**Data bricks Use Cases**

Table of Contents

[1. SUMMARY OF USECASES 3](#_Toc212211607)

[2. DATABRICKS USECASES SIMPLE (3-5hrs) 4](#_Toc212211608)

[Use Case 1: Use pandas Data frame 4](#_Toc212211609)

[Use Case 2: Use Spark Data frame 4](#_Toc212211610)

[Use Case 3: Use Pandas API on Spark (pyspark.pandas) 5](#_Toc212211611)

[Use Case 4: Schema Creation and Data Reading & Filter Orders by Customer Name 5](#_Toc212211612)

[Use Case 5: Orders Placed in December 2020, Customer Name Filter 5](#_Toc212211613)

[2. DATABRICKS USECASES MEDIUM (6-8hrs) 5](#_Toc212211614)

[Use Case 1: Drivers Data Analysis 5](#_Toc212211615)

[Use Case 2: Customer Purchase Insights 5](#_Toc212211616)

[Use Case 3: ICC World Cup Statistics 6](#_Toc212211617)

[Use Case 4: Order Analysis 6](#_Toc212211618)

[Use Case 5: Sales Performance Analysis 6](#_Toc212211619)

[Use Case 6: Employee Performance Evaluation 6](#_Toc212211620)

[Use Case 7: Real-Time Customer Order Tracking 7](#_Toc212211621)

[3. DATABRICKS USECASES COMPLEX (14-18hrs) 7](#_Toc212211622)

[Use Case 1: COVID-19 Hospitalization 7](#_Toc212211623)

[BRONZE LAYER - Raw Data Ingestion 7](#_Toc212211624)

[SILVER LAYER - Cleaned & Enriched Data 8](#_Toc212211625)

[GOLD LAYER - Business-Ready Aggregated Data 8](#_Toc212211626)

[4. PROJECT FOR HANDS-ON PRACTICE (3 Days) 10](#_Toc212211627)

[Project Overview: 10](#_Toc212211628)

[Business Requirements: 10](#_Toc212211629)

[Technical Requirements 10](#_Toc212211630)

[Architecture Overview 10](#_Toc212211631)

[Step-by-Step Implementation 10](#_Toc212211632)

[Output Validation 10](#_Toc212211633)

[Error Handling and Debugging 11](#_Toc212211634)

[Future Enhancements 11](#_Toc212211635)

[Key Learnings 11](#_Toc212211636)

[5. References 11](#_Toc212211637)

# 1. SUMMARY OF USECASES

1. DATA BRICKS USE CASES (SIMPLE)  
This section covers beginner-level Databricks use cases. It introduces basic operations using Pandas DataFrames, Spark DataFrames, Pandas API on Spark, schema creation, data reading, and a simple data filtering scenario like retrieving orders placed in December.

2. DATA BRICKS USE CASES (Medium)  
Intermediate-level use cases for practical Databricks applications. Includes data analysis tasks like Drivers Data Analysis, Customer Purchase Insights, ICC World Cup Statistics, Order Analysis, Sales Performance Analysis, Employee Performance Evaluation, and Real-Time Customer Order Tracking. These examples demonstrate both batch processing and basic analytics on larger datasets.

3. DATA BRICKS USE CASES (COMPLEX)  
Advanced-level use cases focusing on complex data scenarios. Examples include COVID-19 Hospitalization analysis, which involves multiple datasets, advanced transformations, and more sophisticated logic suitable for real-world applications.

4. PROJECT FOR HANDS-ON PRACTICE  
A guided project to consolidate learning from previous use cases. Includes:

* Project Overview – Introduction and objectives.
* Business Requirements – What the project aims to achieve.
* Technical Requirements – Tools, libraries, and infrastructure needed.
* Architecture Overview – System design and data flow.
* Step-by-Step Implementation – Practical coding and pipeline steps.
* Output Validation – Methods to verify the results.
* Error Handling and Debugging – Common errors and their resolutions.
* Key Learnings – Summary of skills and insights gained.

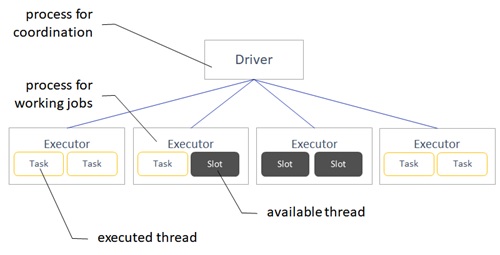
5. REFERENCES  
List of resources, documentation, tutorials, and websites used for completing the use cases and project.

# 2. DATABRICKS USECASES SIMPLE (3-5hrs)

## [Use Case 1: Use pandas Data frame](https://vmivsp-my.sharepoint.com/personal/sanjyoti_gudimetla_valuemomentum_com/_layouts/15/Doc.aspx?sourcedoc=%7B7070766D-9A19-49B5-81A8-EA968D99FC94%7D&file=Databricks_UseCases%20Documentation.docx&action=default&mobileredirect=true)

You can run primitive Python scripts and use regular Pandas dataframe in Databricks. But, how does it differ from using Spark-compliant Pyspark, Spark Dataframe, or Spark ML programmig ?

Apache Spark is in-memory distribution platform for processing large scale of data.  
When you submit a Spark job in Apache Spark (such as, running code in notebook or submitting a job), a single job is decomposed to physical executions called stages. Each stage runs multiple tasks, which are the real execution's units running on Spark executor (worker) process. (See below.)



These decompositions (such as, logical execution plans and physical execution plans) are transparent, when you use Spark-oriented libraries or components, such as, PySpark, Spark SQL, or other spark components.

In this exercise, you will learn :

* First, run ordinary Python code with pandas dataframe, and see that it runs only on driver (master) and not distributed on Spark executors.
* Next, rewrite and run the same code with PySpark and Spark Dataframe, and see that it runs as Spark worker jobs in distributed manners.
* Finally, I'll introduce new pyspark.pandas (formerly, Koalas), in which you can use familiar pandas syntax in Spark distributed manners.

[Use Case 1: Use pandas Data frame](#_Use_Case_1:_1)

Scenario: First, we use the familiar pandas data frame with a scikit-learn framework.  
All operations run on a single driver and will not be distributed in Spark workers.

## Use Case 2: Use Spark Data frame

Scenario: Next we will use Spark data frame and scalable Spark ML libraries.  
You will find that operations are invoked as Spark jobs and will be scaled on Spark clusters.

## 

## Use Case 3: Use Pandas API on Spark (pyspark.pandas)

Scenario: Finally, we will use pyspark.pandas dataframe. This code is also distributed as Spark jobs, while we can use familiar pandas syntax.

To run this example, use Databricks Runtime 10.0 (Spark 3.2) or above.

Scenario: Once you have got pyspark.pandas dataframe, you can wirte with same syntax as familiar pandas dataframe. (Compare with above code of pandas datafrme.)  
The data is processed in distributed manners on Spark executors.

## Use Case 4: Schema Creation and Data Reading & Filter Orders by Customer Name

Scenario: You have a CSV file containing orders data. Define a schema using Struct Type and Struct Field by inferring the data types from the file and read it using PySpark.

Scenario: Retrieve all orders where the customer's name has 'a' as the second character and 'd' as the fourth character.

## Use Case 5: Orders Placed in December 2020, Customer Name Filter

Scenario: Retrieve all orders where the order\_date is in December 2020.

Scenario: Retrieve all orders where the customer's name neither starts with 'A' nor ends with 'n'.

Scenario: Retrieve all orders where the profit is negative.

Scenario: Perform inner, left, right, and full joins on Data Frames. Apply aggregations like sum, avg, count, and filters to extract required subsets.

# 2. [DATABRICKS USECASES MEDIUM (6-8hrs)](#_2._Medium_(6-8hrs))

## Use Case 1: Drivers Data Analysis

Scenario: Find Total rides and Profit Rides where Profit rides are the number of rides where end location of a ride is same as start location of immediate next ride for a driver Table. Structure: create table drivers (id varchar (10), start\_time time, end\_time time, start\_loc varchar (10), end\_loc varchar (10));

Sample Data:

insert into drivers values('dri\_1', '09:00', '09:30', 'a','b'),('dri\_1', '09:30', '10:30', 'b','c'),('dri\_1','11:00','11:30', 'd','e');   
insert into drivers values('dri\_1', '12:00', '12:30', 'f','g'),('dri\_1', '13:30', '14:30', 'c','h');   
insert into drivers values('dri\_2', '12:15', '12:30', 'f','g'),('dri\_2', '13:30', '14:30',

## Use Case 2: Customer Purchase Insights

Scenario: Query to print customer name and no of occurence of character 'n' in the customer name.   
customer\_name , count\_of\_occurence\_of\_n

Table Structure: customers(customer\_id, name, purchase\_date, amount)

Sample Data:

(101, 'Alice', '2024-06-01', 200)

(102, 'Bob', '2024-06-05', 350)

(101, 'Alice', '2024-06-12', 150)

## Use Case 3: ICC World Cup Statistics

Scenario: Find to produce below output from icc\_world\_cup table.   
team\_name, no\_of\_matches\_played , no\_of\_wins , no\_of\_losses

Table Structure: create table icc\_world\_cup   
(   
Team\_1 Varchar(20),   
Team\_2 Varchar(20),   
Winner Varchar(20)   
);

Sample Data:

INSERT INTO icc\_world\_cup values('India','SL','India');   
INSERT INTO icc\_world\_cup values('SL','Aus','Aus');   
INSERT INTO icc\_world\_cup values('SA','Eng','Eng');   
INSERT INTO icc\_world\_cup values('Eng','NZ','NZ');   
INSERT INTO icc\_world\_cup values('Aus','India','India'); 

## Use Case 4: Order Analysis

Problem Statement: find premium customers from orders data. Premium customers are those who have done more orders than average no of orders per customer.

Table Structure: orders(order\_id, item, quantity, price)

Sample Data:

* (1001, 'Laptop', 2, 80000)
* (1002, 'Mouse', 5, 500)
* (1003, 'Keyboard', 3, 1500)

## Use Case 5: Sales Performance Analysis

**Scenario:** You work for a retail company that wants to analyze sales performance across different **regions** and **products**. Management wants insights like:

* Total sales per region.
* Total sales per product.
* Total sales per region and product combination.
* Overall total sales.

## 

## Use Case 6: Employee Performance Evaluation

**Scenario:** print emp name, salary and dep id of highest salaried employee in each department

Table Structure: employees(emp\_id, emp\_name, department, salary)

Sample Data:

* (1, 'Raj', 'Data', 85000)
* (2, 'Meena', 'HR', 60000)
* (3, 'Karan', 'Data', 95000)

## Use Case 7: Real-Time Customer Order Tracking

**Scenario:** An e-commerce company wants to track customer orders in near real-time to monitor order volume, region-wise performance, and high-demand products.

**Requirements:**

Stream order events from Kafka or a message queue.

Aggregate order data by region, product, and order status.

Maintain historical order data for trend analysis.

Flag products with unusually high order volume.

# 3. DATABRICKS USECASES COMPLEX (14-18hrs)

## Use Case 1: COVID-19 Hospitalization

Build a complete **Medallion Architecture** (Bronze → Silver → Gold) using Delta Lake for COVID-19 hospitalization data.

## BRONZE LAYER - Raw Data Ingestion

**Requirements:**

1. **Load the raw CSV file** into a Databricks DataFrame
2. **Add metadata columns:**

* ingestion\_timestamp (when data was loaded)
* source\_file (filename)
* data\_quality\_flag (for future validation)

1. **Save as Delta table** with these specifications:

* Table name: covid\_bronze
* Partitioned by: date (year and month)
* Enable Change Data Feed (CDC)

1. **Data validation:**
   * Check for duplicate records
   * Record count verification
   * Schema validation

Expected Columns:

* All original columns from CSV +
* ingestion\_timestamp
* source\_file
* data\_quality\_flag

Questions to Answer:

* How many total records in Bronze?
* How many unique countries?
* What's the date range?
* Are there any duplicate rows?

## SILVER LAYER - Cleaned & Enriched Data

**Requirements:**

1. **Data Cleaning:**
   * Remove non-country entities (World, continents, income groups)
   * Filter for 8 specific countries: USA, UK, Germany, France, Italy, Spain, India, Brazil
   * Handle NULL values in key columns: hosp\_patients, icu\_patients
   * Remove duplicate records based on location and date
   * Convert data types correctly
2. **Data Enrichment - Add Calculated Columns:**
   * icu\_to\_hosp\_ratio: ICU patients / Total hospitalized patients
   * hosp\_per\_million: Hospitalization rate per 1 million population
   * icu\_per\_million: ICU rate per 1 million population
   * year: Extracted from date
   * month: Extracted from date
   * quarter: Q1, Q2, Q3, Q4
   * day\_of\_week: Monday, Tuesday, etc.
   * Is\_weekend: Boolean flag
3. **Rolling Calculations (Important!):**
   * hosp\_7day\_avg: 7-day rolling average of hospitalizations
   * hosp\_7day\_change: Change from 7 days ago
   * icu\_7day\_avg: 7-day rolling average of ICU patients
4. **Save as Delta table:**
   * Table name: covid\_silver
   * Partitioned by: location, year, month
   * Add table properties for documentation
   * Enable optimize and Z-order by date

## GOLD LAYER - Business-Ready Aggregated Data

**Requirements:**

Create **THREE separate Gold tables:**

#### Table 1: covid\_gold\_monthly\_summary

* Monthly aggregations per country
* Columns:
  + location
  + year
  + month
  + avg\_hosp\_patients (monthly average)
  + max\_hosp\_patients (monthly peak)
  + total\_hosp\_days (sum of all daily hospitalizations)
  + avg\_icu\_patients
  + max\_icu\_patients
  + avg\_icu\_ratio
  + month\_over\_month\_change (% change from previous month)

#### Table 2: covid\_gold\_country\_peaks

* Peak values for each country
* Columns:
  + location
  + peak\_hosp\_date (date of maximum hospitalizations)
  + peak\_hosp\_value (maximum number of hospitalized)
  + peak\_icu\_date
  + peak\_icu\_value
  + total\_days\_with\_data
  + avg\_daily\_hosp
  + country\_ranking (rank by peak hospitalizations)

#### Table 3: covid\_gold\_weekly\_trends

* Weekly aggregations for dashboards
* Columns:
  + location
  + week\_start\_date
  + week\_end\_date
  + avg\_weekly\_hosp
  + week\_over\_week\_change
  + trend\_direction (Increasing/Decreasing/Stable)

Questions to Answer:

* Which country had the highest monthly peak?
* Which month had the most hospitalizations across all countries?
* Show week-over-week trends for USA.

# 

# 4. PROJECT FOR HANDS-ON PRACTICE (3 Days)

## Project Overview:

This project aims to build a unified Customer 360 profile by integrating multiple domain datasets — Customers, Policies, and Claims — stored in CSV format, using Delta Lake for handling full and incremental data loads with SCD Type-2 history tracking on Databricks.

## Business Requirements:

· Create a single source of truth (Customer 360 table) combining customer, policy, and claims data.

· Support incremental data ingestion to capture only new or updated records.

· Maintain historical changes (SCD Type-2).

· Implement using Databricks, Delta Lake, and PySpark.

· Automate execution using Databricks Jobs.

## Technical Requirements

Component Description

Environment Databricks Community/Enterprise

Storage Databricks Volumes (/Volumes/customers\_insurance/customer360/insurance\_data/)

Format CSV (input), Delta (output)

Language PySpark (Python)

Tables customer, policy, claims, customer\_360

Incremental Tables customer\_incremental, policy\_incremental,claims\_incremental

Database customers\_insurance.customer360

## Architecture Overview

The data flows from CSV files in Volumes through transformation and consolidation notebooks, finally producing a unified Customer 360 Delta table with historical tracking.

## Step-by-Step Implementation

1. Notebook 1 — Volume Setup: Created base folders in Volumes and uploaded CSV files.

2. Notebook 2 — Table DDL Creation: Created Delta tables (customer, policy, claims).

3. Notebook 3 — Base Data Load: Loaded data and added timestamps.

4. Notebook 4 — Build Customer 360: Joined base tables and created master table with SCD columns.

5. Notebook 5 — Incremental Load: Read incremental data and applied Delta MERGE (SCD Type-2).

## 

## Output Validation

Validated record counts and ensured correct flagging of active vs historical records. is\_current = 1 for active records, and end\_ts = NULL indicates current records.

## Error Handling and Debugging

|  |  |  |
| --- | --- | --- |
| Error | Cause | Resolution |
| jsparkSession not supported | Issue in transformation or merge step | Replaced with spark.catalog.ttable |
| Exists() | Issue in transformation or merge step | Corrected to created\_ts |
| DELTA\_MULTIPLE\_SOURCE\_ROW\_MATCHING\_TARGET\_ROW\_IN\_MERGE | Issue in transformation or merge step | Resolved using deduplication logic |

## 

## Future Enhancements

* Parameterize file paths and table names.
* Add metadata logging for job runs.
* Integrate with Power BI for analytics.
* Enable email alerts for job success/failure.
* Support for new datasets like renewals or payments.

## Key Learnings

This project enhanced understanding of Delta Lake SCD Type-2 operations, Databricks job orchestration, and modular ETL design using PySpark. It demonstrated how to handle data consistency, incremental merges, and schema evolution effectively.

# [References](#_5._References)

