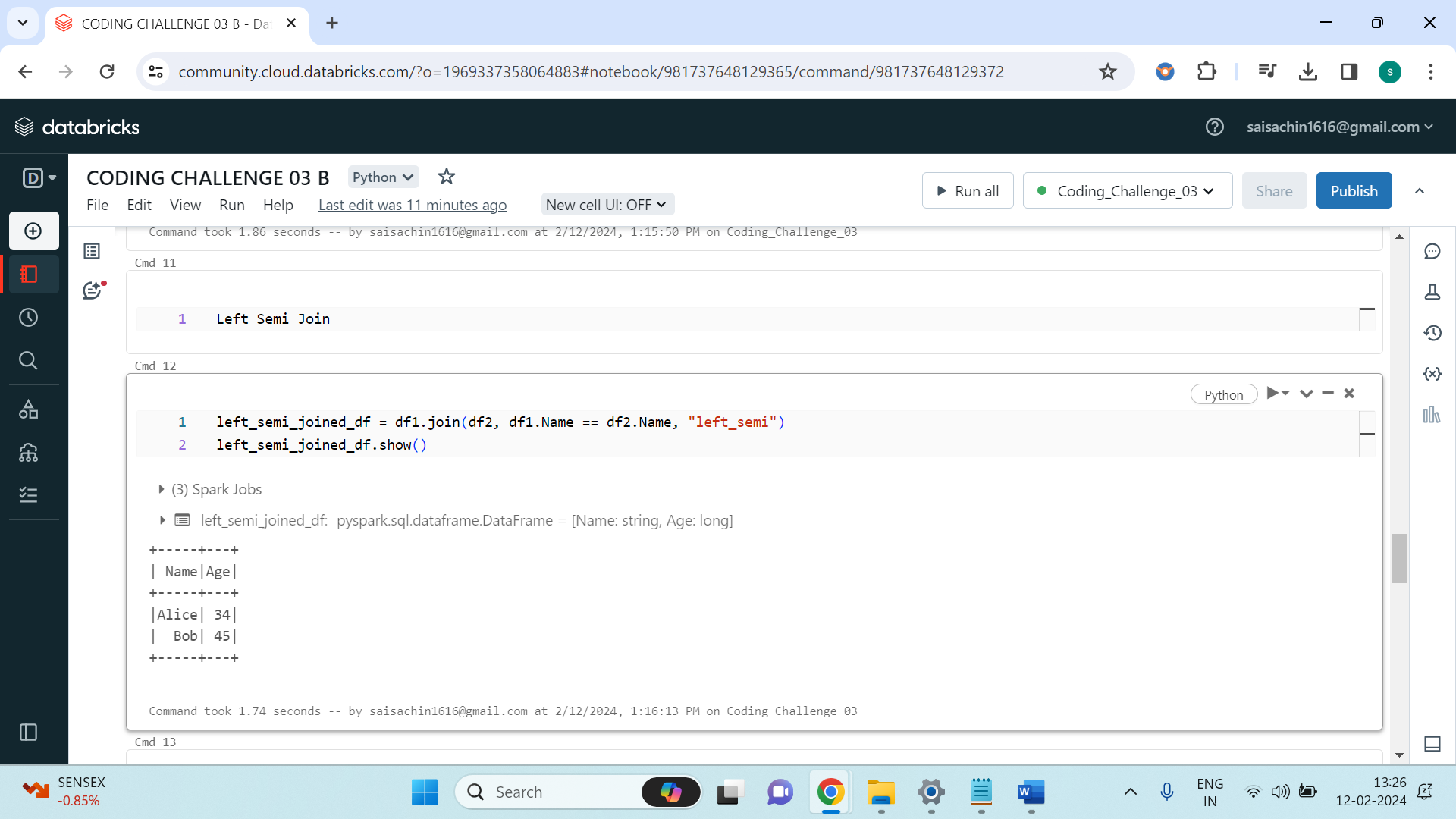
**Right Join**: A right join is similar to a left join, but it returns all rows from the right DataFrame (**df2**) and the matching rows from the left DataFrame (**df1**). Again, if there are no matching rows, it fills the missing values with nulls from the left DataFrame. In the provided example, "Dave" from **df2** doesn't have a match in **df1**, so "Age" will be null for "Dave" in the resulting DataFrame.



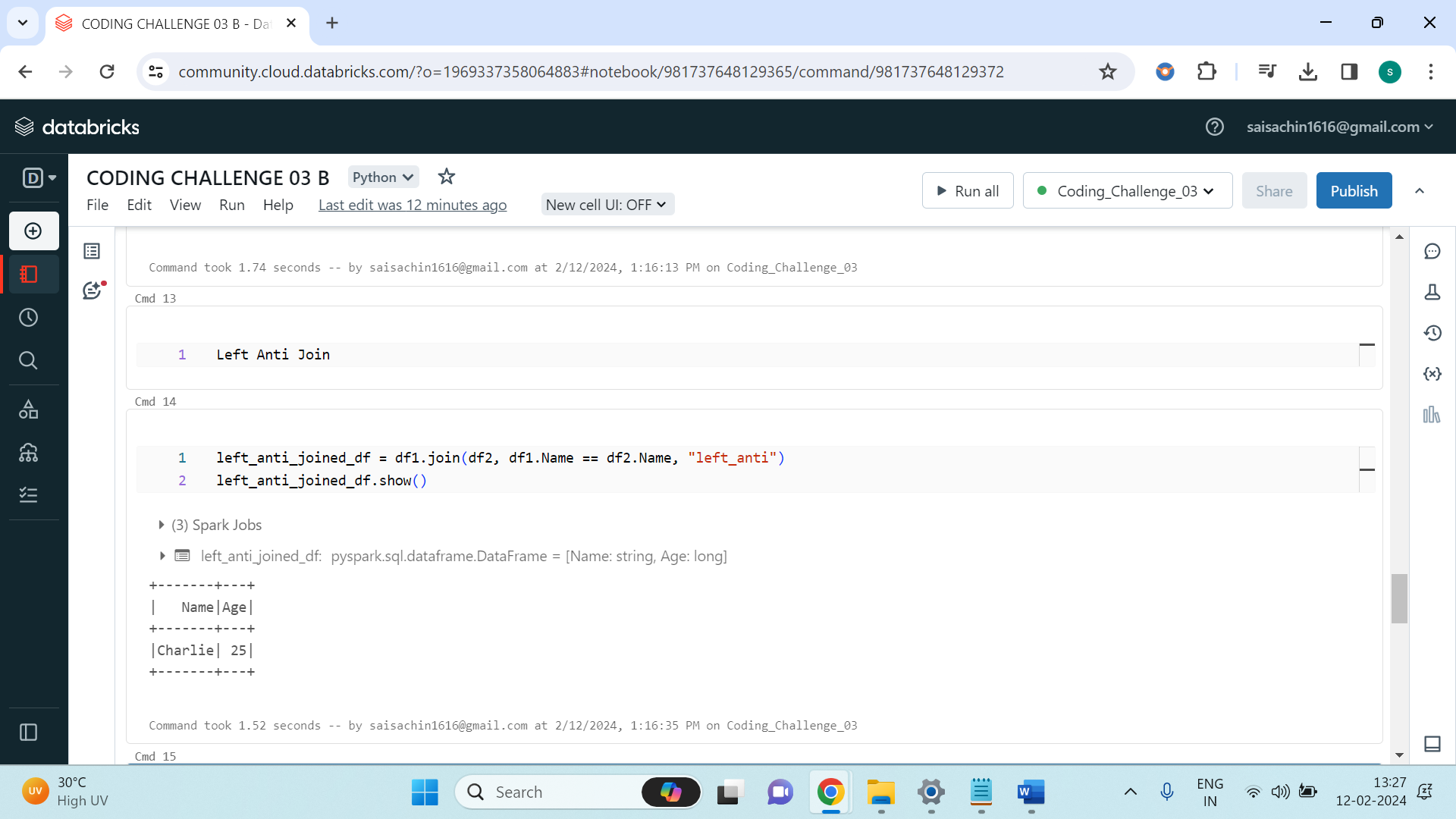
**Full Outer Join**: A full outer join returns all rows from both DataFrames, matching rows from both sides where available, and filling in nulls for missing values. This join type is useful when you want to combine all the information from both DataFrames, regardless of whether there are matches or not.



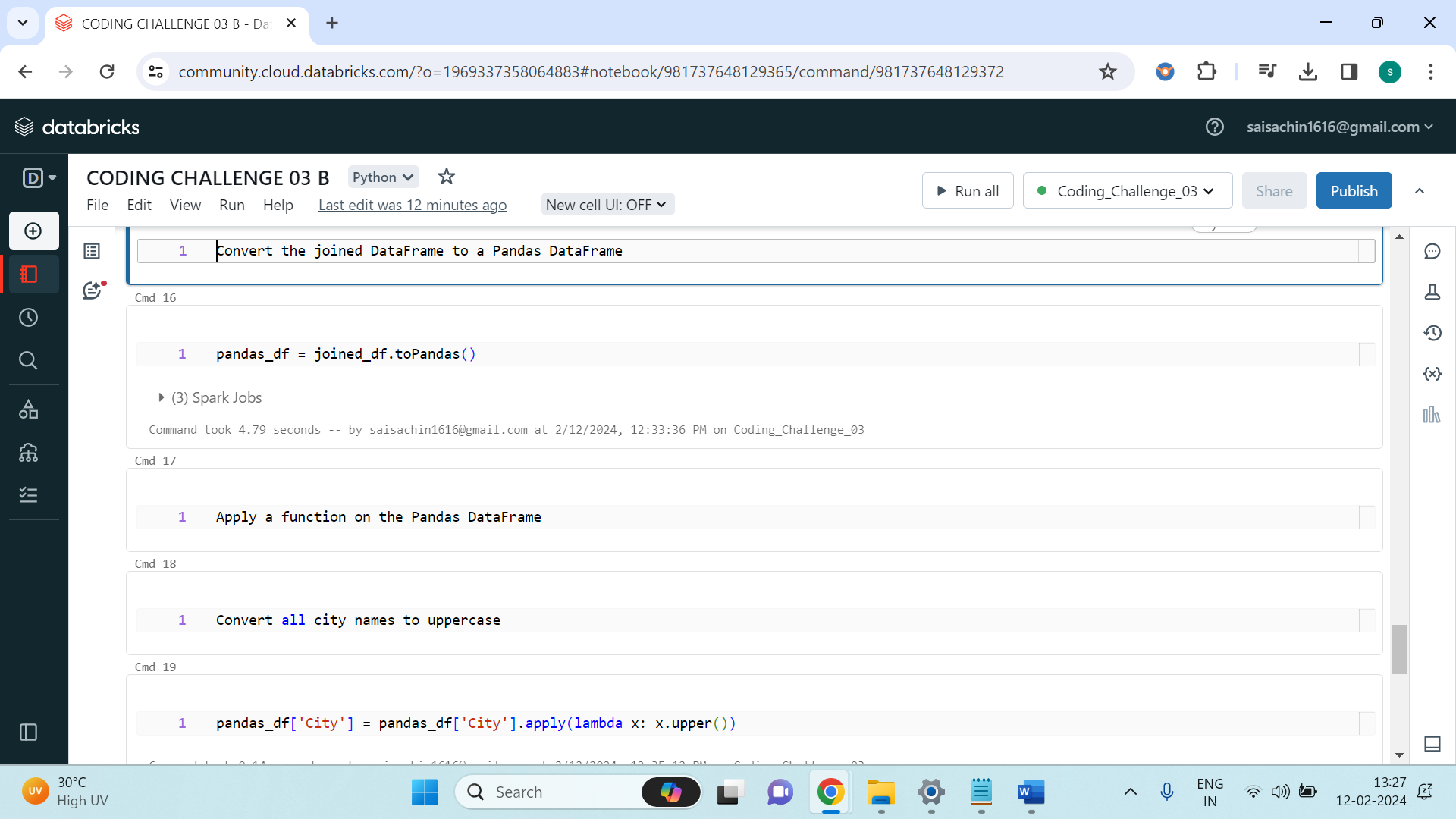
Left Semi Join: A left semi join returns all the rows from the left DataFrame (**df1**) where there is a match in the right DataFrame (**df2**). It doesn't include any columns from the right DataFrame in the result. This join is useful for filtering rows in the left DataFrame based on a condition in the right DataFrame.

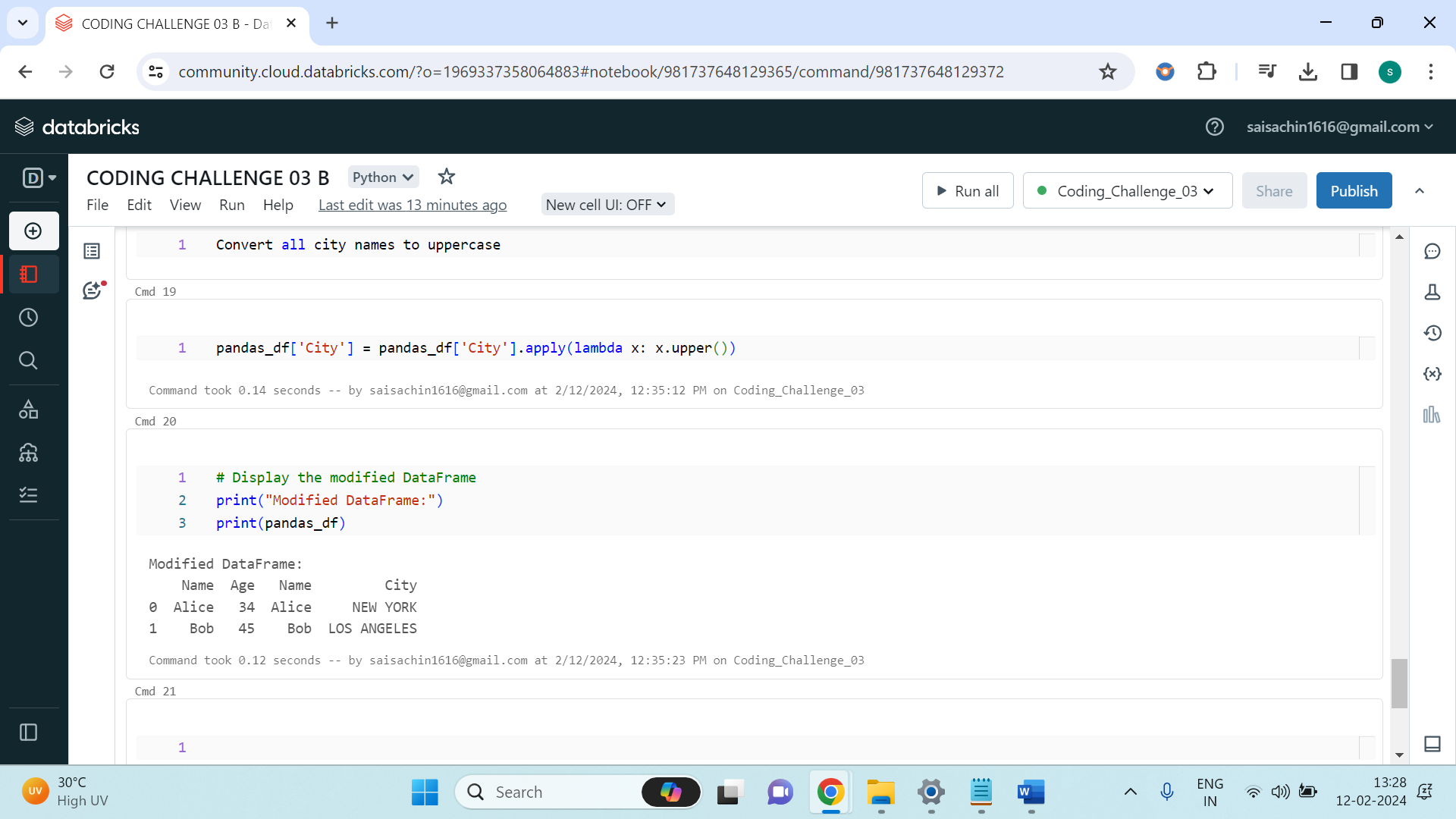


**Left Anti Join**: A left anti join returns all the rows from the left DataFrame (**df1**) where there is no match in the right DataFrame (**df2**). It excludes rows from the left DataFrame that have a match in the right DataFrame. This type of join is useful for finding records in one DataFrame that don't have corresponding records in another DataFrame.



Following the join operations, the code converts the resulting joined DataFrame into a Pandas DataFrame using the toPandas() method. This allows for easier manipulation of the data using Pandas functions. In the provided example, it applies a lambda function to convert all city names to uppercase.





Finally, the modified Pandas DataFrame is printed, demonstrating the successful transformation of data. Lastly, the SparkSession is stopped to release resources.

Overall, the code showcases how to perform various types of joins using PySpark SQL, demonstrates interoperability with Pandas for further data manipulation, and efficiently manages resources by stopping the SparkSession once the operations are completed.