

INTERIM PROJECT

Topic: Streaming ETL: Perform Extract-Transform-Load (ETL) operations on streaming data, transforming it into a more structured format or loading it into a database.



Submitted by

Sanjeev M

EMP ID: 2320436

Abstract:

Streaming Extract-Transform-Load (ETL) operations have become essential in modern data processing pipelines, especially with the rise of real-time data analytics and decision-making. This paper explores the concept of Streaming ETL, which involves extracting data from continuous streams, transforming it on-the-fly, and loading it into a structured format or database for further analysis. The primary objective of Streaming ETL is to handle high-volume, high-velocity data streams efficiently, ensuring timely and accurate data processing.

The abstract further delves into the key components and challenges of Streaming ETL. It discusses the importance of stream processing frameworks like Apache Kafka, Apache Flink, and Apache Storm for handling streaming data ingestion and processing. Additionally, it highlights the significance of scalable data transformation techniques, such as stream-based transformations using Apache Beam or Spark Streaming, to handle complex data manipulations in real-time.

Furthermore, the abstract touches upon the role of destination systems such as relational databases, data warehouses, or data lakes in storing and organizing the transformed data for downstream applications. It emphasizes the need for robust error handling, data quality checks, and monitoring mechanisms in Streaming ETL workflows to ensure data integrity and reliability.

Introduction:

In today's data-driven world, the ability to process data in both real-time and batch modes is crucial for organizations aiming to extract valuable insights and make informed decisions. Streaming Extract-Transform-Load (ETL) and batch ETL are two essential methodologies that enable organizations to process data continuously as it flows in real-time or in larger, periodic batches. These methodologies involve extracting raw data, transforming it to a more structured format, and loading it into a database or data warehouse for further analysis.

Table of Contents

CHAPTER	CONTENT
1	Introduction
2	Data Ingestion
3	Batch processing Using ETL
4	Stream Processing Using ETL

Overview:

Streaming ETL (Real-Time):

Real-time data ingestion using Apache Kafka from sensors and IoT devices in stores capturing sales data.

Apache Spark Streaming is used for real-time data processing and transformations on the streaming data.

Transformations include enrichment, aggregation, fraud detection, and inventory management as described earlier.

Enriched and transformed data is loaded into a data warehouse or data lake in near real-time for immediate analysis and reporting.

Batch ETL (Near Real-Time):

In addition to real-time processing, the retail company also performs batch ETL processes at regular intervals (e.g., daily, hourly) to handle historical data and larger volumes.

Batch ETL jobs are scheduled using Apache Airflow or similar workflow management tools to process data in batches.

The batch ETL pipeline includes similar transformations as the real-time pipeline but operates on larger datasets and historical data.

Processed batch data is also loaded into the data warehouse or data lake, merging seamlessly with the real-time data for comprehensive analytics.

Real-Time Streaming ETL

Real-time customer segmentation based on current purchases and behaviours.

Instantaneous fraud detection and prevention for immediate action.

Dynamic pricing adjustments based on real-time market demand signals.

Live inventory management and restocking notifications for store managers.

Batch ETL:

Historical trend analysis to identify seasonal patterns and long-term customer trends.

Large-scale data aggregation for quarterly and annual business reviews.

Regulatory compliance reporting requiring historical data analysis.

Data archiving and backup for long-term data retention and disaster recovery.

Streaming ETL involves processing data in near real-time as it is generated, enabling organizations to derive insights and respond to events promptly. On the other hand, batch ETL processes data in larger volumes and at scheduled intervals, allowing for deeper analysis of historical data and trend identification. Both methodologies play complementary roles in modern data architectures, addressing different use cases and requirements.

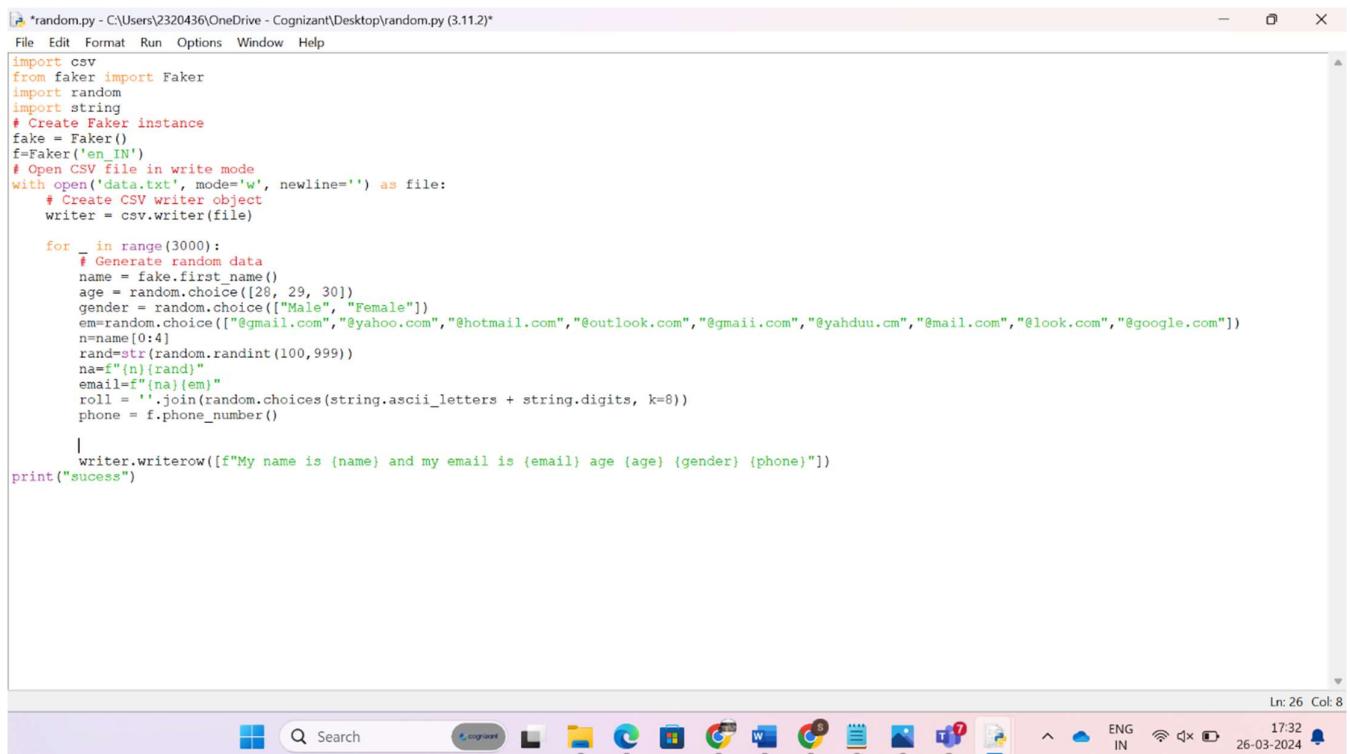
Key Components:

1. Data Ingestion: Both streaming and batch ETL begin with data ingestion, where raw data is collected from various sources such as sensors, applications, databases, and external APIs. Streaming ETL uses technologies like Apache Kafka, Amazon Kinesis, or Azure Event Hubs to ingest and buffer real-time data streams. Batch ETL typically involves bulk data extraction from databases, files, or data lakes at predefined intervals.
2. Transformation: After data ingestion, the raw data undergoes transformations to clean, enrich, aggregate, and structure it for analysis. Streaming ETL employs stream processing frameworks like Apache Flink, Apache Spark Streaming, or Apache Storm to perform transformations on-the-fly as data arrives. Batch ETL uses batch processing frameworks like Apache Spark, Apache Hadoop, or Apache Beam to process data in larger chunks and perform complex transformations.
3. Loading: The transformed data is then loaded into a destination system such as a relational database, data warehouse, or data lake for storage and analysis. Both streaming and batch ETL pipelines integrate seamlessly with various storage and processing systems, ensuring that the transformed data is available for downstream analytics, reporting, and application integration.

CHAPTER 1- Data Ingestion

Data ingestion is the process of importing data from various sources into the system for analysis. In this project, the Unstructured dataset is ingested into the system using Apache Spark, specifically PySpark, which is a powerful data processing framework. PySpark provides functionality to read data from various sources, including CSV files, databases, and distributed storage systems.

For this project, using Python, I generated the student dataset is generated and stored in a CSV file, and PySpark's DataFrame API is used to read the data from this file into a DataFrame for further analysis.

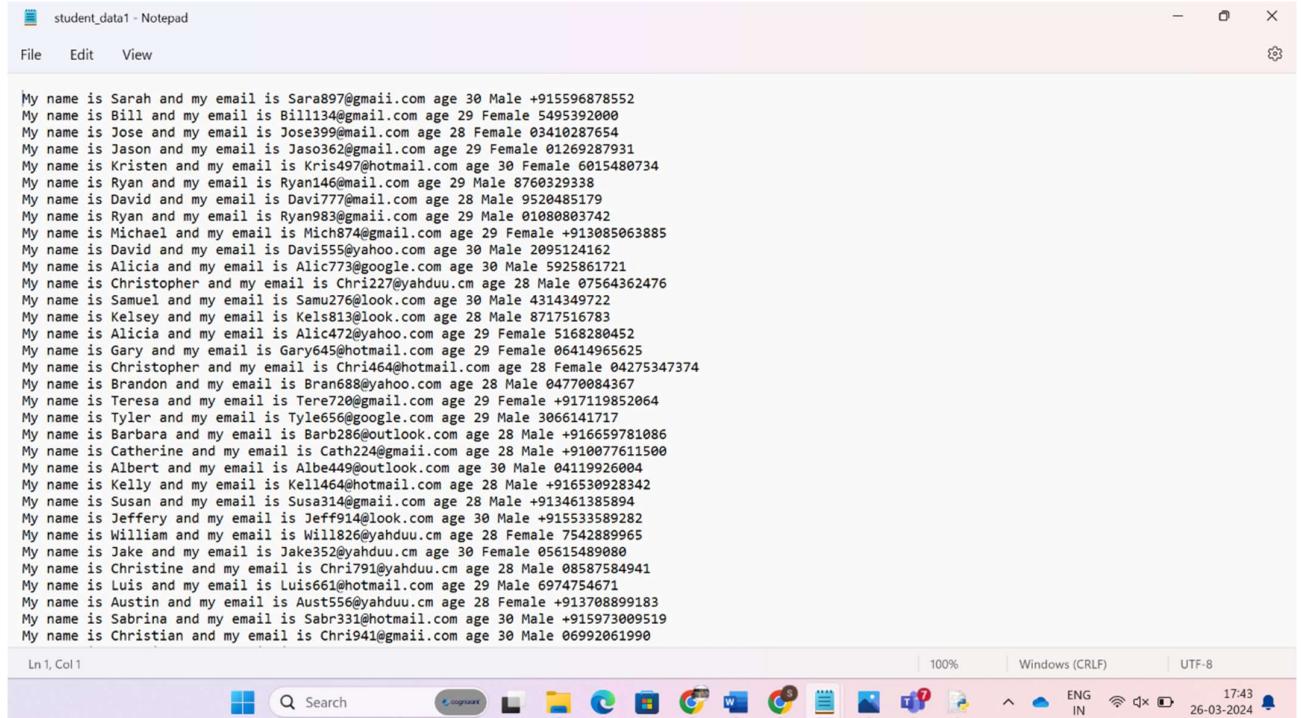


The screenshot shows a Windows desktop environment. A code editor window titled "random.py - C:\Users\2320436\OneDrive - Cognizant\Desktop\random.py (3.11.2)*" is open. The code in the editor is as follows:

```
random.py - C:\Users\2320436\OneDrive - Cognizant\Desktop\random.py (3.11.2)*
File Edit Format Run Options Window Help
import csv
from faker import Faker
import random
import string
# Create Faker instance
fake = Faker()
f=Faker('en_IN')
# Open CSV file in write mode
with open('data.txt', mode='w', newline='') as file:
    # Create CSV writer object
    writer = csv.writer(file)
    for i in range(3000):
        # Generate random data
        name = fake.first_name()
        age = random.choice([20, 29, 30])
        gender = random.choice(["Male", "Female"])
        em=random.choice(["@gmail.com", "@yahoo.com", "@hotmail.com", "@outlook.com", "@gmai.com", "@yahduu.cm", "@mail.com", "@look.com", "@google.com"])
        n=name[0:4]
        rand=str(random.randint(100,999))
        na=f'{n}{rand}'
        email=f'{na}{em}'
        roll = ''.join(random.choices(string.ascii_letters + string.digits, k=8))
        phone = f.phone_number()
        writer.writerow([f"My name is {name} and my email is {email} age {age} {gender} {phone}"])
    print("success")
```

The status bar at the bottom of the window displays "Ln: 26 Col: 8". The taskbar at the bottom of the screen shows various application icons, including Microsoft Edge, Google Chrome, and Microsoft Word, along with the date and time "26-03-2024" and "17:32".

Here input file is in unstructured format. Our goal is unstructured to structured format using spark. In both batch processing and stream processing we are going to do.



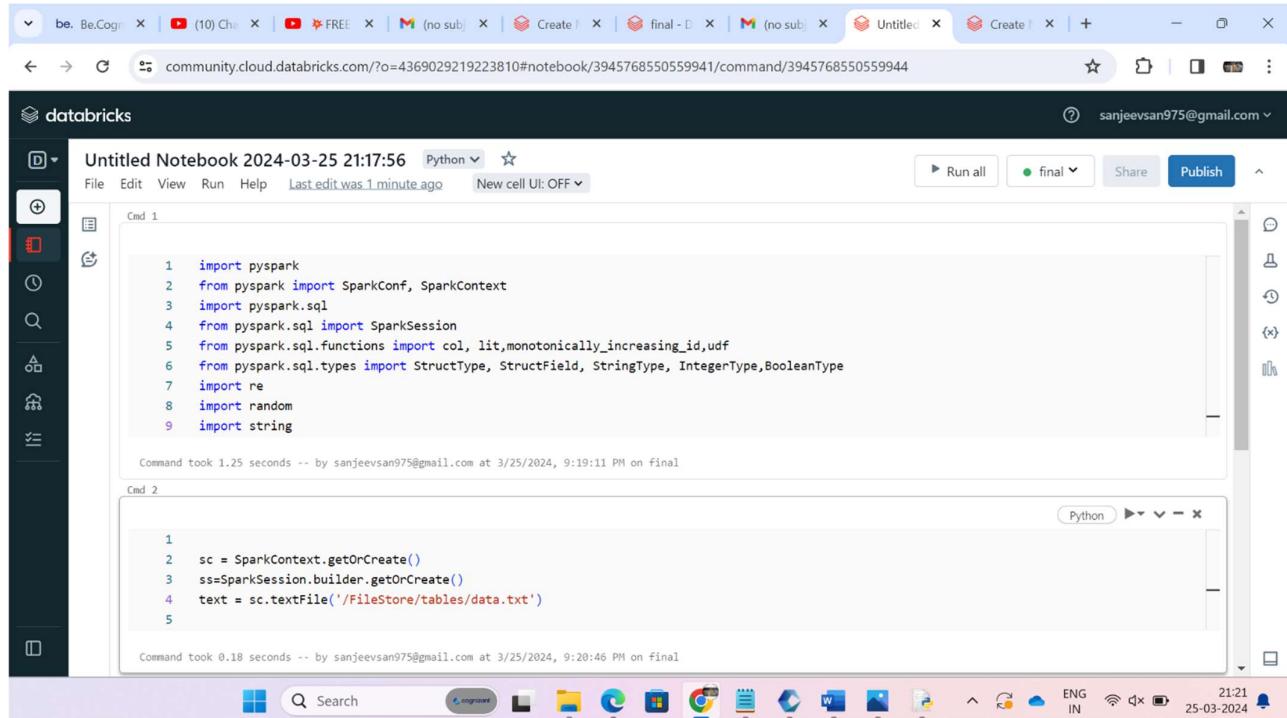
```
student_data1 - Notepad
File Edit View
My name is Sarah and my email is Sara897@gmail.com age 30 Male +915596878552
My name is Bill and my email is Bill134@gmail.com age 29 Female 5495392000
My name is Jose and my email is Jose399@mail.com age 28 Female 03410287654
My name is Jason and my email is Jaso362@gmail.com age 29 Female 01269287931
My name is Kristen and my email is Kris497@hotmail.com age 30 Female 6015480734
My name is Ryan and my email is Ryan146@mail.com age 29 Male 8760329338
My name is David and my email is Davi777@gmail.com age 28 Male 9520485179
My name is Ryan and my email is Ryan983@gmail.com age 29 Male 01080803742
My name is Michael and my email is Mich874@gmail.com age 29 Female +913085063885
My name is David and my email is Davi555@yahoo.com age 30 Male 2095124162
My name is Alicia and my email is Alic773@google.com age 30 Male 5925861721
My name is Christopher and my email is Chri227@yahduu.cm age 28 Male 07564362476
My name is Samuel and my email is Samu276@look.com age 30 Male 4314349722
My name is Kelsey and my email is Kels813@look.com age 28 Male 8717516783
My name is Alicia and my email is Alic472@yahoo.com age 29 Female 5168280452
My name is Gary and my email is Gary645@hotmail.com age 29 Female 06414965625
My name is Christopher and my email is Chri464@hotmail.com age 28 Female 04275347374
My name is Brandon and my email is Bran688@yahoo.com age 28 Male 04770084367
My name is Teresa and my email is Tere720@gmail.com age 29 Female +917119852064
My name is Tyler and my email is Tyle656@google.com age 29 Male 3066141717
My name is Barbara and my email is Barb286@outlook.com age 28 Male +916659781086
My name is Catherine and my email is Cath224@gmail.com age 28 Male +910077611500
My name is Albert and my email is Albe449@outlook.com age 30 Male 04119926004
My name is Kelly and my email is Kell1464@hotmail.com age 28 Male +916530928342
My name is Susan and my email is Susa314@gmail.com age 28 Male +913461385894
My name is Jeffery and my email is Jeff914@look.com age 30 Male +915533589282
My name is William and my email is Will1826@yahduu.cm age 28 Female 7542889965
My name is Jake and my email is Jake352@yahduu.cm age 30 Female 05615489088
My name is Christine and my email is Chri791@yahduu.cm age 28 Male 08587584941
My name is Luis and my email is Luis661@hotmail.com age 29 Male 6974754671
My name is Austin and my email is Aust556@yahduu.cm age 28 Female +913708899183
My name is Sabrina and my email is Sabr331@hotmail.com age 30 Male +915973009519
My name is Christian and my email is Chri941@gmail.com age 30 Male 06992061990
```

Chapter 2

BATCH STREAMING USING ETL:

Importing Libraries:

The code imports necessary libraries for working with Spark, including `pyspark`, `SparkConf`, `SparkContext`, `SparkSession`, and various functions and types from `pyspark.sql`.



The screenshot shows a Databricks notebook interface. The top navigation bar includes tabs for 'Untitled' and 'final'. The main workspace contains two command cells labeled 'Cmd 1' and 'Cmd 2'. Cell 'Cmd 1' contains the following Python code:

```
1 import pyspark
2 from pyspark import SparkConf, SparkContext
3 import pyspark.sql
4 from pyspark.sql import SparkSession
5 from pyspark.sql.functions import col, lit, monotonically_increasing_id, udf
6 from pyspark.sql.types import StructType, StructField, StringType, IntegerType, BooleanType
7 import re
8 import random
9 import string
```

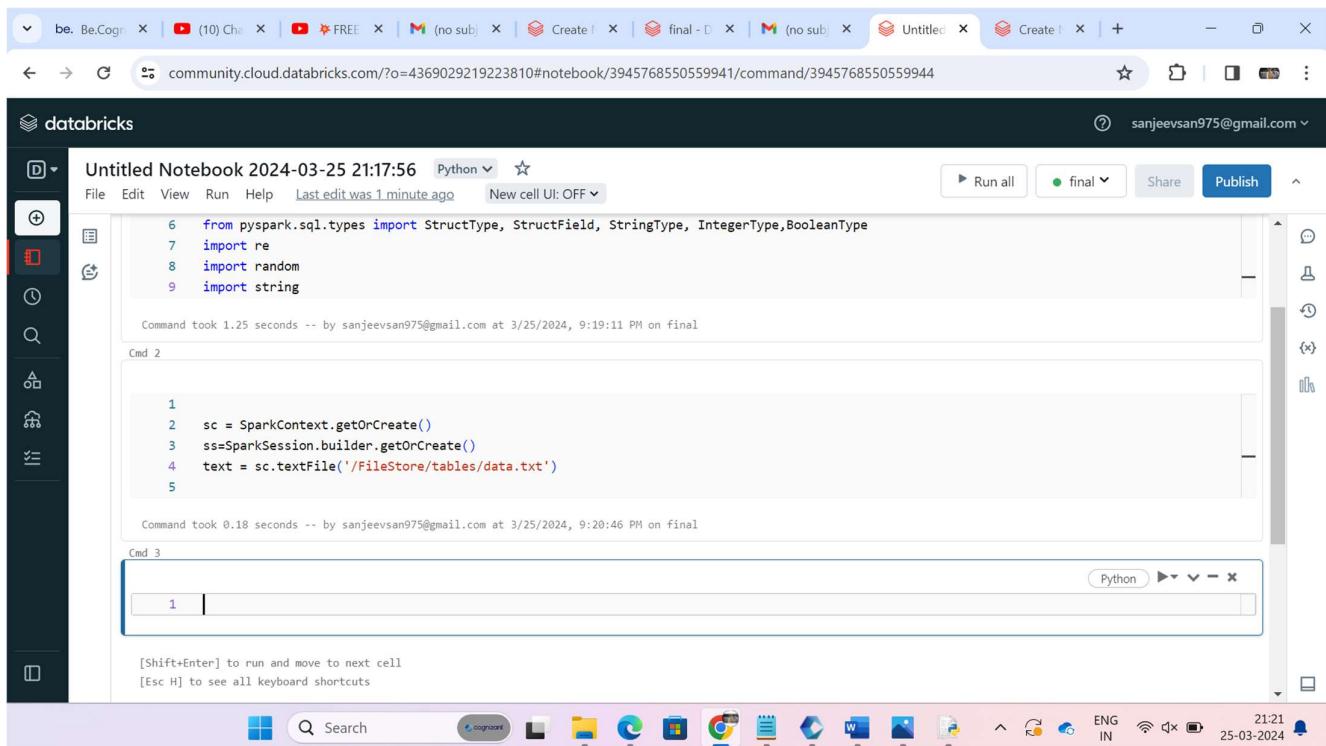
Cell 'Cmd 1' was run successfully, taking 1.25 seconds. Cell 'Cmd 2' contains the following Python code:

```
1 sc = SparkContext.getOrCreate()
2 ss=SparkSession.builder.getOrCreate()
3 text = sc.textFile('/FileStore/tables/data.txt')
```

Cell 'Cmd 2' was run successfully, taking 0.18 seconds. The bottom status bar shows the date and time as 25-03-2024 21:21.

Spark Context and Session Creation:

It creates a Spark Context (`sc`) and Spark Session (`ss`) to interact with Spark functionalities.

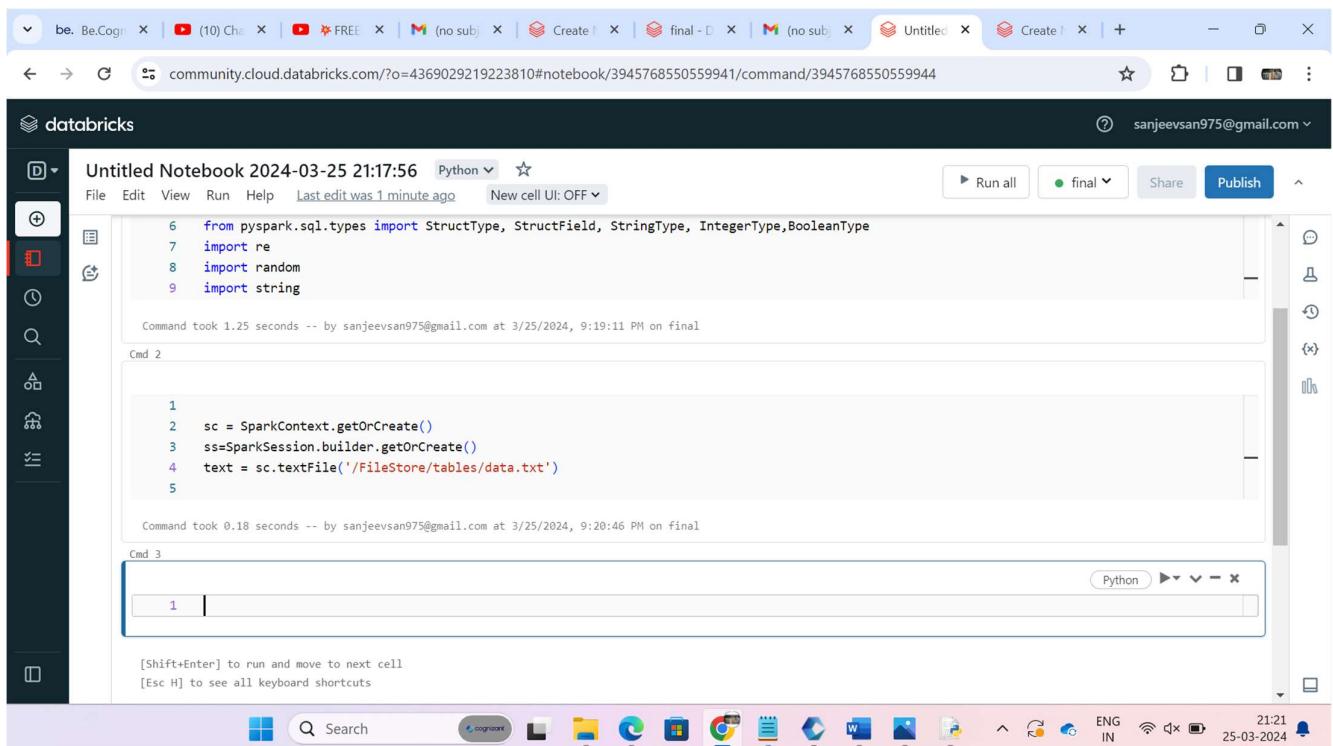


The screenshot shows a Databricks notebook interface with the following details:

- Title:** Untitled Notebook 2024-03-25 21:17:56 Python
- User:** sanjeevsan975@gmail.com
- Code Cells:**
 - Cmd 1:** Imports required modules: `from pyspark.sql.types import StructType, StructField, StringType, IntegerType, BooleanType`, `import re`, `import random`, and `import string`. It also includes a command to run all cells: `Command took 1.25 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:19:11 PM on final`.
 - Cmd 2:** Creates Spark Context and Session, reads a file, and prints the first line. It includes a command to run all cells: `Command took 0.18 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:20:46 PM on final`.
 - Cmd 3:** An empty cell with the Python icon, ready for input.
- Bottom Bar:** Includes icons for various applications like Cognite, Google Sheets, Microsoft Word, and Python, along with system status indicators for battery, signal, and date/time (21:21, 25-03-2024).

Reading Text File:

The code reads the content of a text file named `dupli.txt` located in the `/FileStore/tables/` directory using Spark's `textFile` function. This file likely contains comma-separated values (CSV).



The screenshot shows a Databricks notebook titled "Untitled Notebook 2024-03-25 21:17:56" in Python. The notebook interface includes a sidebar with various icons, a top navigation bar with tabs like "File", "Edit", "View", "Run", "Help", and "New cell UI: OFF", and a right sidebar with sharing and publishing options.

The code in the notebook is as follows:

```
6 from pyspark.sql.types import StructType, StructField, StringType, IntegerType, BooleanType
7 import re
8 import random
9 import string
```

Execution results for Command 2:

```
1
2 sc = SparkContext.getOrCreate()
3 ss=SparkSession.builder.getOrCreate()
4 text = sc.textFile('/FileStore/tables/data.txt')
5
```

Command took 0.18 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:20:46 PM on final

Execution results for Command 3:

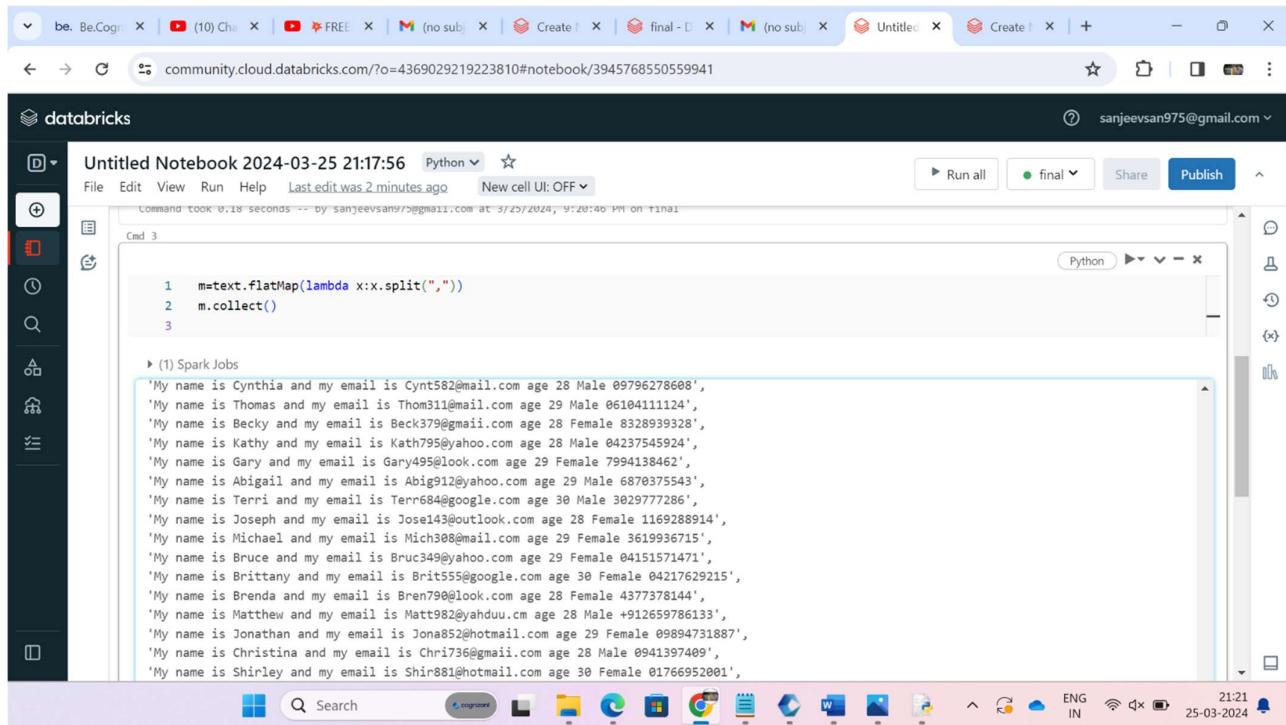
```
1
```

[Shift+Enter] to run and move to next cell
[Esc H] to see all keyboard shortcuts

The bottom status bar shows system icons, language settings (ENG IN), and the date/time (25-03-2024 21:21).

Flat Mapping and Splitting:

The `flatMap` operation is applied to split each line of the text file by comma delimiter, resulting in a flattened RDD of individual data elements.



The screenshot shows a Databricks notebook interface. The notebook title is "Untitled Notebook 2024-03-25 21:17:56". The code cell contains the following Python code:

```
1 m=text.flatMap(lambda x:x.split(","))
2 m.collect()
3
```

The output section displays the results of the `collect` operation, showing a list of names and emails:

```
'My name is Cynthia and my email is Cynt582@mail.com age 28 Male 09796278608',
'My name is Thomas and my email is Thom311@mail.com age 29 Male 06104111124',
'My name is Becky and my email is Beck379@gmail.com age 28 Female 8328939328',
'My name is Kathy and my email is Kath795@yahoo.com age 28 Male 04237545924',
'My name is Gary and my email is Gary495@look.com age 29 Female 7994138462',
'My name is Abigail and my email is Abig912@yahoo.com age 29 Male 6870375543',
'My name is Terri and my email is Terr684@google.com age 30 Male 3029777286',
'My name is Joseph and my email is Jose143@outlook.com age 28 Female 1169288914',
'My name is Michael and my email is Mich308@mail.com age 29 Female 3619936715',
'My name is Bruce and my email is Bruc349@yahoo.com age 29 Female 04151571471',
'My name is Brittany and my email is Brit555@google.com age 30 Female 04217629215',
'My name is Brenda and my email is Bren790@look.com age 28 Female 4377378144',
'My name is Matthew and my email is Matt982@ahduu.cm age 28 Male +912659786133',
'My name is Jonathan and my email is Jona852@hotmail.com age 29 Female 09894731887',
'My name is Christina and my email is Chri736@gmail.com age 28 Male 0941397409',
'My name is Shirley and my email is Shir881@hotmail.com age 30 Female 01766952001'
```

The notebook interface includes a sidebar with various icons for navigation and management. The status bar at the bottom right shows the date (25-03-2024), time (21:21), and language (ENG IN).

Data Extraction:

Lambda functions are used to extract specific fields like name, email, age, gender, and phone number from the flattened RDD.

The screenshot shows a Databricks notebook interface with the following details:

- Header:** Untitled Notebook 2024-03-25 21:17:56 Python
- Actions:** Interrupt, final, Share, Publish
- Output:** A list of names, emails, ages, genders, and phone numbers extracted from a flattened RDD. The output starts with:

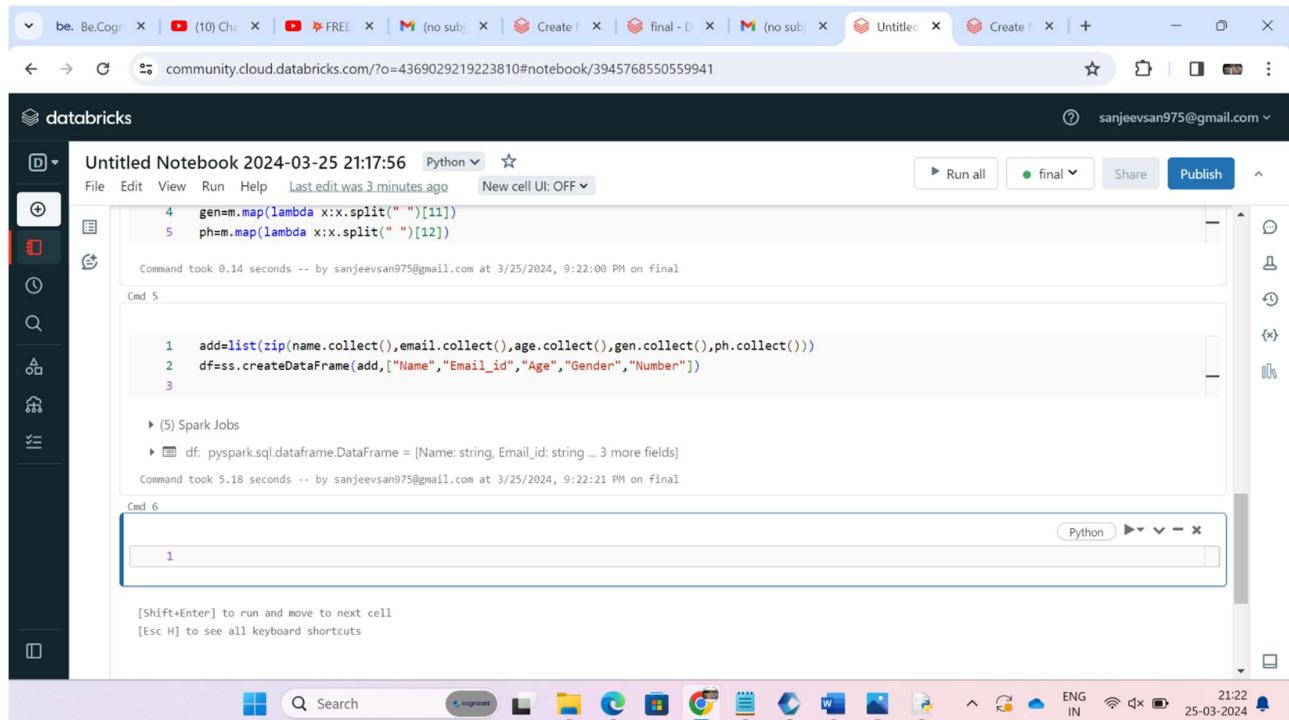
```
'My name is Matthew and my email is Matt982@yahoo.cm age 28 Male +912659786133',
'My name is Jonathan and my email is Jona852@hotmail.com age 29 Female 09894731887',
'My name is Christina and my email is Chri736@gmail.com age 28 Male 0941397409',
'My name is Shirley and my email is Shir881@hotmail.com age 30 Female 01766952001',
'My name is Brittany and my email is Brit307@gmail.com age 30 Male 04591332394',
'My name is Stephanie and my email is Step916@hotmail.com age 28 Female 6901640117',
'My name is Kathryn and my email is Kath122@hotmail.com age 29 Female +918877141289',
'My name is Michael and my email is Mich160@look.com age 29 Male 1020532443',
...]
```
- Command 4:** Shows the Scala code used to map each row into individual fields:

```
1   name=m.map(lambda x: x.split(" ")[3])
2   email=m.map(lambda x: x.split(" ")[8])
3   age=m.map(lambda x: x.split(" ")[10])
4   gen=m.map(lambda x:x.split(" ")[11])
5   ph=m.map(lambda x:x.split(" ")[12])
```
- Command 5:** Shows the Scala code used to collect all fields into a single list:

```
1   add=list(zip(name.collect(),email.collect(),age.collect(),gen.collect(),ph.collect()))
```
- System Bar:** Includes icons for search, notifications, and system status (ENG IN, 25-03-2024, 21:22).

Creating DataFrame:

The extracted data is used to create a DataFrame named `df` with appropriate column names.



The screenshot shows a Databricks notebook interface with the following details:

- Title:** Untitled Notebook 2024-03-25 21:17:56
- Languages:** Python
- Cells:** 6 cells, with Cmd 5 and Cmd 6 visible.
- Code in Cmd 5:**

```
4 gen=m.map(lambda x:x.split(" ")[11])
5 ph=m.map(lambda x:x.split(" ")[12])
```

Command took 0.14 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:22:00 PM on final

```
1 add=list(zip(name.collect(),email.collect(),age.collect(),gen.collect(),ph.collect()))
2 df=ss.createDataFrame(add,['Name','Email_id','Age','Gender','Number'])
3
```

▶ (5) Spark Jobs

▶ df: pyspark.sql.dataframe.DataFrame = [Name: string, Email_id: string ... 3 more fields]

Command took 5.18 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:22:21 PM on final
- Code in Cmd 6:**

```
1
```
- Bottom Status Bar:** Shows the date (25-03-2024), time (21:22), and network status (ENG IN).

Untitled Notebook 2024-03-25 21:17:56 Python

```
1 df.show()
```

(1) Spark Jobs

Alex	Alex729@yahoo.com	29	Female	0396219178
Maria	Mari485@look.com	30	Male	5712320616
Amanda	Aman227@look.com	29	Male	06183463156
Cody	Cody86@mail.com	30	Male	7035067437
Savannah	Sava708@ahduu.cm	29	Male	02378531294
Catherine	Cath658@outlook.com	29	Male	8932083922
Robert	Robe957@look.com	30	Female	0696925565
James	Jame140@ahduu.cm	29	Male	+914720157449
Patricia	Patr582@ahduu.cm	29	Female	+916413416861
Chad	Chad343@yahoo.com	29	Female	+918141417306
Zachary	Zach738@google.com	30	Female	01666478551
John	John518@look.com	28	Female	+912921407081
Brian	Bria865@outlook.com	30	Female	01952326389
Vincent	Vinc997@outlook.com	30	Male	06442086564
Thomas	Thom987@outlook.com	29	Male	06896219146
Cameron	Came859@outlook.com	30	Male	0818649286
Christina	Chri534@gmail.com	29	Male	07285430311
Connie	Conn843@outlook.com	28	Female	05165399019

21:24 25-03-2024

Untitled Notebook 2024-03-25 21:17:56 Python

```
1 df.show()
```

(1) Spark Jobs

Christina	Chri534@gmail.com	29	Male	07205430311
Connie	Conn843@outlook.com	28	Female	05165399019

only showing top 20 rows

Command took 1.48 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:24:10 PM on final

```
1 df.count()
```

(2) Spark Jobs

Out[8]: 3450

Command took 2.65 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:24:34 PM on final

```
1 dff=df.distinct()
```

```
2 dff.show()
```

(2) Spark Jobs

```
1 dff: pyspark.sql.dataframe.DataFrame = [Name: string, Email_id: string ... 3 more fields]
```

Thomas	Thom987@outlook.com	29	Male	06896219146
--------	---------------------	----	------	-------------

21:25 25-03-2024

Distinct Records:

Duplicate records are removed from the DataFrame using the `distinct()` function to ensure each record is unique.

The screenshot shows a Databricks notebook interface. The code cell contains the following Python code:

```
1 dff=df.distinct()
2 dff.show()
```

The output pane displays a list of 30 rows of data, each containing a name, email address, gender, and ID. The data is as follows:

Name	Email Address	Gender	ID
Thomas	Thom087@outlook.com	Male	06896219146
Connie	Conn843@outlook.com	Female	05165390919
Kelsey	Kels68@mail.com	Male	352222174
Cameron	Came859@outlook.com	Male	0818649286
Vincent	Vinc997@outlook.com	Male	06442086564
Savannah	Sava708@yahduu.cm	Male	02378531294
Patricia	Patr582@yahduu.cm	Female	+916413416861
Brian	Bria865@outlook.com	Female	01952326389
Alex	Alex729@yahoo.com	Female	0396219178
Catherine	Cathh58@outlook.com	Male	8932083922
Maria	Mari485@look.com	Male	5712320616
Christina	Chris34@mail.com	Male	07205430311
Amanda	Aman227@look.com	Male	06183463156
Robert	Robe957@look.com	Female	0696925565
Zachary	Zach738@google.com	Female	01666478551
Jesse	Jess654@yahduu.cm	Female	+918130392909
John	John518@look.com	Female	+912921407081
James	Jame140@yahduu.cm	Male	+914720157449

The screenshot shows a Databricks notebook interface. The code cell contains the following Python code:

```
1 dff.count()
```

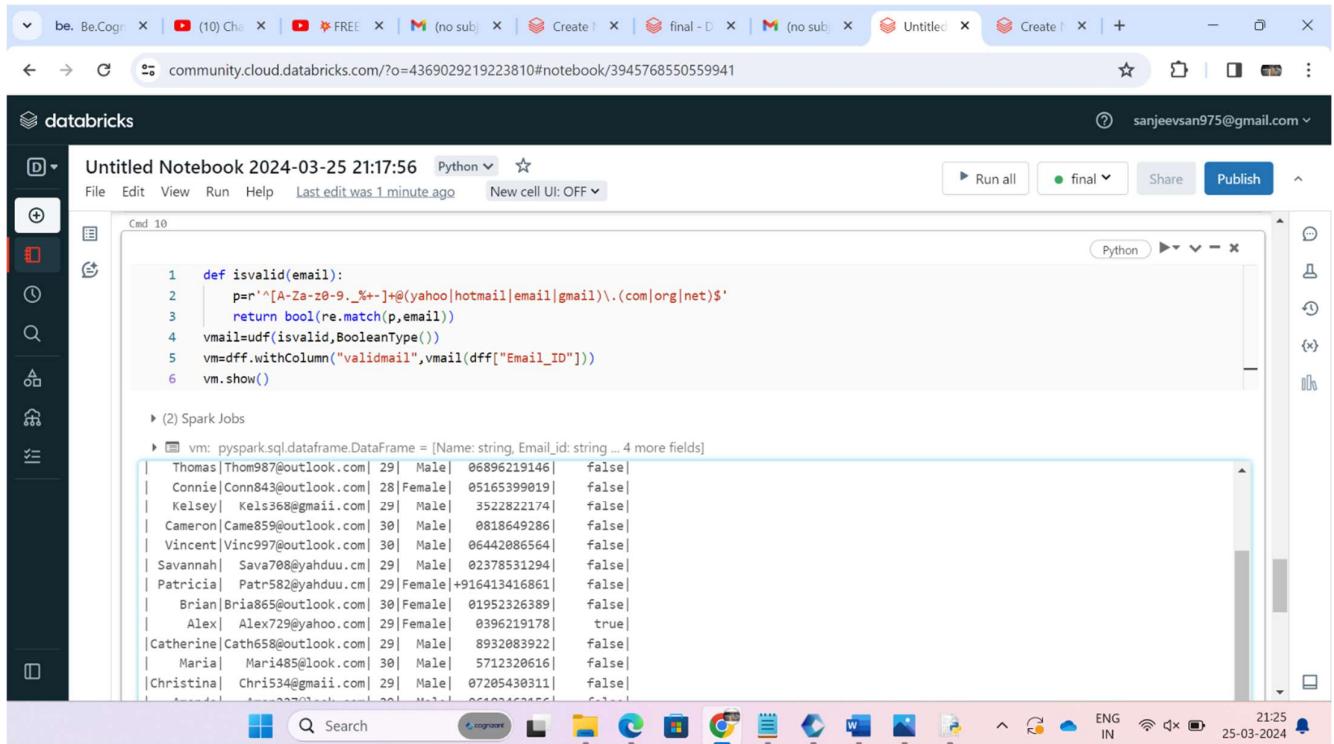
The output pane displays the result of the count operation:

```
Out[10]: 3000
```

Below the code cell, there is a command cell labeled "Cmd 9" which contains the command "Python".

Email Validation:

User-defined function (UDF) named 'isValid' is defined to check the validity of email addresses based on a regular expression pattern. This UDF is applied to the DataFrame to create a new column 'validmail' indicating whether each email is valid or not.



The screenshot shows a Databricks notebook interface. The notebook title is "Untitled Notebook 2024-03-25 21:17:56" and it is set to Python. The code cell contains the following Python code:

```
1 def isValid(email):
2     p=r'^[A-Za-z0-9._%+-]+@(yahoo|hotmail|email|gmail)\.(com|org|net)$'
3     return bool(re.match(p,email))
4 vmail=udf(isValid,BooleanType())
5 vm=dff.withColumn("validmail",vmail(dff["Email_ID"]))
6 vm.show()
```

The output section shows the results of the `show` command:

```
(2) Spark Jobs
▶ (2) Spark Jobs
vm: pyspark.sql.dataframe.DataFrame = [Name: string, Email_id: string ... 4 more fields]
| Thomas|Thom987@outlook.com| 29| Male| 06896219146| false|
| Connie|Conn843@outlook.com| 28|Female| 05165399019| false|
| Keisey| Kels368@gmail.com| 29| Male| 3522822174| false|
| Cameron|Came859@outlook.com| 30| Male| 0818649286| false|
| Vincent|Vinc97@outlook.com| 30| Male| 06442086564| false|
| Savannah| Sava708@yahoo.cm| 29| Male| 02378531294| false|
| Patricia| Patr582@yahoo.cm| 29|Female| +916413416861| false|
| Brian|Bria865@outlook.com| 30|Female| 01952326389| false|
| Alex| Alex729@yahoo.com| 29|Female| 0396219178| true|
| Catherine|Cath658@outlook.com| 29| Male| 8932083922| false|
| Maria| Mari485@look.com| 30| Male| 5712320616| false|
| Christina| Chri534@gmail.com| 29| Male| 07205430311| false|
| Amanda| Amad22701@gmail.com| 29| Female| 05620420561| false|
```

Filtering Valid Email Addresses:

Rows with valid email addresses are filtered out from the DataFrame..

The screenshot shows a Databricks notebook interface. The notebook title is "Untitled Notebook 2024-03-25 21:17:56" and the language is Python. The user's email is listed at the top right: "sanjeevsan975@gmail.com".

Cell 11 contains the following code:

```
1 fs=vm.filter(vm["validmail"]== True)
2 res=fs.drop("validmail")
```

The output of this cell shows a table with two rows of data:

Name	Email	Gender	Age	Valid Mail
John	John@LOOK.COM	Male	28	True
James	Jame140@yahoo.cm	Male	29	False

Cell 12 is currently empty, indicated by the number "1" in the input field.

At the bottom, the taskbar shows various application icons and the system clock "21:26 25-03-2024".

Untitled Notebook 2024-03-25 21:17:56 Python

```
1 res.show()
```

(2) Spark Jobs

Name	Email	Age	Gender	Phone
Rachel	Rach352@hotmail.com	29	Male	+913388570460
Carl	Carl1654@hotmail.com	29	Female	5147296812
Nicole	Nico274@gmail.com	29	Male	06802621288
Jose	Jose529@yahoo.com	29	Female	+9158009481341
Ronald	Rona01@yahoo.com	29	Female	2566667833
Alex	Alex729@yahoo.com	29	Female	0396219178
Phyllis	Phyl174@hotmail.com	28	Male	+914964370948
Cynthia	Cynth937@hotmail.com	29	Female	+912254429087
Teresa	Tere724@yahoo.com	28	Female	+917804572063
Xavier	Xav1405@gmail.com	30	Female	05949052321
Steven	Steve655@hotmail.com	29	Male	06158299231
Katherine	Kath696@gmail.com	28	Male	+914534824788
Chad	Chad343@yahoo.com	29	Female	+918141417306
Susan	Susa948@hotmail.com	30	Female	4212300968
Veronica	Ver0684@gmail.com	30	Male	05717150833
Jessica	Jess572@gmail.com	30	Male	+913644939979
Stephen	Step764@hotmail.com	30	Female	+919628815056
Denise	Deni1889@hotmail.com	29	Male	+911486756075

21:26 25-03-2024

Untitled Notebook 2024-03-25 21:17:56 Python

```
only showing top 20 rows
```

Command took 2.38 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:26:24 PM on final

```
1 res.count()
```

(3) Spark Jobs

Out[14]: 1011

Command took 1.94 seconds -- by sanjeevsan975@gmail.com at 3/25/2024, 9:26:41 PM on final

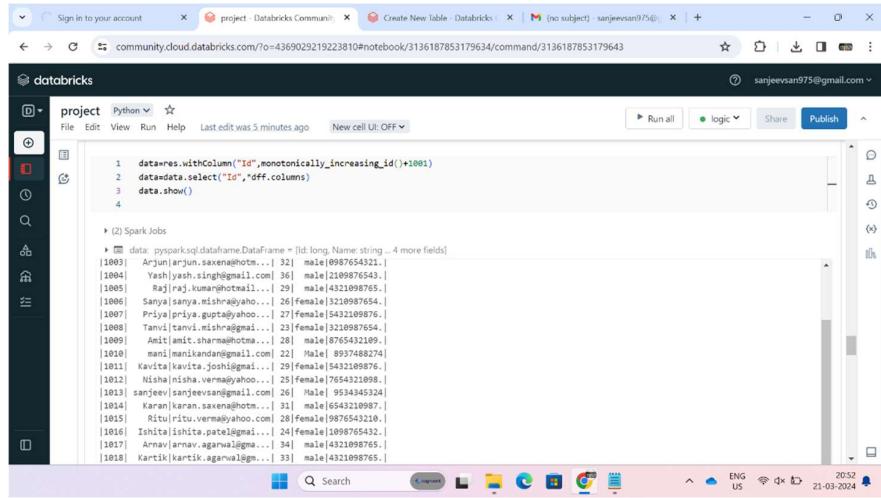
```
1
```

[Shift+Enter] to run and move to next cell
[Esc H] to see all keyboard shortcuts

21:33 25-03-2024

Adding Unique IDs:

Unique IDs are added to each row using the `monotonically_increasing_id()` function, ensuring each row has a distinct identifier.



A screenshot of a Databricks notebook interface. The code cell contains the following Python code:

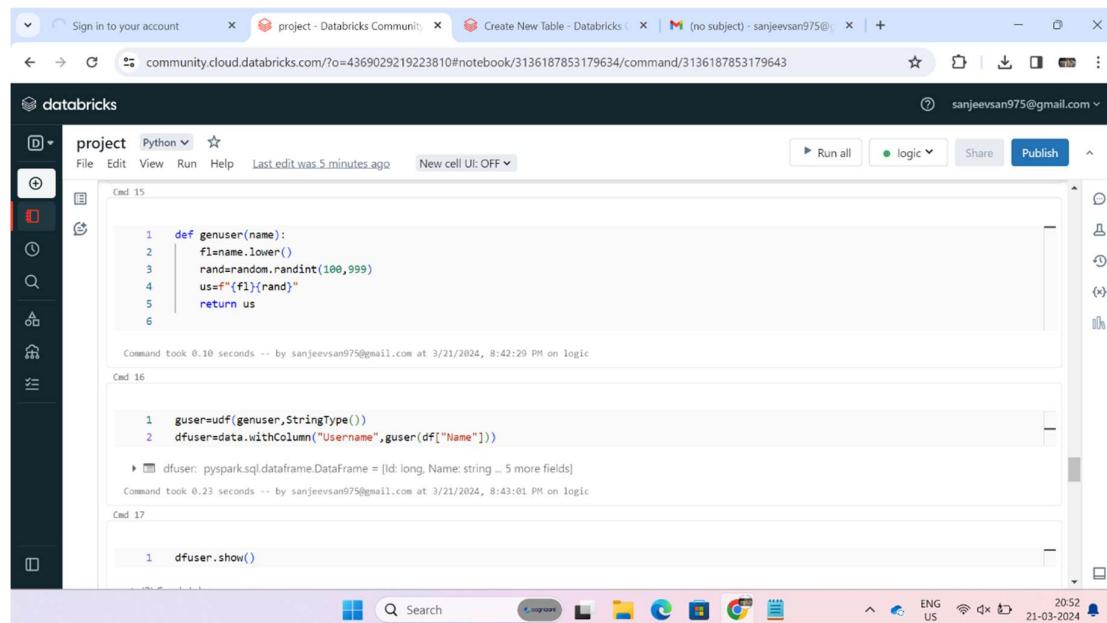
```
1 data=res.withColumn("Id",monotonically_increasing_id()+1001)
2 data=data.select("Id","dfc.columns")
3 data.show()
4
```

The output shows a DataFrame with 10 rows, each containing an 'Id' column with values ranging from 1001 to 1010, and the original columns from the input DataFrame.

Id	Name	Gender	Email
1001	Arjun arjun.saxena@hotmail.com	Male	8997654321...
1002	Yash yash.singh@gmail.com	Male	2109876543...
1003	Raj raj.kumar@hotmail.com	Male	4321098765...
1004	Sanya sanya.dishra@yahoo...	Female	3210987654...
1005	Priya priya.gupt@yahoo...	Female	5432109876...
1006	Tanvi tanvi.nishra@gmail...	Female	3210987654...
1007	Ami ami.sharma@hotmail...	Male	8765432109...
1008	Mansi mankandare@gmail.com	Male	89374688274...
1009	Kavita kavita.joshi@gmail...	Female	5432109876...
1010	Nisha nisha.verma@yahoo...	Female	7654321098...
1011	Sanjeev sanjeev.san@gmail.com	Male	9534345924...
1012	Karan karan.saxena@hotmail...	Male	6543210989...
1013	Ritika ritika.singh@gmail...	Female	4321098765...
1014	Ishita ishita.patel@gmail...	Female	1098765432...
1015	Anupama anupama.agrawal...	Male	4321098765...
1016	Kartik kartik.agrawal@gmail...	Male	4321098765...

Generating Usernames:

Another UDF named 'genuser' is defined to generate usernames based on the lowercased first name and a random integer. This UDF is applied to create a new column 'Username'.



A screenshot of a Databricks notebook interface. The code cell contains the following Python code for defining a UDF:

```
1 def genuser(name):
2     f1=name.lower()
3     rand=random.randint(100,999)
4     us=f1{rand}
5     return us
6
```

The command took 0.10 seconds.

The next cell shows the application of the UDF:

```
1 guser=udf(genuser,StringType())
2 dfuser=data.withColumn("Username",guser(df["Name"]))
```

The command took 0.23 seconds.

The final cell shows the resulting DataFrame:

```
1 dfuser.show()
```

```

1 dfuser.show()

> (2) Spark Jobs
+---+---+---+---+---+
| Id | Name | Email_id|Age|Gender| Number| Username|
+---+---+---+---+---+
|1001| Sonam|sonam.sharma@yahoo...| 30|female|1098765432.| sonam871|
|1002| Rohit|rohit.verma@hotmail...| 33| male|2109876543.| rohit162|
|1003| Arjun|arjun.saxena@hotmail...| 32| male|0987654321.| arjun512|
|1004| Yash|yash.singh@gmail.com| 36| male|2109876543.| yash471|
|1005| Raj|raj.kumar@hotmail...| 29| male|4321098765.| raj758|
|1006| Sanya|sanya.mishra@yahoo...| 26|female|3210987654.| sanya663|
|1007| Priya|priya.gupta@yahoo...| 27|female|5432109876.| priya666|
|1008| Tanvi|tanvi.mishra@gmail...| 23|female|3210987654.| tanvi529|
|1009| Amit|amit.sharma@hotmail...| 28| male|8765432109.| amit211|
|1010| mani|mani.kandan@gmail.com| 22| Male|8937488274| mani555|
|1011| Kavita|kavita.joshi@gmail...| 29|female|5432109876.| kavita436|
|1012| Nisha|nisha.verma@yahoo...| 25|female|7654321098.| nisha287|
|1013| sanjeev|sanjeevsan@gmail.com| 26| Male|9534345324| sanjeev423|
|1014| Karan|karan.saxena@hotmail...| 31| male|6543210987.| karan471|
|1015| Ritu|ritu.verma@yahoo.com| 28|female|9876543210.| ritu184|
+---+---+---+---+---+

```

Generating Passwords:

Similarly, a UDF named `genpass` is defined to generate random passwords consisting of letters and digits. This UDF is applied to create a new column `Password`

```

1 def genpass():
2     ln=8
3     passc=string.ascii_letters+string.digits
4     p=''.join(random.choice(passc) for i in range(ln))
5     return p

Command took 0.09 seconds -- by sanjeevsan975@gmail.com at 3/21/2024, 8:47:02 PM on logic

1 genp=udf(genpass,StringType())
2 dfp=dfuser.withColumn("Password",genp())

Command took 0.23 seconds -- by sanjeevsan975@gmail.com at 3/21/2024, 8:47:05 PM on logic

1 dfp.show(54)
2

```

```

1 dfp.show(54)
2

▶ (2) Spark Jobs
|1036| Anjali|anjali.ali@gmail.com| 28|female|7654321098.| anjali603|Fjj7UF4d|
|1037| Vikram|vikram.singh@gmail...| 35| male|2109876543.| vikram525|jtaj8DS|
|1038| Meera|meera.patel@gmail...| 25|female|1098765432.| meera335|NNJ7f1SA|
|1039| Shruti|shruti.sharma@gmail...| 30|female|9876543210.| shruti766|eQ0582W1|
|1040| Aditya|aditya.kumar@gmail...| 31| male|8987654321.| aditya822|Egk69iis|
|1041| Mohit|mohit.verma@gmail...| 33| male|8765432109.| mohit962|Fc80UPes|
|1042| Sneha|sneha.mishra@gmail...| 24|female|3210987654.| sneha463|5FFVcC55|
|1043| Deepak|deepak.gupta@hotmail...| 34| male|8765432109.| deepak272|2HrE6H4W|
|1044| Meera|meera.patel@gmail...| 44|female| 3210987654| meera395|YGMmB7F|
|1045| Riya|riya.joshi@yahoo.com| 27|female|3210987654.| riya139|ZHyg10W|
|1046| Vineet|vineet.agarwal@ho...| 35| male|2109876543.| vineet474|kBWTMMeA|
|1047| Sameer|sameer.verma@hotmail...| 29| male|6543210987.| sameer375|H3loasCS|
|1048| Neha|neha.patel@yahoo.com| 30|female|9876543210.| neha897|QxD2Msj6|
|1049| Pooja|pooja.sharma@yahoo...| 31|female|7654321098.| pooja263|KBd579TR|
|1050| Kritika|kritika.al@yahoo...| 26|female|5432109876.| kritika983|HqyzQt1|
|1051| Anjali|anjali.mishra@yahoo...| 24|female|1098765432.| anjali410|XtpaENn9|
|1052| Mohan|mohan.saxena@hotmail...| 33| male|8987654321.| mohan243|CBUKw3LD|

```

Loading to MySQL Cloud:

Finally, the processed DataFrame (`dfp`) is written to a MySQL database table specified by the `table` parameter using JDBC connection parameters such as driver, URL, username, and password. The data is appended to the table in the database.

For that we are creating a google cloud SQL for loading a data.

MySQL in the cloud offers scalability, managed services, high availability, global accessibility, and cost efficiency. It easily scales to accommodate growing data needs, freeing up time with managed services for backups, maintenance, and updates. High availability features ensure uptime and data durability, while global accessibility fosters collaboration. Cost efficiency stems from a pay-as-you-go model.

Conversely, local MySQL provides performance advantages, data control, predictable costs, customization, and offline access. It offers faster query execution, control over infrastructure, predictable costs, customization options, and ensures uninterrupted access in environments with limited internet connectivity. Organizations often choose based on needs, budget, and regulatory requirements.

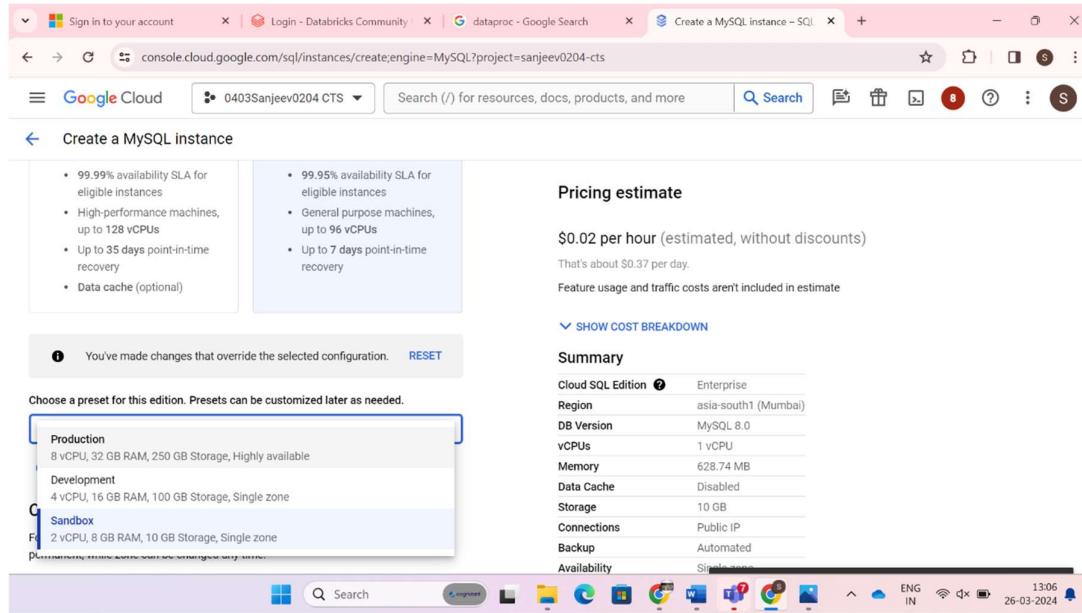
1) Create instance in Mysql cloud:

The screenshot shows the 'Create a MySQL instance' page in the Google Cloud Platform. The URL in the browser is `console.cloud.google.com/sql/instances/create;engine=MySQL?project=sanjeev0204-cts`. The page is divided into several sections:

- Instance info:** Includes fields for Instance ID (set to "sanjeev123"), Password (set to "*****"), and a checkbox for "No password".
- Pricing estimate:** Shows a cost of "\$2.21 per hour (estimated, without discounts)".
- Summary:** Lists instance details:

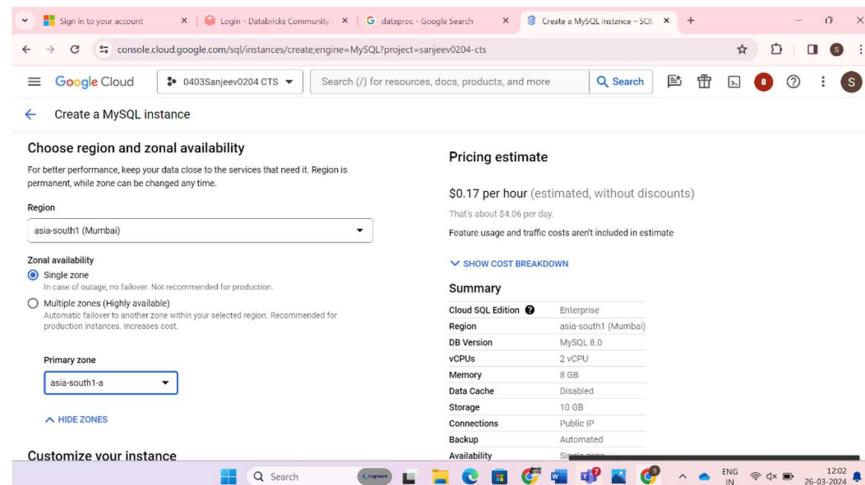
Cloud SQL Edition	Enterprise Plus
Region	us-central1 (Iowa)
DB Version	MySQL 8.0
vCPUs	8 vCPU
Memory	64 GB
Data Cache	Enabled (375 GB)
Storage	250 GB
Connections	Public IP
Backup	Automated
Availability	Multiple regions (highly available)
- Choose a Cloud SQL edition:** A note states: "A Cloud SQL edition determines foundational characteristics of your instance and cannot be changed after creation." Below this, there's a "Search" bar and a toolbar with various icons.

2) Compare to other production sandbox is less in features and prize.



Region and zone:

A region is a specific geographical location where you can run your resources. Each region is subdivided into several zones. For example, the us-central1 region in the central United States has zones us-central1-a , us-central1-b , us-central1-c , and us-central1-f .



3) Shared Core (vCPU) Instances and Dedicated Core (vCPU) Instances:

Shared Core (vCPU) Instances:

- Shared-core machine types provide **one virtual CPU** that runs for a portion of the time on a single hardware hyper-thread of the host CPU.
- These instances are **cost-effective** for running small, non-resource-intensive applications.

Dedicated Core (vCPU) Instances:

- In contrast, dedicated core instances allocate **exclusive CPU resources** to your virtual machine.
- Each core is reserved for your instance, ensuring consistent performance and security.

So, we using shared core for cost-effective

The screenshot shows the Google Cloud Platform interface for creating a MySQL instance. The URL in the address bar is `console.cloud.google.com/sql/instances/create;engine=MySQL?project=sanjeev0204-cts`. The page is titled "Create a MySQL instance".
Machine configuration:
- Machine shapes: Choose from the most commonly used shapes or choose a custom shape. Make sure your selection has enough memory to hold your largest table.
- Shared core:
 - 1 vCPU, 0.614 GB
 - 1 vCPU, 1.7 GB
- Data cache: Exclusive to Enterprise Plus edition. Adds a local SSD for read caching to reduce read latency and improve performance at additional cost. [Learn more](#)
- Storage: Storage type is SSD. Storage size is 10 GB, and will automatically scale as needed. Google-managed key enabled (most common).
Pricing estimate: \$0.02 per hour (estimated, without discounts). That's about \$0.37 per day. Feature usage and traffic costs aren't included in estimate.
Show cost breakdown
Summary:

Cloud SQL Edition	Enterprise
Region	asia-south1 (Mumbai)
DB Version	MySQL 8.0
vCPUs	1 vCPU
Memory	628.74 MB
Data Cache	Disabled
Storage	10 GB
Connections	Public IP
Backup	Automated
Availability	Single zone

4) Storage SSD or HDD:

Solid state drives (SSDs) typically use flash-based memory to store data and thus have no moving parts. They have faster read/write speeds than HDDs, lower access times (less latency), and a higher cost per gigabyte of storage. So, we choose SSD.

Storage

Storage type
Choice is permanent. Storage type affects performance.

SSD (Recommended)
Most popular choice. Lower latency than HDD with higher QPS and data throughput.

HDD
Lower performance than SSD with lower storage rates.

Storage capacity
10 - 65,536 GB. Higher capacity improves performance, up to the limits set by the machine type. Capacity can't be decreased later.

10 GB
 20 GB
 100 GB
 250 GB
 Custom

Enable automatic storage increases
If enabled, whenever you are nearing capacity, storage will be incrementally (and permanently) increased. [Learn more](#)

Pricing estimate
\$0.02 per hour (estimated, without discounts)
That's about \$0.37 per day.
Feature usage and traffic costs aren't included in estimate

Show COST BREAKDOWN

Summary

Cloud SQL Edition	Enterprise
Region	asia-south1 (Mumbai)
DB Version	MySQL 8.0
vCPUs	1 vCPU
Memory	628.74 MB
Data Cache	Disabled
Storage	10 GB
Connections	Public IP
Backup	Automated
Availability	Single zone

5) Network connection :

A public IP address is a unique IP address assigned to your network router by your internet service provider and can be accessed directly over the internet. A private IP address is a unique address that your network router assigns to your device. It is used within a private network to connect securely to other devices.

The screenshot shows the 'Create a MySQL instance' page in the Google Cloud Platform. The top navigation bar includes tabs for 'Sign in to your account', 'Login - Databricks Community', 'dataproc - Google Search', and 'Create a MySQL instance - SQL'. The main content area has a header 'Create a MySQL instance' with a back arrow. On the left, there's a 'Connections' section with 'Instance IP assignment' options: 'Private IP' (unchecked) and 'Public IP' (checked). Below that is an 'Authorized networks' section with a note about adding CIDR ranges. A warning message states: 'You have added 0.0.0.0/0 as an allowed network. This prefix will allow any IPv4 client to pass the network firewall and make login attempts to your instance, including clients you did not intend to allow. Clients still need valid credentials to successfully log in to your instance.' On the right, there's a 'Pricing estimate' section showing '\$0.02 per hour (estimated, without discounts)' and a summary table:

Cloud SQL Edition	Enterprise
Region	asia-south1 (Mumbai)
DB Version	MySQL 8.0
vCPUs	1 vCPU
Memory	628.74 MB
Data Cache	Disabled
Storage	10 GB
Connections	Public IP
Backup	Automated
Availability	Single zone

The status bar at the bottom shows various icons and the date/time: 26-03-2024, 12:03.

This screenshot is similar to the one above, but it shows a 'New network' dialog box open on the left. The dialog has fields for 'Name' (set to 'all') and 'Network' (set to '0.0.0.0/0'). Below these fields is an example: 'Example: 199.27.25.0/24'. At the bottom of the dialog are 'DONE' and 'ADD A NETWORK' buttons. The rest of the page and interface are identical to the first screenshot.

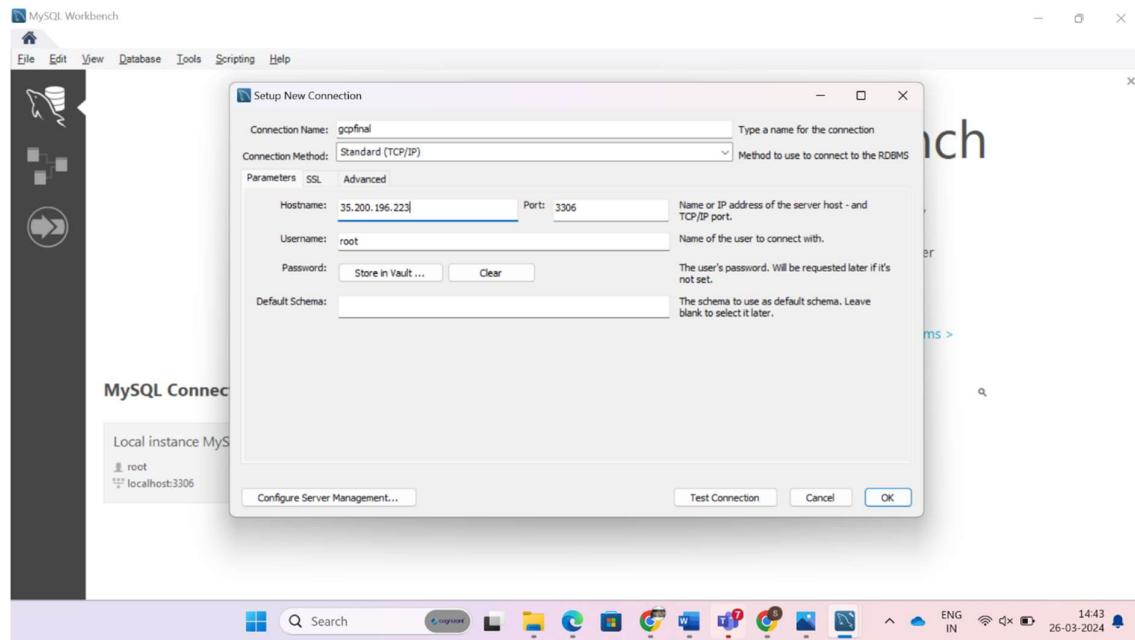
6) Then create an instance:

The screenshot shows the 'Create a MySQL instance' configuration page in the Google Cloud Platform. On the left, there are several sections with dropdown menus: 'Automatic backups enabled. Point-in-time recovery (via binary logs) enabled.', 'Maintenance' (Updates may occur any day of the week. Cloud SQL chooses the maintenance timing.), 'Flags' (No flags set.), 'Query insights' (Query insights disabled), and 'Labels' (No labels set.). Below these is a 'HIDE CONFIGURATION OPTIONS' button. At the bottom are 'CREATE INSTANCE' and 'CANCEL' buttons. On the right, under 'SHOW COST BREAKDOWN', is a 'Summary' table with the following details:

Cloud SQL Edition	Enterprise
Region	asia-south1 (Mumbai)
DB Version	MySQL 8.0
vCPUs	1 vCPU
Memory	628.74 MB
Data Cache	Disabled
Storage	10 GB
Connections	Public IP
Backup	Automated
Availability	Single zone
Point-in-time recovery	Enabled
Network throughput (MB/s)	125 of 125
Disk throughput (MB/s)	Read: 4.8 of 125.0 Write: 4.8 of 107.8
IOPS	Read: 300 of 12,000 Write: 300 of 10,000

Now MySql instance is created,

- 1) Open mysql work bench.
- 2) And open setup new connections and enter a connection name.
And Hostname in that we need give mysql instance public IP address.
- 3) Enter a password what we are given in mysql instance.
- 4) And check for the connection After the successful connection.
- 5) Then open the mysql work bench and create a database as “demo”.



- We have a code to connect Mysql cloud to Databricks.
- Now we need to give IP address in URL.
- Enter database name and table name in table row.
- Then enter the password and run the program.

```

1 driver = "com.mysql.cj.jdbc.Driver"
2 url = "jdbc:mysql://35.200.196.223"
3 table = "demo.demo123"
4 user = "root"
5 password = "sanjeev@123"
6
7 dfp.write.format("jdbc").option("driver", driver).option("url", url).option("dbtable", table).option("mode", "append").option("user", user).option("password", password).save()
  
```

(2) Spark Jobs

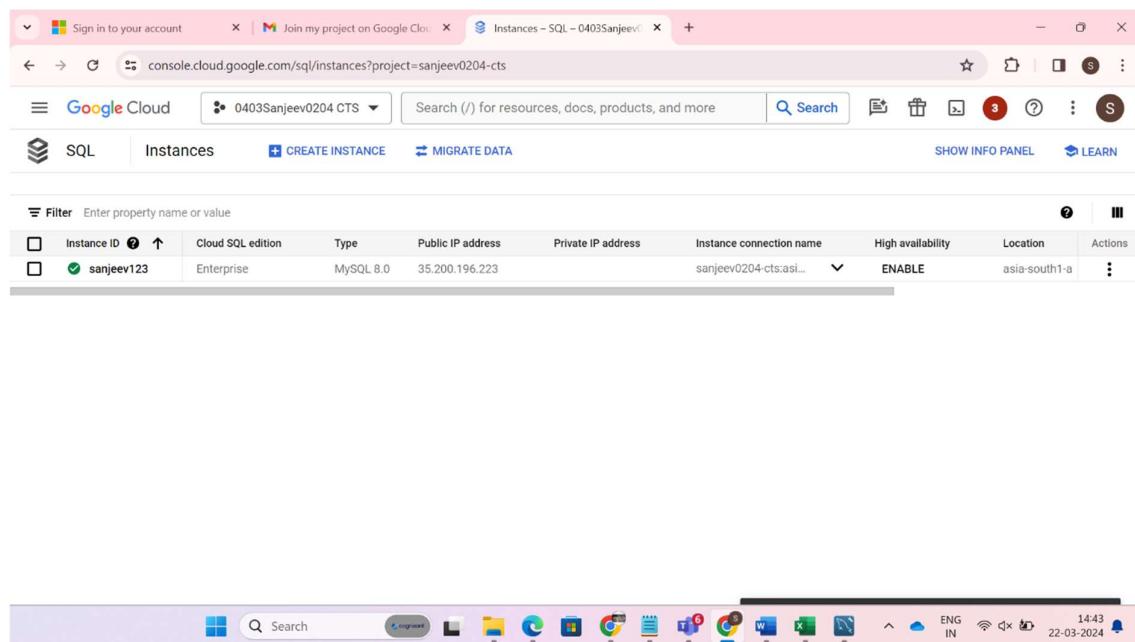
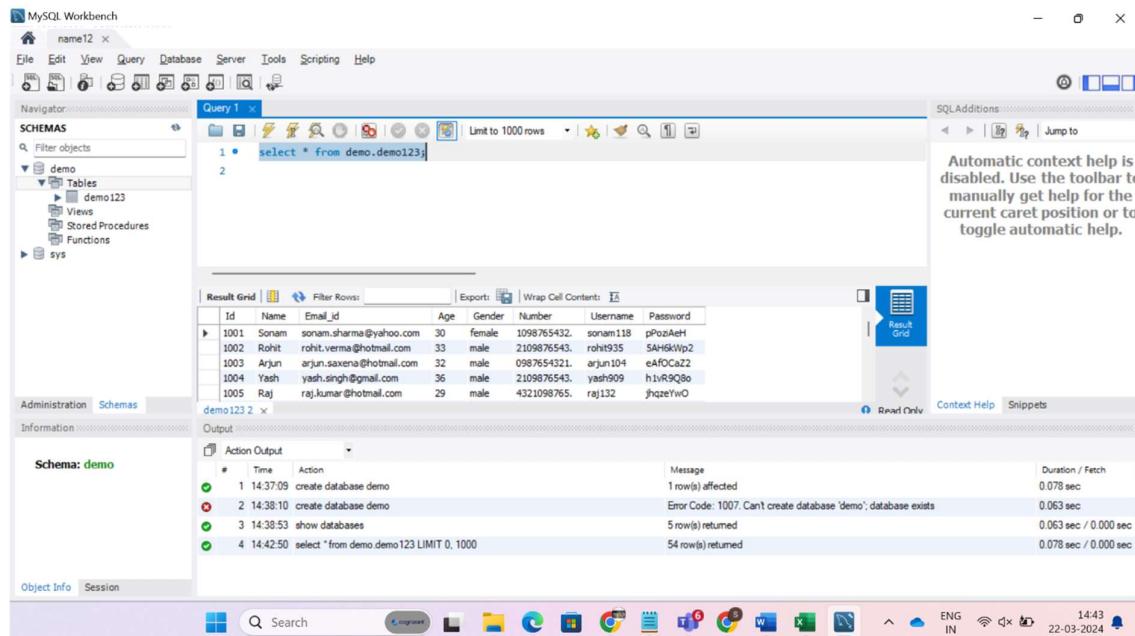
- Job 28 View (Stages: 1/1)
- Job 29 View (Stages: 1/1, 1 skipped)

```

1
Command skipped
  
```

Check load data in Mysql cloud:

- Open Mysql work bench and enter the command. Using select statement.
- You will see the loaded data.



Chapter-3

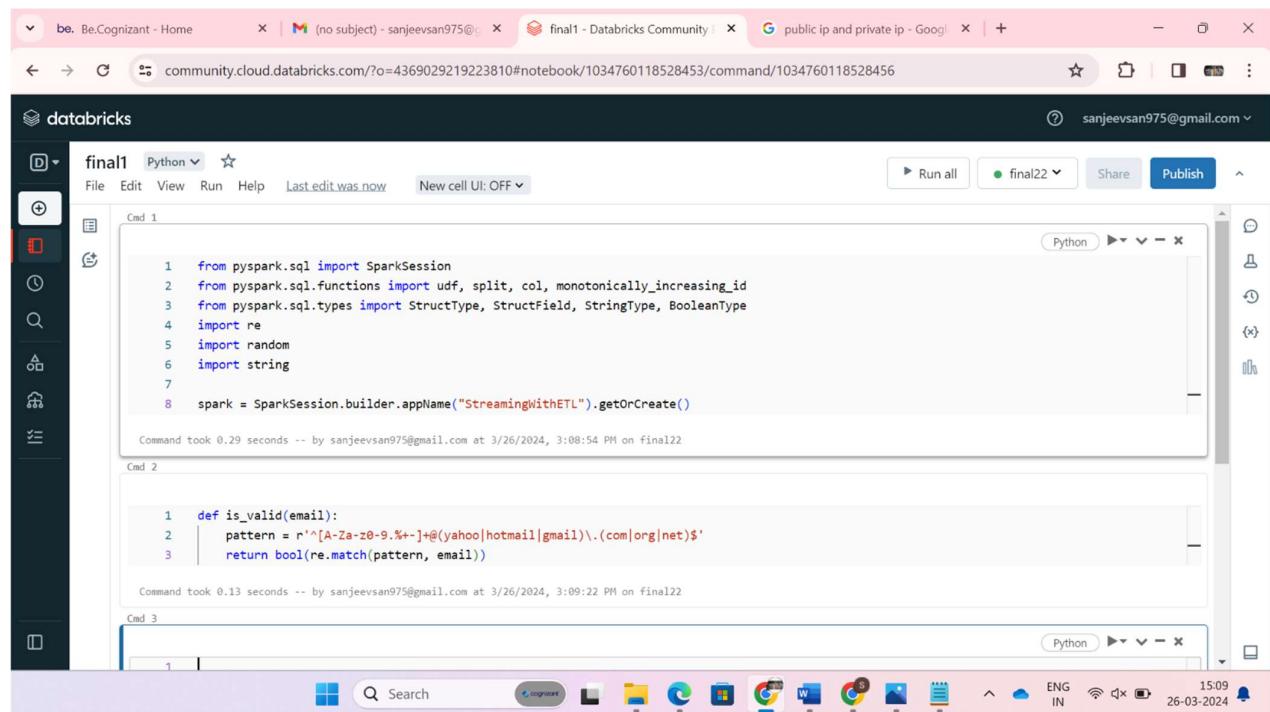
STREAMING USING ETL:

1) Imports:

It imports necessary modules and functions from PySpark, such as SparkSession, udf, split, col, monotonically_increasing_id, StructType, StructField, StringType, BooleanType, as well as standard Python modules like re (regular expressions), random, and string.

2) SparkSession:

It creates a SparkSession named "StreamingWithETL" or retrieves an existing one if available



The screenshot shows a Databricks notebook titled 'final1' in Python. The notebook contains two code cells. The first cell (Cmd 1) imports necessary modules and creates a SparkSession named 'StreamingWithETL'. The second cell (Cmd 2) defines a function 'is_valid' to check if an email address is valid based on a regular expression pattern. Both cells were run successfully, with execution times of 0.29 seconds and 0.13 seconds respectively. The notebook interface includes a sidebar with various icons and a status bar at the bottom showing system information like battery level, signal strength, and date/time.

```
1  from pyspark.sql import SparkSession
2  from pyspark.sql.functions import udf, split, col, monotonically_increasing_id
3  from pyspark.sql.types import StructType, StructField, StringType, BooleanType
4  import re
5  import random
6  import string
7
8  spark = SparkSession.builder.appName("StreamingWithETL").getOrCreate()

Command took 0.29 seconds -- by sanjeevsan975@gmail.com at 3/26/2024, 3:08:54 PM on final22

Cmd 2

1  def is_valid(email):
2      pattern = r'^[A-Za-z0-9.%+-]+@[yahoo|hotmail|gmail]\.(com|org|net)$'
3      return bool(re.match(pattern, email))

Command took 0.13 seconds -- by sanjeevsan975@gmail.com at 3/26/2024, 3:09:22 PM on final22
```

3) User-Defined Functions (UDFs):

- **`is_valid`**: A function to validate email addresses based on a regular expression pattern.

- **`generate_username`**: A function to generate a username from a given name.

- **`generate_password`**: A function to generate a random password.

The screenshot shows a Databricks notebook titled "final1" in Python. It contains three code cells (Cmd 2, Cmd 3, Cmd 4) and one command cell (Cmd 3). The code in Cmd 2 defines a function `is_valid` to validate email addresses. The code in Cmd 3 defines a function `generate_username` to generate a username from a name. The command in Cmd 3 runs the `generate_username` function. The output of the command cell shows the generated username `n{r}`. The status bar at the bottom indicates the command took 0.10 seconds and was run by sanjeevsan975@gmail.com at 3/26/2024, 3:11:21 PM on final22.

```
1 def is_valid(email):
2     pattern = r'^[A-Za-z-0-9.%+-]+@(yahoo|hotmail|gmail)\.(com|org|net)$'
3     return bool(re.match(pattern, email))

Command took 0.10 seconds -- by sanjeevsan975@gmail.com at 3/26/2024, 3:11:21 PM on final22

1
[Shift+Enter] to run and move to next cell
[Esc] to cancel
15:11
26-03-2024
```

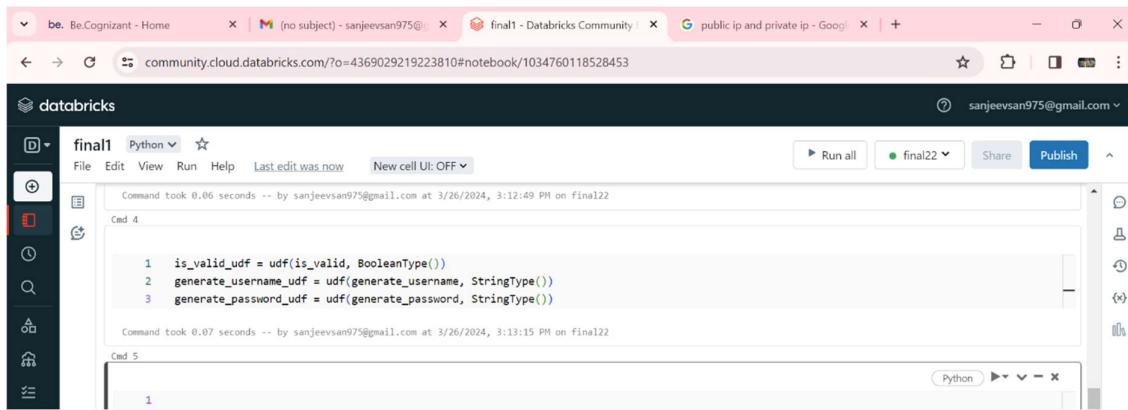
The screenshot shows a Databricks notebook titled "final1" in Python. It contains three code cells (Cmd 2, Cmd 3, Cmd 4) and one command cell (Cmd 3). The code in Cmd 2 defines a function `is_valid` to validate email addresses. The code in Cmd 3 defines a function `generate_username` to generate a username from a name. The code in Cmd 4 defines a function `generate_password` to generate a random password. The command in Cmd 3 runs the `generate_username` function. The output of the command cell shows the generated username `n{r}`. The command in Cmd 4 runs the `generate_password` function. The output of the command cell shows a randomly generated password of length 8. The status bar at the bottom indicates the command took 0.06 seconds and was run by sanjeevsan975@gmail.com at 3/26/2024, 3:12:49 PM on final22.

```
1 def generate_username(name):
2     name.lower()
3     r=random.randint(100, 999)
4     return f"{name}{r}"

1
[Shift+Enter] to run and move to next cell
[Esc] to cancel
15:11
26-03-2024
```

4) UDF Registration:

Registers the UDFs using the `udf` function provided by PySpark.



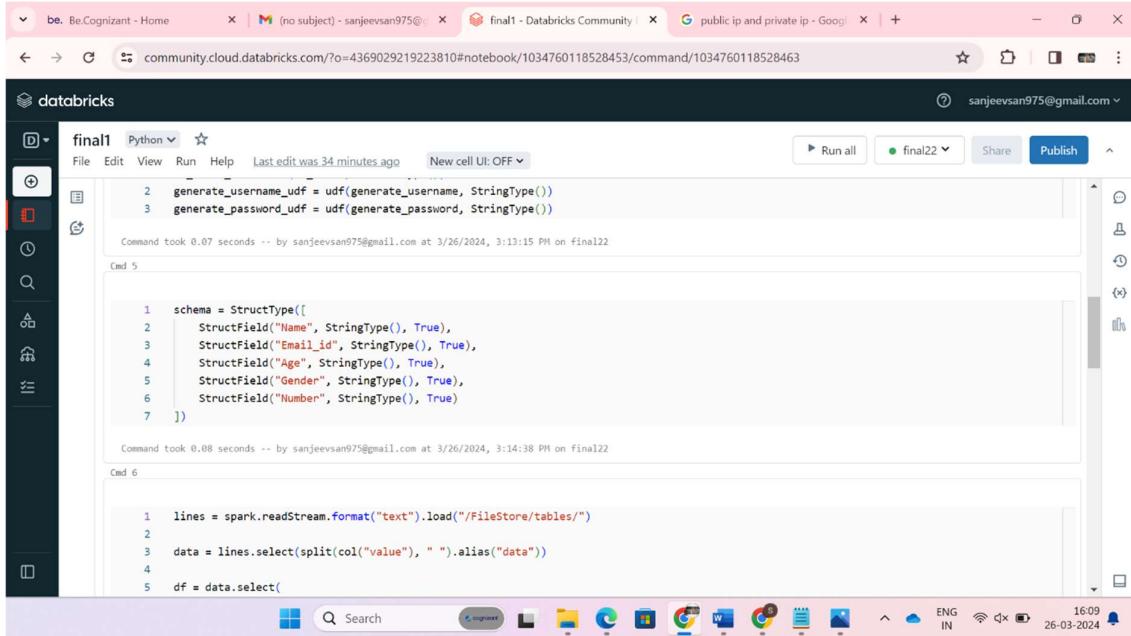
The screenshot shows a Databricks notebook titled "final1" in Python. The code in the cell registers three UDFs:

```
1 is_valid_udf = udf(is_valid, BooleanType())
2 generate_username_udf = udf(generate_username, StringType())
3 generate_password_udf = udf(generate_password, StringType())
```

The cell output shows the command took 0.06 seconds.

5) Schema Definition:

Defines the schema for the structured streaming data. The schema includes fields for "Name", "Email id", "Gender", and "Number".



The screenshot shows a Databricks notebook titled "final1" in Python. The code defines a schema and reads data from a file store.

```
1 schema = StructType([
2     StructField("Name", StringType(), True),
3     StructField("Email_id", StringType(), True),
4     StructField("Age", StringType(), True),
5     StructField("Gender", StringType(), True),
6     StructField("Number", StringType(), True)
7 ])
```

```
1 lines = spark.readStream.format("text").load("/FileStore/tables/")
2 data = lines.select(split(col("value"), " ").alias("data"))
3 df = data.select(
```

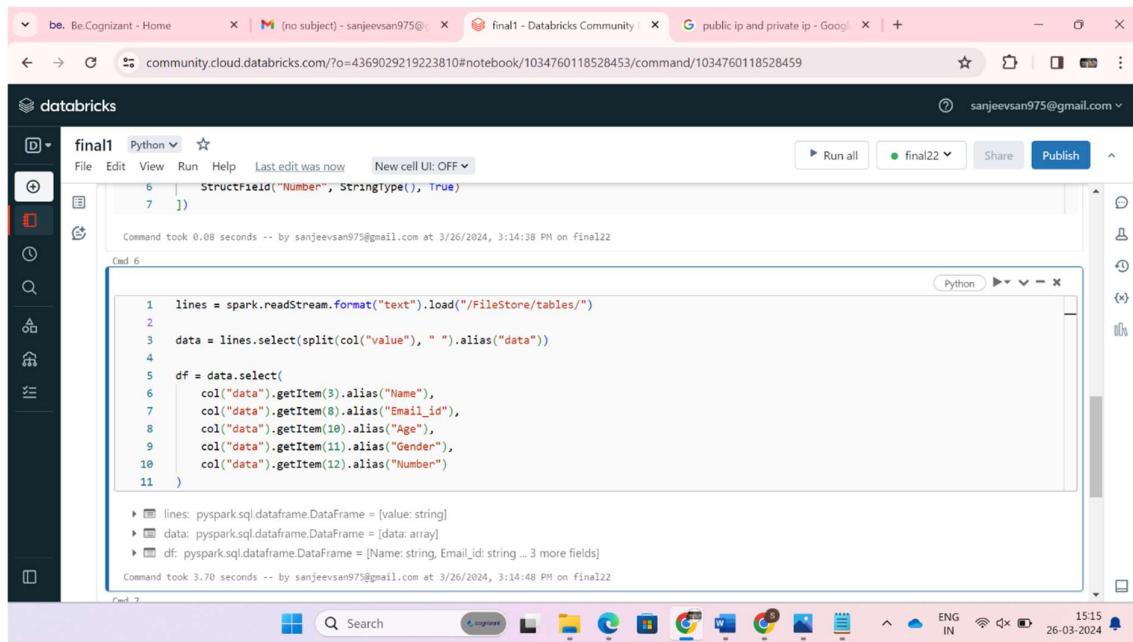
The cell output shows the command took 0.08 seconds.

6) Reading Stream:

- Reads streaming data from a specified location ("FileStore/tables") as text files.

Data Transformation:

- Splits the lines into columns using the 'split' function and assigns the result to a column named "data".
 - Selects specific columns from the "data" column to create a DataFrame ('df') with columns "Name", "Email_id", "Age", "Gender", and "Number".



The screenshot shows a Databricks notebook interface with the following details:

- Title:** final1
- Languages:** Python
- Cells:** 1 cell run, 1 cell pending, 1 cell published.
- Code:**

```
1 lines = spark.readStream.format("text").load("/FileStore/tables/")
2
3 data = lines.select(split(col("value"), " ").alias("data"))
4
5 df = data.select(
6     col("data").getItem(3).alias("Name"),
7     col("data").getItem(8).alias("Email_id"),
8     col("data").getItem(10).alias("Age"),
9     col("data").getItem(11).alias("Gender"),
10    col("data").getItem(12).alias("Number")
11 )
```
- Output:** Command took 0.08 seconds -- by sanjeevsan975@gmail.com at 3/26/2024, 3:14:38 PM on final22
- Code:**

```
1 lines = spark.readStream.format("text").load("/FileStore/tables/")
2
3 data = lines.select(split(col("value"), " ").alias("data"))
4
5 df = data.select(
6     col("data").getItem(3).alias("Name"),
7     col("data").getItem(8).alias("Email_id"),
8     col("data").getItem(10).alias("Age"),
9     col("data").getItem(11).alias("Gender"),
10    col("data").getItem(12).alias("Number")
11 )
```
- Output:** Command took 3.70 seconds -- by sanjeevsan975@gmail.com at 3/26/2024, 3:14:48 PM on final22

7) Data Deduplication:

- Removes duplicate records from the DataFrame ('df') using the 'distinct' method.

```

> df: pyspark.sql.dataframe.DataFrame = [Name: string, Email_id: string ... 3 more fields]
Command took 3.70 seconds -- by sanjeevsan975@gmail.com at 3/26/2024, 3:14:48 PM on final22
Cmd 7
1 dd=df.distinct()
Cancel Uploading command
Cmd 8
1

```

8) Data Validation:

- Adds a column "Valid Email" to indicate whether the email addresses are valid by applying the `is_valid_udf` function.
- Filters out rows where the email address is valid.
- Drops the "Valid_Email" column from the DataFrame

```

be. Be.Cognizant - Home | M (no subject) - sanjeevsan975@... | final1 - Databricks Community | public ip and private ip - Google | + | Run all | final22 | Share | Publish | Sanjeev San975@gmail.com
databricks
final1 Python
File Edit View Run Help Last edit was now New cell UI: OFF
1 dd=df.distinct()
dd: pyspark.sql.dataframe.DataFrame = [Name: string, Email_id: string ... 3 more fields]
Command took 0.17 seconds -- by sanjeevsan975@gmail.com at 3/26/2024, 3:29:12 PM on final22
Cmd 8
1 df_validated = dd.withColumn("Valid_Email", is_valid_udf(col("Email_id"))).filter(col("Valid_Email") == True)
2 df_fin=df_validated.drop("Valid_Email")
df_validated: pyspark.sql.dataframe.DataFrame = [Name: string, Email_id: string ... 4 more fields]
df_fin: pyspark.sql.dataframe.DataFrame = [Name: string, Email_id: string ... 3 more fields]
Command took 0.19 seconds -- by sanjeevsan975@gmail.com at 3/26/2024, 3:30:10 PM on final22
Cmd 9
1
[Shift+Enter] to run and move to next cell
[Esc H] to see all keyboard shortcuts

```

Additional Data Transformation:

- Adds a "Username" column by applying the `generate_username_udf` function.
- Adds a "Password" column by applying the `generate_password_udf` function.



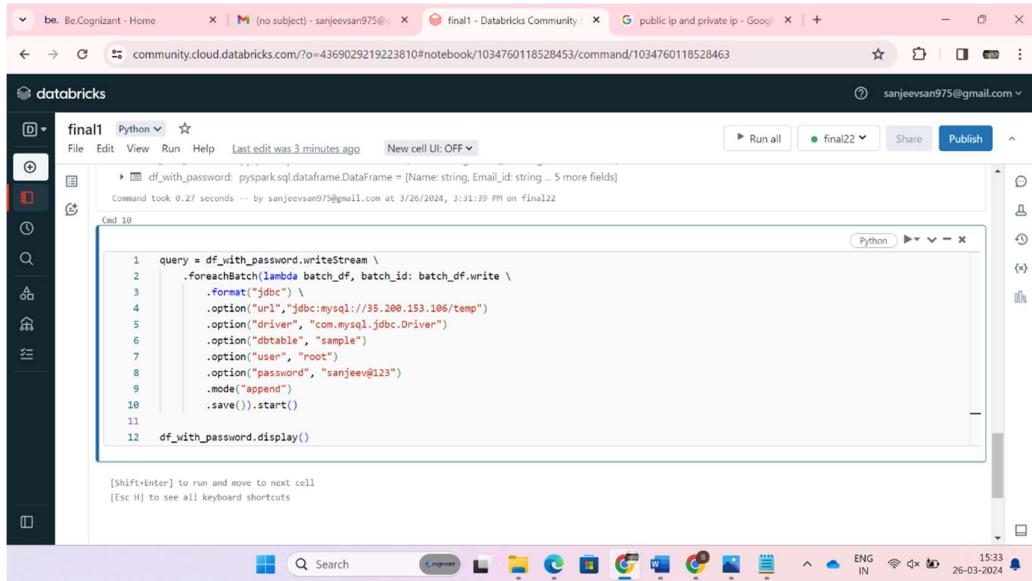
The screenshot shows a Jupyter Notebook interface with a sidebar containing icons for file operations, search, and help. The main area displays Python code for generating new columns in a DataFrame:

```
1 df_fin: pyspark.sql.DataFrame = [Name: string, Email_id: string ... 3 more fields]
2 Command took 0.19 seconds -- by sanjeevan975@gmail.com at 3/26/2024, 3:30:10 PM on final22
Cmd 9
1 df_with_username = df_fin.withColumn("Username", generate_username_udf(col("Name")))
2 df_with_password = df_with_username.withColumn("Password", generate_password_udf())
3 df_with_username: pyspark.sql.DataFrame = [Name: string, Email_id: string ... 4 more fields]
4 df_with_password: pyspark.sql.DataFrame = [Name: string, Email_id: string ... 5 more fields]
```

9) Writing to Sink in MySQL Cloud:

- Writes the processed data to a JDBC sink (MySQL Cloud) using a foreach Batch writer.
- Specifies connection parameters such as URL IP Address, table name, username, and password.
- Create mysql instance in cloud.

-In mysql workbench create new connection and link with Cloud data.

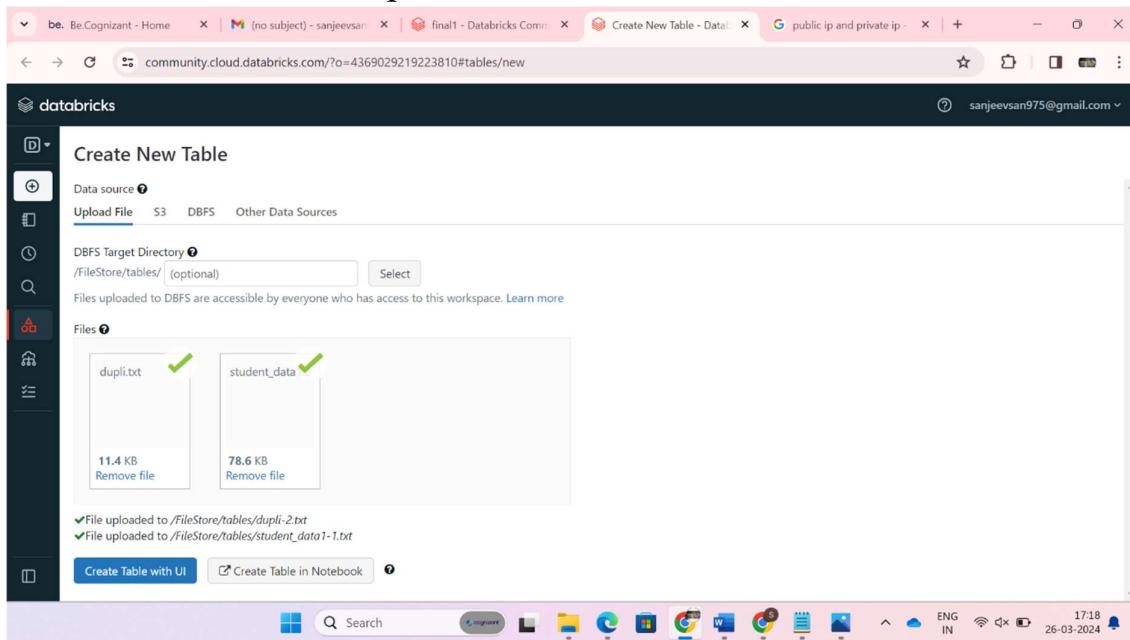


The screenshot shows a Databricks Notebook interface. The notebook has a single cell titled 'final1' in Python. The code in the cell is:

```
1 query = df_with_password.writeStream \
2     .foreachBatch(lambda batch_df, batch_id: batch_df.write \
3         .format("jdbct") \
4         .option("url", "jdbc:mysql://35.200.153.106/temp") \
5         .option("driver", "com.mysql.jdbc.Driver") \
6         .option("dbtable", "sample") \
7         .option("user", "root") \
8         .option("password", "sanjeev@123") \
9         .mode("append") \
10        .save())
11 df_with_password.display()
```

Below the code, there is a note: '[Shift+Enter] to run and move to next cell [Esc H] to see all keyboard shortcuts'. The status bar at the bottom right shows the date as 26-03-2024 and the time as 15:33.

- Now the streaming is started. Now we will insert the file in data bricks as a steam process.



The screenshot shows the 'Create New Table' interface in Databricks. Under 'Data source', 'Upload File' is selected. The 'DBFS Target Directory' field contains '/FileStore/tables/'. Below the file list, it says 'Files uploaded to DBFS are accessible by everyone who has access to this workspace. Learn more'. There are two files listed: 'dupli.txt' (11.4 KB) and 'student_data' (78.6 KB). Both files have a green checkmark icon. At the bottom, there are buttons for 'Create Table with UI' and 'Create Table in Notebook'. The status bar at the bottom right shows the date as 26-03-2024 and the time as 17:18.

The screenshot shows a Databricks notebook titled 'final' in Python. The notebook interface includes a sidebar with various icons, a top bar with navigation links and cluster status, and a central workspace. In the workspace, there are two notebooks listed: '(2) Spark Jobs' and 'display_query_2'. Below these, a table named 'final' is displayed with the following schema and data:

	Name	Email_id	Age	Gender	Number	Username	Password
1	Kartik	kartik.agarwal@gmail.com	33	male	4321098765.	kartik214	6d5jfwR0
2	Rishi	rishi.saxena@gmail.com	32	male	6543210987.	rishi857	dTrdOeq
3	Priyanka	priyanka.patel@gmail.com	26	female	1098765432.	priyanka495	TXrAUCuj
4	Priya	priya.sharma@gmail.com	29	female	9876543210.	priya120	iuqtfHzZF
5	Sonam	sonam.sharma@yahoo.com	30	female	1098765432.	sonam578	xLREwbp7
6	Arjun	arjun.gupta@gmail.com	29	male	0987654321.	arjun907	8HEW97QE
7	Rohit	rohit.verma@hotmail.com	33	male	2109876543.	rohit175	NZ4o2obl

At the bottom of the table, it says '387 rows'.

Now data will be stored in MySQL cloud. To get that data we need to open MySQL workbench and using select statement we can see that data.

The screenshot shows the MySQL Workbench interface. The 'Schemas' pane on the left lists 'sys' and 'temp'. The 'Query 1' pane contains the following SQL code:

```
1 • create database temp;
2 • select * from temp.sample;
3 • select count(*) from temp.sample;
```

The 'Result Grid' pane below shows the output of the third query:

count(*)
334

The 'Output' pane at the bottom displays the execution log:

#	Time	Action	Message	Duration / Fetch
1	16:05:11	create database temp	1 row(s) affected	0.188 sec
2	16:13:17	select * from temp.sample LIMIT 0, 1000	54 row(s) returned	0.672 sec / 0.000 sec
3	16:15:24	select count(*) from temp.sample LIMIT 0, 1000	1 row(s) returned	1.063 sec / 0.000 sec
4	16:16:44	select count(*) from temp.sample LIMIT 0, 1000	1 row(s) returned	0.109 sec / 0.000 sec

- before creating a instance in MySQL cloud.

The screenshot shows the Google Cloud SQL Instances page. At the top, there's a navigation bar with tabs for 'SQL' and 'Instances'. Below the navigation is a search bar and a 'CREATE INSTANCE' button. A table lists the instance details:

Instance ID	Issues	Cloud SQL edition	Type	Public IP address	Private IP address	Instance connection name	High availability	Actions
<input type="checkbox"/> sanjeev123		Enterprise	MySQL 8.0	35.200.153.106		sanjeev0204-cts.asi...	ENABLE	⋮

The taskbar shows a notification from 'Uploads and 0403Sanjeev0204 CTS operations' indicating a file was created at 4:03:56 PM GMT+5 on 25-03-2024.

Conclusion:

Streaming and batch ETL are indispensable components of modern data processing architectures, offering organizations the flexibility to process data in real-time and batch modes according to their specific use cases and requirements. By leveraging both methodologies, organizations can derive actionable insights, optimize decision-making processes, and unlock new opportunities for innovation and growth in today's data-driven landscape.