

ADVANCED DATA PROCESSING TECHNIQUES

PROJECT REPORT

AIRBNB DATA ANALYSIS USING DATABRICKS

KAVIN M

RA2412033010050

PROJECT OVERVIEW:

This project primarily focuses on analysing Airbnb listings in **London** using **Databricks**. The goal is to explore trends in room types, pricing, availability, and neighbourhood distributions. Leveraged Apache **Spark** in **Databricks Community Edition** to process large-scale data efficiently and extract valuable insights for better decision-making.

AIRBNB:

Airbnb is an online marketplace that connects travellers with hosts offering accommodations. Founded in 2008, it has transformed the hospitality industry by providing **flexible, cost-effective, and unique lodging options**. The platform includes listings ranging from shared rooms to luxury villas, with millions of hosts and guests worldwide.

DATABRICKS:

Databricks is a unified **cloud-based data analytics platform** built on **Apache Spark**. It enables **big data processing, machine learning, and real-time analytics** with an optimized runtime for faster computation.

In this project, we use **Databricks Community Edition**, which offers a free environment for Spark-based analytics.

DATASET OVERVIEW:

The name of the Dataset is **listings.csv**, which is a csv file contains the information about the accommodating properties and availabilities for the city called **“London, UK”**.

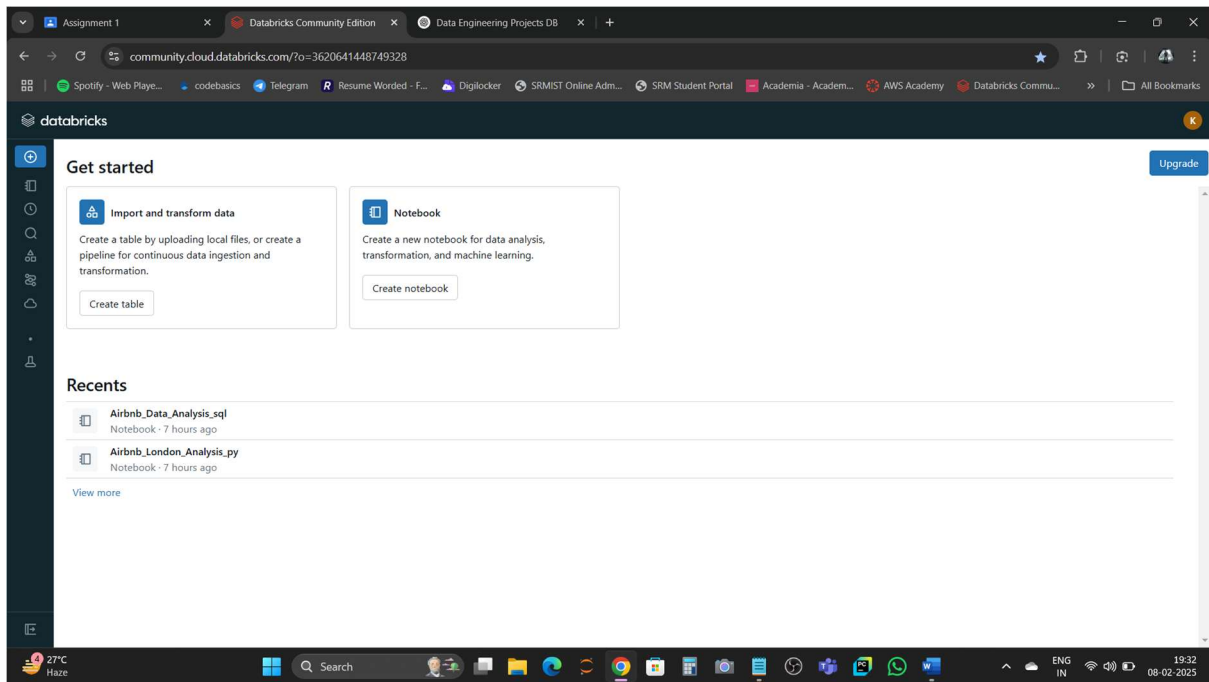
This dataset contains several columns:

- **Id:** Unique listing id.
- **Name:** Name of the listing.
- **Neighbourhood:** Location of the listing.
- **Room_type:** Type of room (e.g., Entire home, Private room, Shared room).
- **Price:** Price per night.
- **minimum_nights:** Minimum stay required
- **number_of_reviews:** Total number of reviews.

PROJECT SETUP:

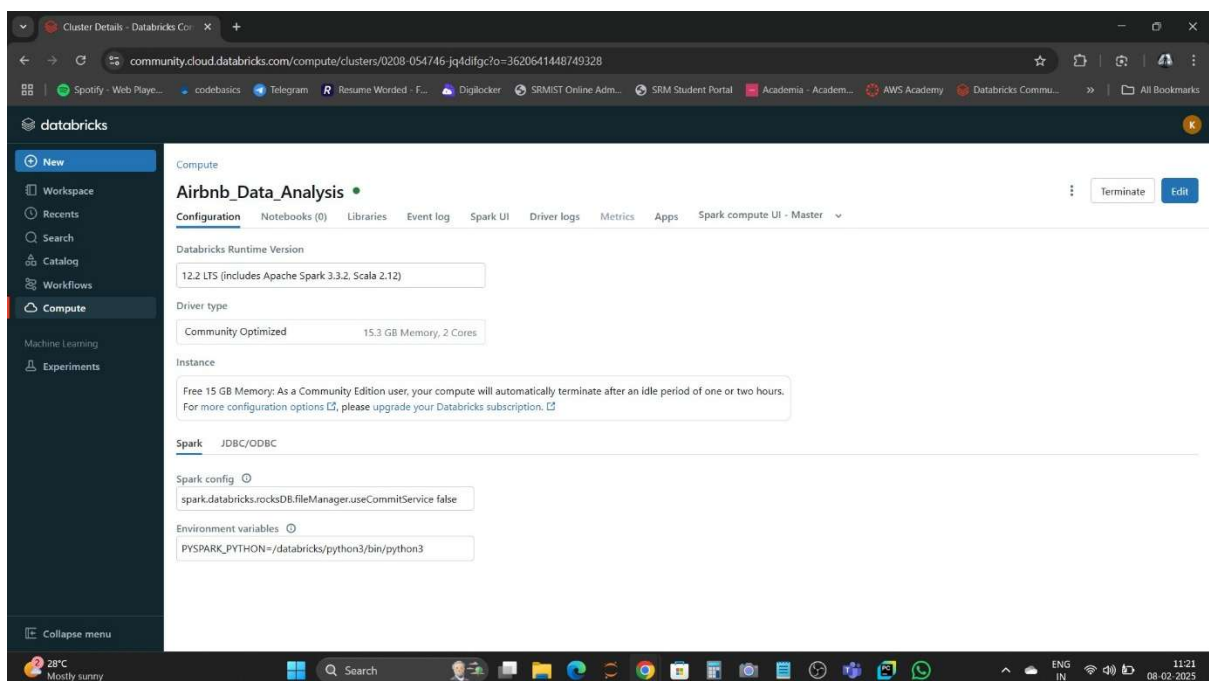
1. Setting Up Databricks Environment:

- Go to Databricks Community Edition
- Sign up and log in.
- Create a new workspace.



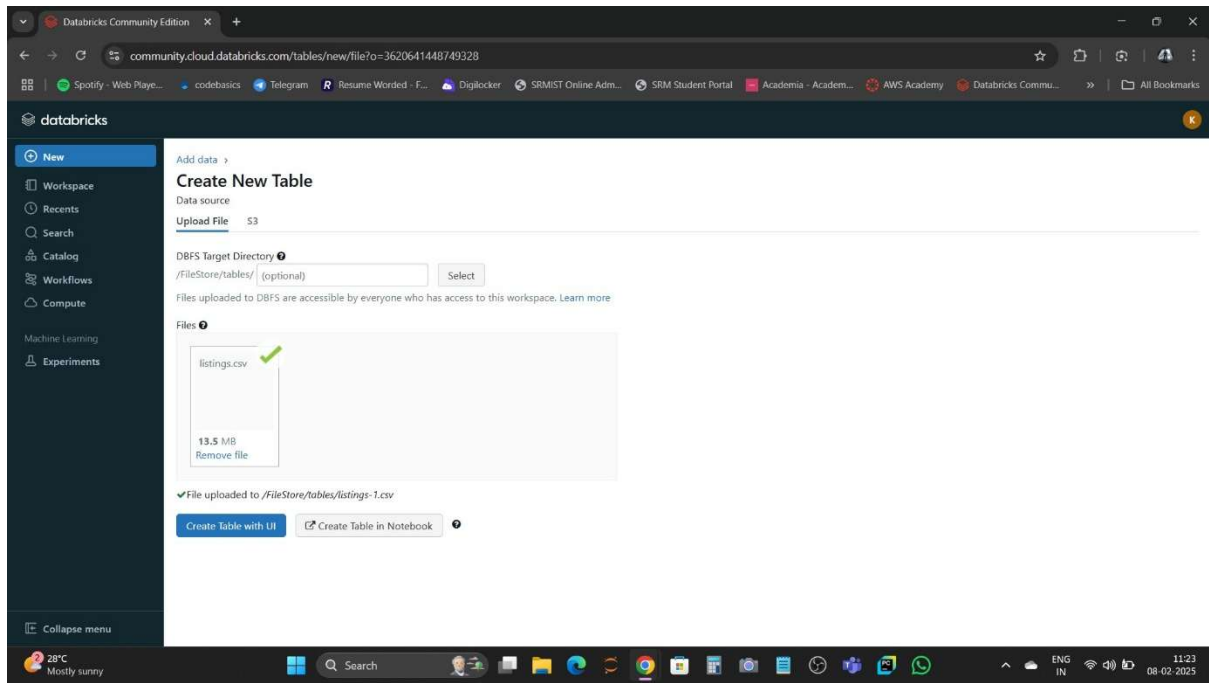
2. Create a New Cluster:

- Click on Compute → Create Cluster.
- Name the cluster (e.g., Airbnb-Analysis).
- Select Single Node Cluster (default).
- Set Runtime Version: Databricks Runtime 12.2 LTS (Scala 2.12, Spark 3.3.2).
- Click Create Cluster and wait for it to start.



3. Download the dataset from the Airbnb site and Upload on Databricks:

- Go to Data → Create Table → Upload listings.csv.
- Select DBFS (Databricks File System) for storage.
- Note the path where the file is stored (e.g., /FileStore/tables/listings.csv).

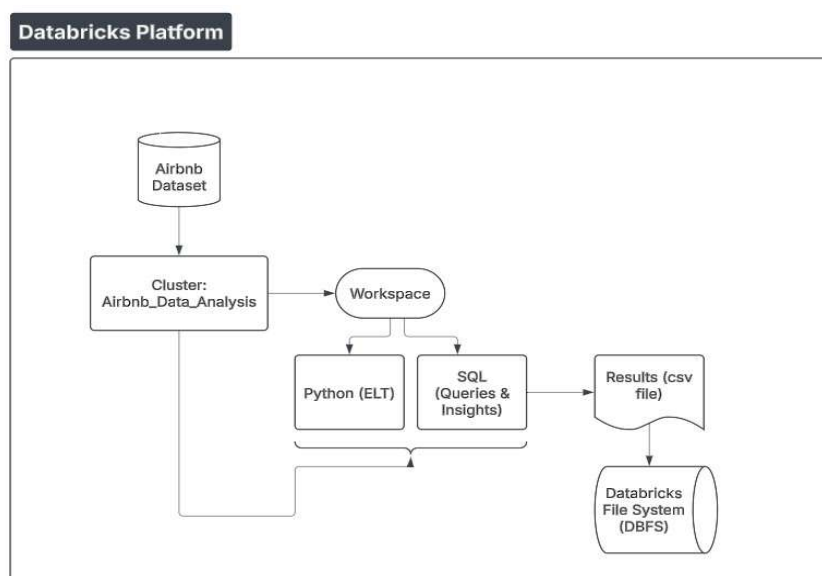


PROJECT DIMENSIONS & WORKFLOW:

Python and SQL are the primary languages used for this project for various purposes.

- Python – For loading, preprocessing and visualizing the data.
- SQL – For generating insights from the data using queries.

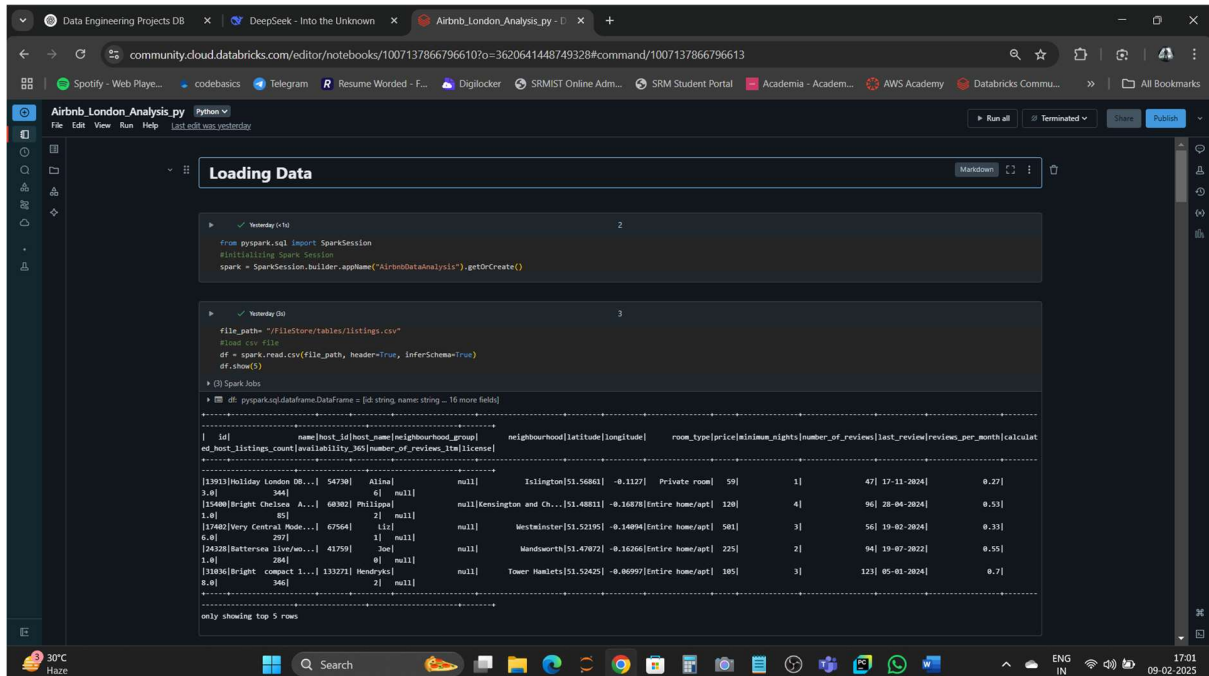
Below flowchart describes the entire project workflow:



IMPLEMENTATION:

Python Notebook:

1. Loading data



```
from pyspark.sql import SparkSession
#initializing Spark Session
spark = SparkSession.builder.appName("AirbnbDataAnalysis").getOrCreate()

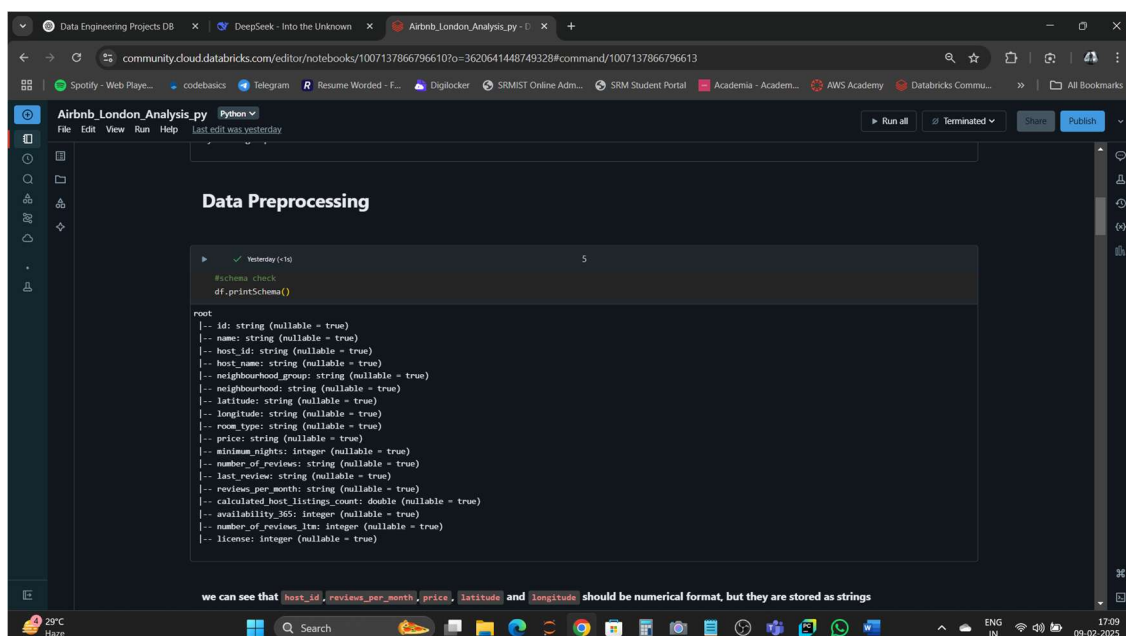
file_path = "/FileStore/tables/listings.csv"
#read csv file
df = spark.read.csv(file_path, header=True, inferSchema=True)
df.show(5)
```

id	name	host_id	host_name	neighbourhood_group	neighbourhood	latitude	longitude	room_type	price	minimum_nights	number_of_reviews	last_review	reviews_per_month	calculated_host_listings_count	availability_365	number_of_reviews_ltm	license
11913	Holiday London DB...	54730	Allina	6	null	Islington	51.56861	-0.1127	Private room	59	1	47	17-11-2024	0.27			
15400	Bright Chelsea A...	68302	Philippa	null	Kensington and Ch...	51.48811	-0.16878	Entire home/apt	120	4	96	28-04-2024	0.53				
17402	Very Central Mode...	67964	liz	11	null	Westminster	51.52195	-0.14094	Entire home/apt	501	3	56	19-02-2024	0.33			
14330	Battersea live-in...	41759	Sam	14	null	Mandsworth	51.47072	-0.16266	Entire home/apt	235	2	94	19-07-2022	0.55			
13836	Bright compact 1...	133271	Hendryks	1	null	Tower Hamlets	51.52425	-0.06997	Entire home/apt	185	3	123	05-01-2024	0.71			

The data has been loaded from the DBFS and the data will initialize the **SparkSession** which allows users to interact with spark functionality, create Dataframes and perform operations.

2. Data Preprocessing

This process involves refining the clumsy data by converting the attributes into their appropriate data types, handling missing values, removing duplicates, evaluating outliers, etc.



```
#schema check
df.printSchema()

root
 |-- id: string (nullable = true)
 |-- name: string (nullable = true)
 |-- host_id: string (nullable = true)
 |-- host_name: string (nullable = true)
 |-- neighbourhood_group: string (nullable = true)
 |-- neighbourhood: string (nullable = true)
 |-- latitude: string (nullable = true)
 |-- longitude: string (nullable = true)
 |-- room_type: string (nullable = true)
 |-- price: string (nullable = true)
 |-- minimum_nights: integer (nullable = true)
 |-- number_of_reviews: string (nullable = true)
 |-- last_review: string (nullable = true)
 |-- reviews_per_month: string (nullable = true)
 |-- calculated_host_listings_count: double (nullable = true)
 |-- availability_365: integer (nullable = true)
 |-- number_of_reviews_ltm: integer (nullable = true)
 |-- license: integer (nullable = true)
```

we can see that **host_id, reviews_per_month, price, latitude and longitude** should be numerical format, but they are stored as strings

we can see that **host_id, reviews_per_month, price, latitude** and **longitude** should be numerical format, but they are stored as strings.

```
#Converting data types
from pyspark.sql.functions import col
df = df.withColumn("host_id", col("host_id").cast("int")) \
        .withColumn("latitude", col("latitude").cast("double")) \
        .withColumn("longitude", col("longitude").cast("double")) \
        .withColumn("price", col("price").cast("float")) \
        .withColumn("reviews_per_month", col("reviews_per_month").cast("float"))
df.printSchema()

root
 |-- id: string (nullable = true)
 |-- name: string (nullable = true)
 |-- host_id: integer (nullable = true)
 |-- host_name: string (nullable = true)
 |-- neighbourhood_group: string (nullable = true)
 |-- neighbourhood: string (nullable = true)
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)
 |-- room_type: string (nullable = true)
 |-- price: float (nullable = true)
 |-- minimum_nights: integer (nullable = true)
 |-- number_of_reviews: string (nullable = true)
 |-- last_review: string (nullable = true)
 |-- reviews_per_month: float (nullable = true)
 |-- calculated_host_listings_count: double (nullable = true)
 |-- availability_365: integer (nullable = true)
 |-- number_of_reviews_ltm: integer (nullable = true)
 |-- license: integer (nullable = true)
```

Handling missing values and removing duplicates,

```
 |-- last_review: string (nullable = true)
 |-- reviews_per_month: float (nullable = true)
 |-- calculated_host_listings_count: double (nullable = true)
 |-- availability_365: integer (nullable = true)
 |-- number_of_reviews_ltm: integer (nullable = true)
 |-- license: integer (nullable = true)

#Handling missing values
from pyspark.sql.functions import count, when
#counting missing values in each column
df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns]).show()

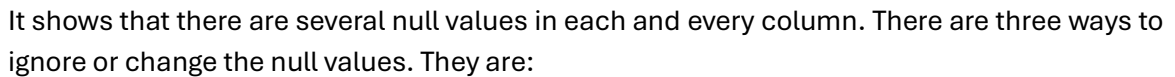
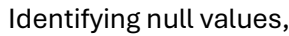
(2) Spark jobs

+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | name | host_id | host_name | neighbourhood_group | neighbourhood | latitude | longitude | room_type | price | minimum_nights | number_of_reviews | last_review | reviews_per_month | calculated_host_listings_count | availability_365 | number_of_reviews_ltm | license |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 0 | 43 | 572 | 569 | 95197 | 322 | 322 | 563 | 367 | 32634 | 314 | 373 | 24894 | 24843 | 315 | 313 | 563 | 95447 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

#Removing duplicates
df=df.dropDuplicates()

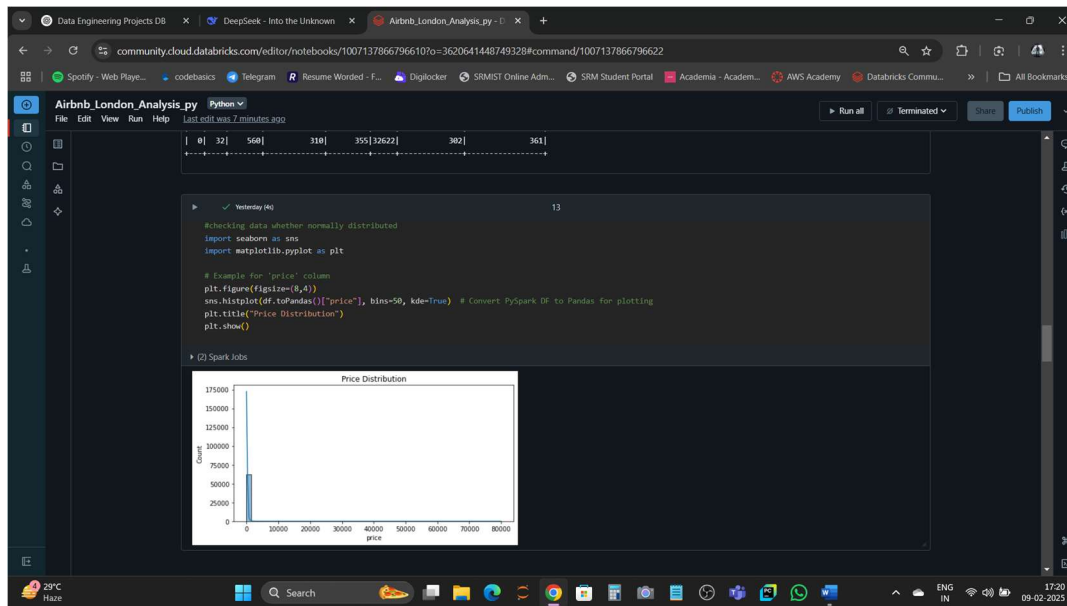
df: pyspark.sql.dataframe.DataFrame = [id: string, name: string ... 16 more fields]
```

3. Feature Engineering,



- Mean – If there normal distribution.
- Median – If there any outliers.
- Fill with 0 – If data has any contextual meaning.

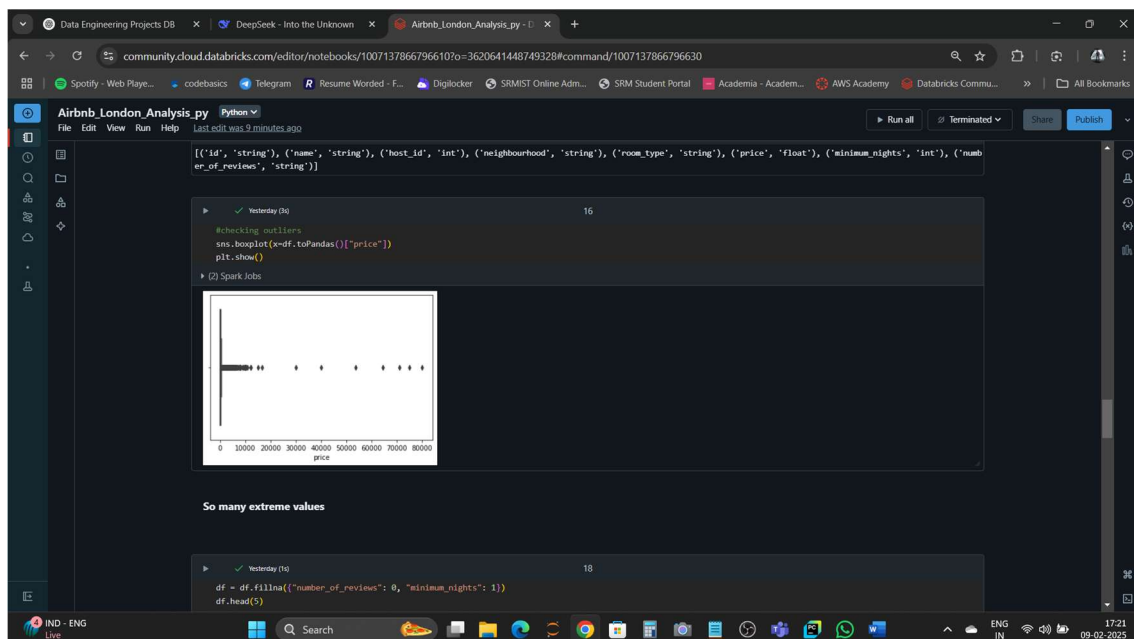
Checking whether the data is normally distributed,



1. If the histogram is **bell-shaped**, the data is likely normally distributed.
2. If it's skewed **left** or **right**, the data is not normally distributed.

The data has been skewed left in our case. Hence it is not normally distributed.

Checking outliers,



So many extreme values.

By Scaling,


```

df = df.filter("number_of_reviews > 0, minimum_nights > 1")
df.head(5)

df.show()

```

id	name	host_id	neighbourhood	room_type	price	minimum_nights	number_of_reviews
11836	Bright compact 1 Bedroom Apartment Brick Lane	133271	Tower Hamlets	Entire home/apt	185.0	3	11
18098	Room 1 Large Double Room	167387	Haringey	Private room	55.0	11	11
142818	You Will Save Mon...	157884	Barnet	Private room	65.0	2	604
124128	Battersea live/work...	41759	Wandsworth	Entire home/apt	125.0	2	94
142611	Stunning flat w...	186518	Hammermith and F...	Private room	null	1	0

Checking Description of the data,

```

df.describe().show()

```

summary	id	name	host_id	neighbourhood	room_type	price	minimum_nights	number_of_reviews
count	95424	95392	94864	95114	95069	62882	95424	95424
mean	5.77848812452083...	2.419262616459144E8	1.9381488081724574E8	51.513505517880806	127.67537839255122	286.61467787658074	5.589757293762576	19.97991263624336
stddev	5.3042768640244991...	2.078667934960383E8	2.0875356728620583E8	0.05830548053115835	114.44250701265062	774.6170449673841	23.23442670458596	47.78966119446791
min	home london...	1 Sofa bed	2594	51.33782843	-0.066388158	1.0	0	0
max	zone 2"	Kingsize room ...	666236454	Westminster	Shared room	80000.0	1125	99

4. Data Visualization

Visualizing the top 4 room types in the London.

```

Airbnb London Analysis.py Python
File Edit View Run Help Last edit was 16 minutes ago Run all Terminated Share Publish

Data Visualization

Top 4 Room Types

Python
import matplotlib.pyplot as plt
import pandas as pd

# Convert to Pandas only if it's a PySpark DataFrame
if not isinstance(df, pd.DataFrame):
    df = df.toPandas()

# Check and fix column names
print("Columns:", df.columns) # Debugging step

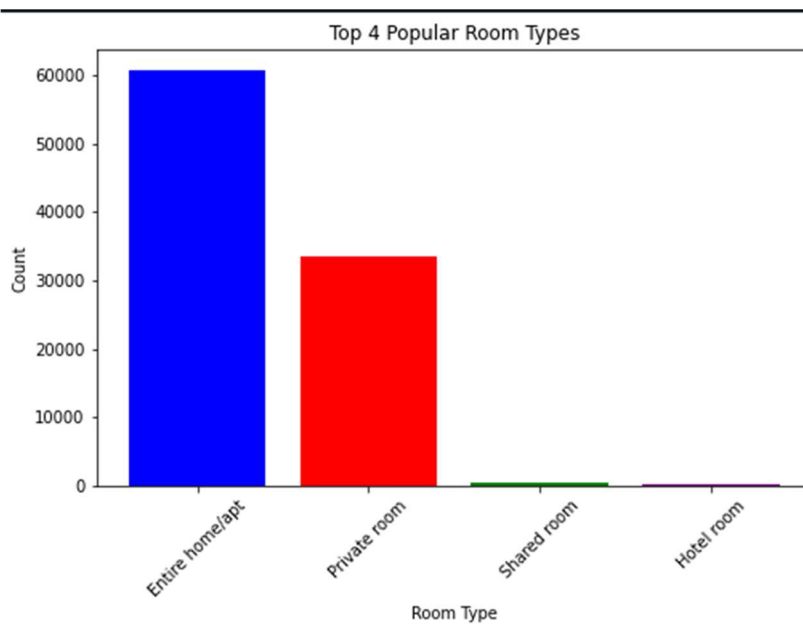
if "count" not in df.columns:
    df = df.groupby("room_type").size().reset_index(name="count") # Aggregate if missing

# Ensure count is numeric
df["count"] = pd.to_numeric(df["count"], errors="coerce").fillna(0)

# Select only the top 4 room types
df_top4 = df.nlargest(4, "count") # Select top 4 based on count

# Plot bar chart
plt.figure(figsize=(8, 5))
plt.bar(df_top4["room_type"], df_top4["count"], color=["blue", "red", "green", "purple"])
plt.xlabel("Room Type")
plt.ylabel("Count")
plt.title("Top 4 Popular Room Types")
plt.xticks(rotation=45)
plt.show()

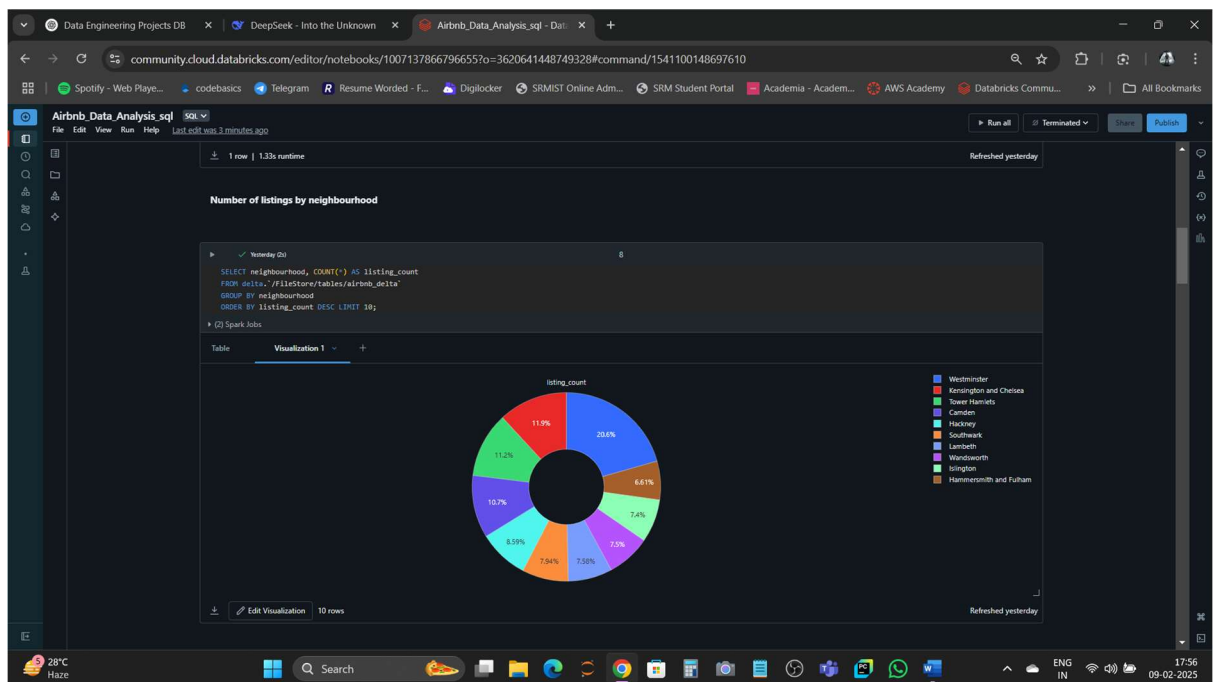
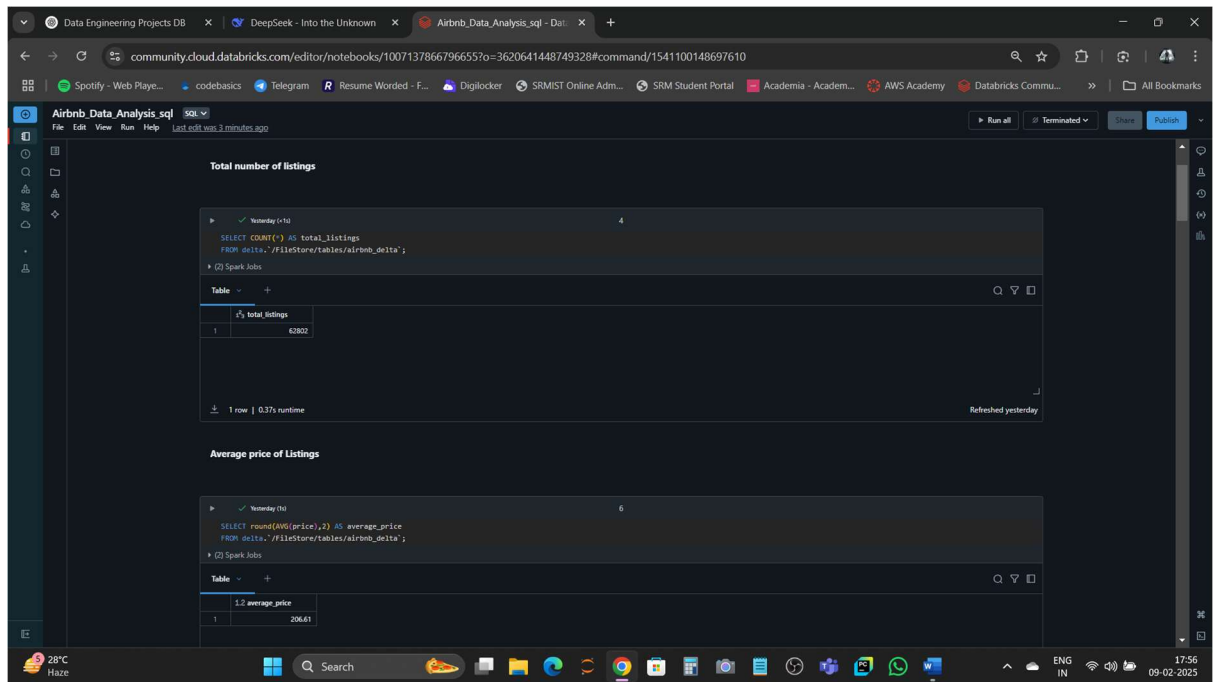
```



SQL Notebook:

This notebook mainly deals about the further insights such as **Listing Analysis**, **Price analysis**, **Host Analysis**, **Occupancy and Availability analysis** and eventually **Correlation analysis** generated from the data.

- **Listing Analysis:** Total number of listings, Average price of listings, Number of listings by neighbourhood.



- **Price Analysis:** Top expensive listings, Average price by room type.

Airbnb Data Analysis.sql

Price Analysis

Top 4 expensive listings

```
SELECT name, neighbourhood, price
FROM delta.`filestore/tables/airbnb_delta`
ORDER BY price DESC
LIMIT 4;
```

#	name	neighbourhood	price
1	Room In Zone 1 (TDR)	Southwark	£80,000
2	Close To London Eye (HED)	Lambeth	£75,000
3	Short Walk To London Eye (SUR)	Lambeth	£71,000
4	CLOSE TO LONDON EYE (CHECZ)	Lambeth	£64,296

4 rows | 2.00s runtime

Airbnb Data Analysis.sql

Average price by Room type

```
SELECT room_type, round(AVG(price),2) as avg_price
FROM delta.`filestore/tables/airbnb_delta`
GROUP BY room_type
ORDER BY avg_price desc LIMIT 4;
```

#	room_type	avg_price
1	Hotel room	£560
2	Entire home/apt	£254
3	Shared room	£144
4	Private room	£110

4 rows | 2.06s runtime

- **Host Analysis: Average price by host.**

Airbnb Data Analysis.sql

Host Analysis

Average price by host

```
SELECT host_id, round(AVG(price),2) as avg_price
FROM delta.`filestore/tables/airbnb_delta`
GROUP BY host_id
ORDER BY avg_price DESC
LIMIT 5;
```

#	host_id	avg_price
1	34349317	53588
2	5391456	30000
3	489477288	15000
4	144786680	14555.22
5	555237030	10285

5 rows | 2.42s runtime

- **Occupancy and Availability Analysis:** Top listings with high prices but low reviews, Most popular room types in each neighbourhood.

Airbnb Data Analysis.sql

File Edit View Run Help last edit was 11 minutes ago

Run all Terminated Share Publish

Occupancy and Availability Analysis

Top 5 Listings with High prices but Low reviews

```

SELECT name, neighbourhood, price, number_of_reviews
FROM delta.`/filestore/tables/airbnb_delta`
WHERE price > (SELECT AVG(price) FROM delta.`/filestore/tables/airbnb_delta`)
AND number_of_reviews < (SELECT AVG(number_of_reviews) FROM delta.`/filestore/tables/airbnb_delta`)
ORDER BY price DESC LIMIT 5;

```

19

Table

#	name	neighbourhood	price	number_of_reviews
1	Room In Zone 1 (TOB)	Southwark	£80,000	12
2	Close To London Eye (RED)	Lambeth	£75,000	1
3	Short Walk To London Eye (SUR)	Lambeth	£71,000	2
4	CLOSE TO LONDON EYE (CHECZ)	Lambeth	£64,296	2
5	Central Room - Walk to Eye (XR)	Lambeth	£40,000	0

5 rows | 2.88s runtime

Refreshed yesterday

Airbnb Data Analysis.sql

File Edit View Run Help last edit was 12 minutes ago

Run all Terminated Share Publish

Most Popular Room Types in Each Neighborhood

```

SELECT neighbourhood, room_type, count(*) listing_count
FROM delta.`/filestore/tables/airbnb_delta`
WHERE room_type IS NOT NULL
GROUP BY neighbourhood, room_type
ORDER BY listing_count DESC LIMIT 5;

```

21

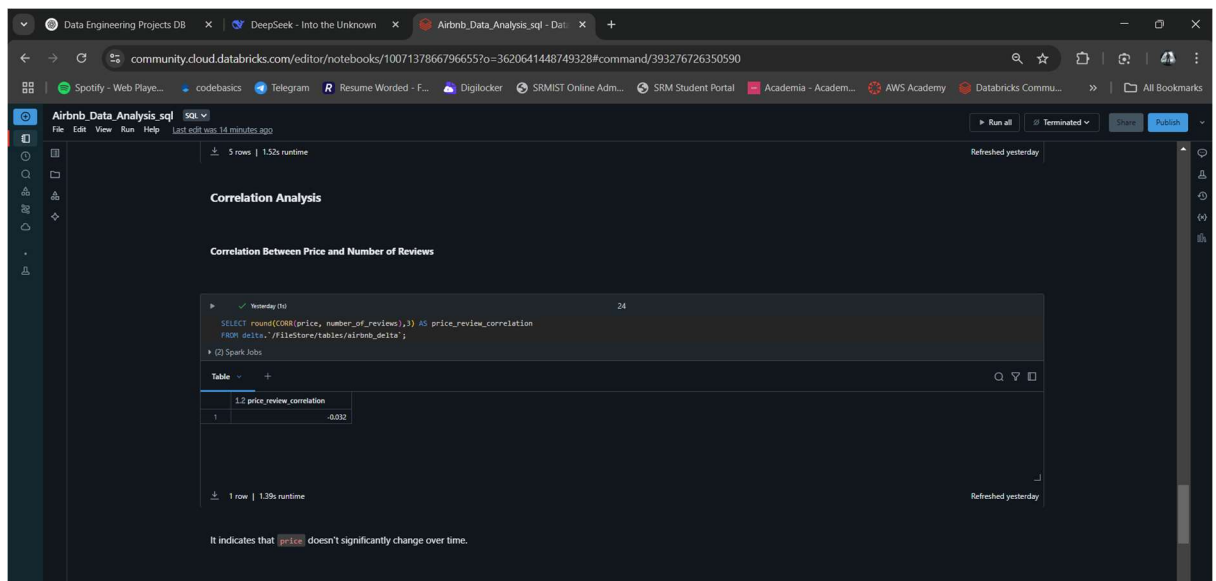
Table

#	neighbourhood	room_type	listing_count
1	Westminster	Entire home/apt	6655
2	Kensington and Chelsea	Entire home/apt	4074
3	Camden	Entire home/apt	3095
4	Tower Hamlets	Entire home/apt	2810
5	Hackney	Entire home/apt	2417

5 rows | 1.52s runtime

Refreshed yesterday

Correlation Analysis: Correlation between Price and Number of reviews, Correlation between Price and Minimum nights.



Correlation Analysis

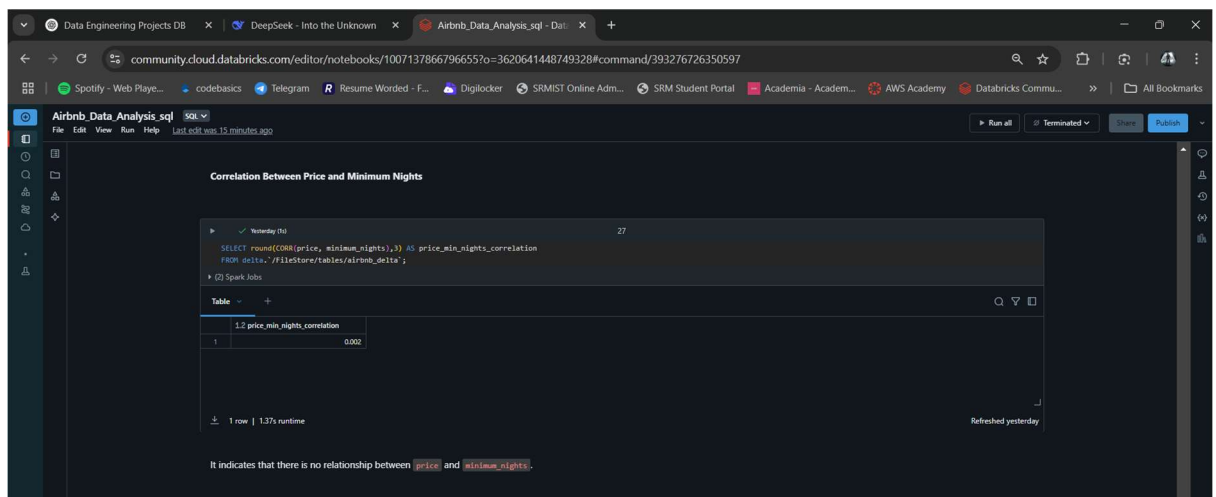
Correlation Between Price and Number of Reviews

```
SELECT round(CORR(price, number_of_reviews), 3) AS price_review_correlation
FROM delta.`filestore/tables/airbnb_delta`;
```

price_review_correlation
-0.032

It indicates that **price** doesn't significantly change over time.

It indicates that **price** doesn't significantly change over time.



Correlation Between Price and Minimum Nights

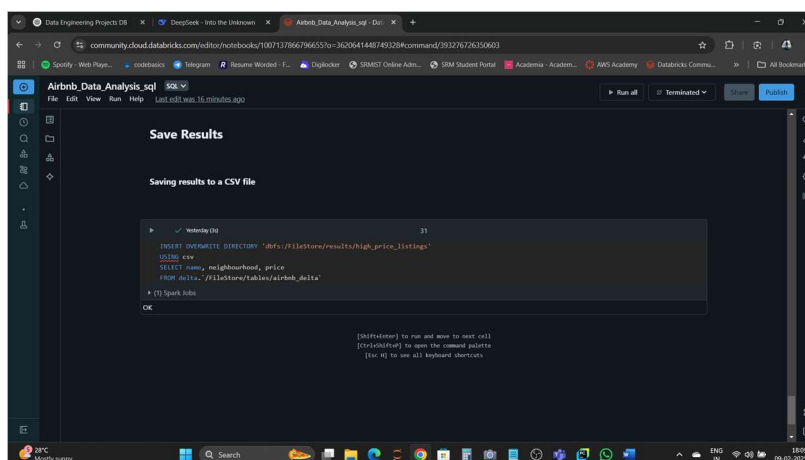
```
SELECT round(CORR(price, minimum_nights), 3) AS price_min_nights_correlation
FROM delta.`filestore/tables/airbnb_delta`;
```

price_min_nights_correlation
0.002

It indicates that there is no relationship between **price** and **minimum_nights**.

It indicates that there is no relationship between **price** and **minimum_nights**.

Saving Results:

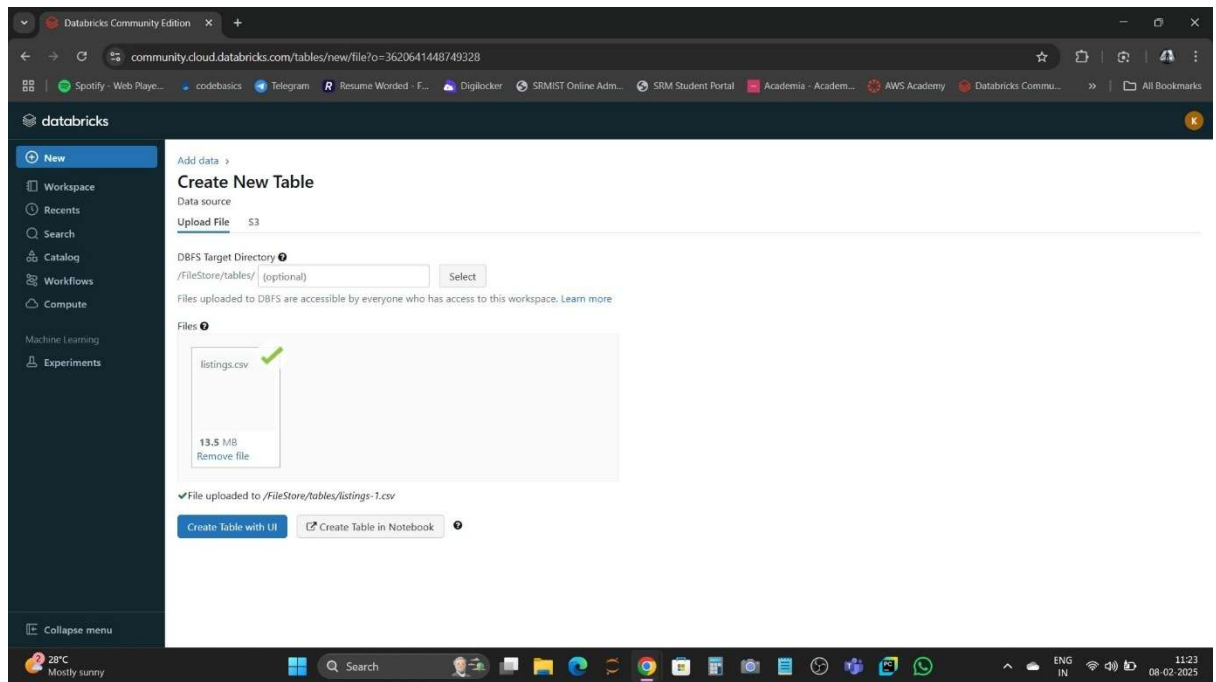


Save Results

Saving results to a CSV file

```
INSERT OVERWRITE DIRECTORY 'dbfs:/filestore/results/high_price_listings'
FORMAT CSV
SELECT name, neighbourhood, price
FROM delta.`filestore/tables/airbnb_delta`;
```

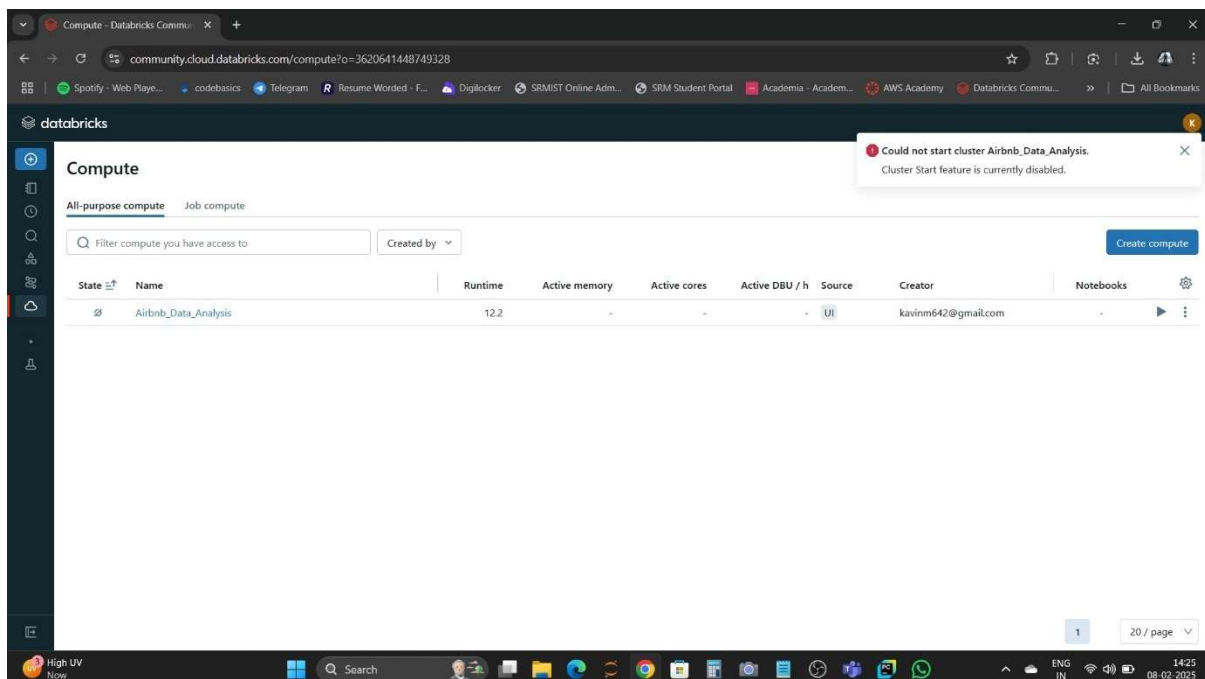
OK



The query extracted Airbnb listings (name, neighbourhood and price). It overwrote any existing files in **high_price_listings**. And finally, the output saved as a csv file which can be viewed and accessed on DBFS (i.e.,) Data Bricks File Store.

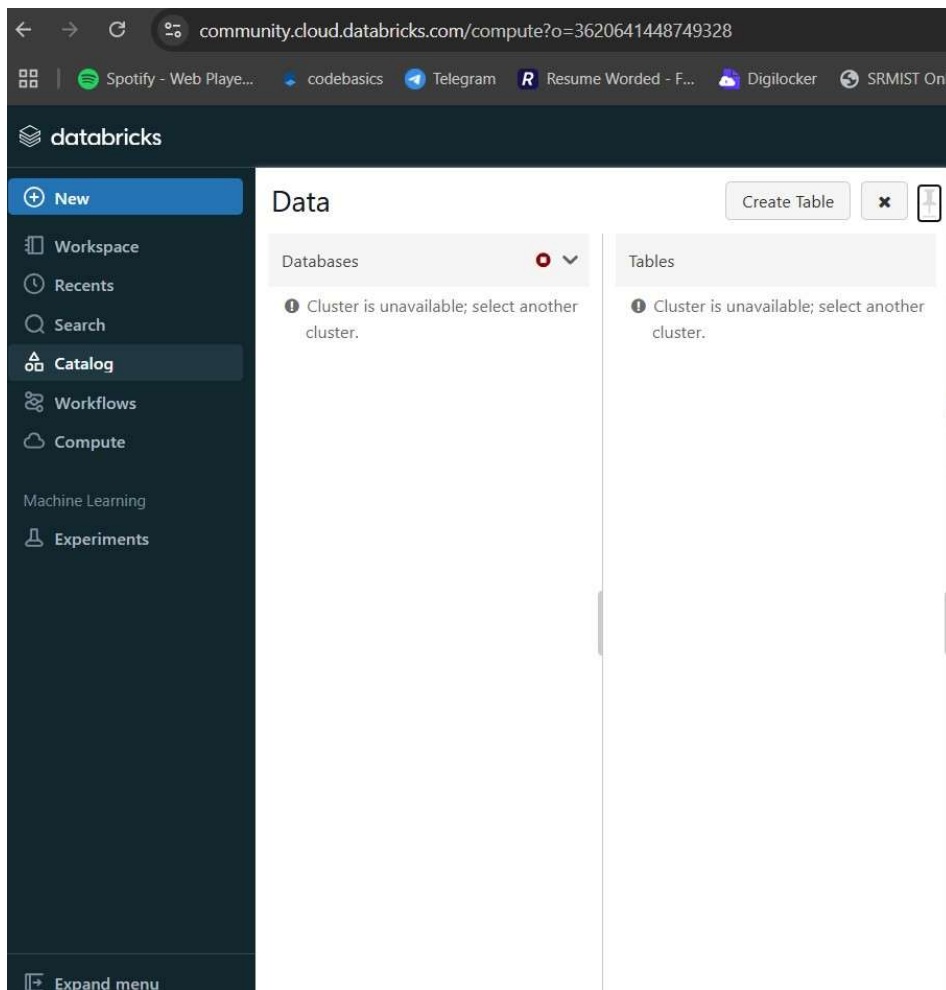
CLUSTER STATUS:

Once the work has been completed, the cluster should be terminated as it consumes more storage and cost if it is a community edition. One major constraint in this community edition is we have certain restrictions such as denial of workflow access, can't restart the cluster once it was terminated and our notebooks will be disabled.



It disabled the cluster as it was terminated once.

Also, we can't able to view the saved report as it will be deleted once the cluster becomes disabled.



PROJECT OUTCOME:

Key findings:

- **Most common room type:** (e.g., Private Room dominates the market).
- **Average price trends:** How prices vary across different neighborhoods.
- **Top neighborhoods:** Areas with the highest number of listings.
- **Availability insights:** Understanding booking patterns.

CONCLUSION:

This analysis provides valuable insights into the Airbnb market in London. Hosts can optimize pricing strategies, while travelers can find budget-friendly locations. Additionally, investors and policymakers can leverage this data to understand market dynamics and improve regulations.

FUTURE WORKS:

- Leverage Machine Learning models to predict price fluctuations.
- Exploring Sentiment Analysis on Airbnb reviews.
- Extend the analysis to include seasonality trends.

LINKS:

- Dataset Link: <https://data.insideairbnb.com/united-kingdom/england/london/2024-12-11/data/listings.csv.gz>
- Python Notebook: <https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3620641448749328/1007137866796610/7261851981964810/latest.html>
- SQL Notebook: <https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/3620641448749328/1007137866796655/7261851981964810/latest.html>