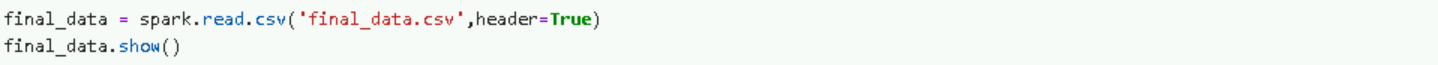
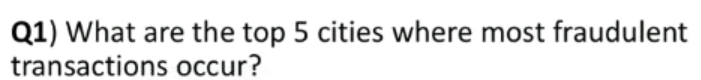
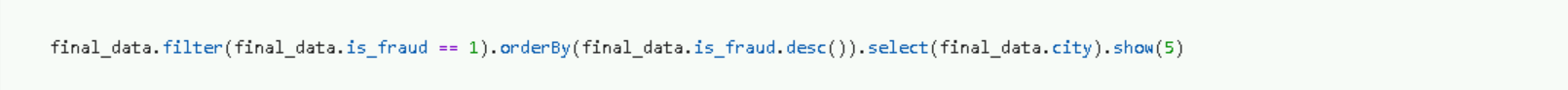
TEAM-D

Date:14-05-2025

#READING CSV FILE

1.QUESTION

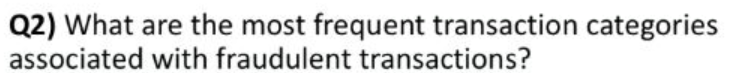
CODE



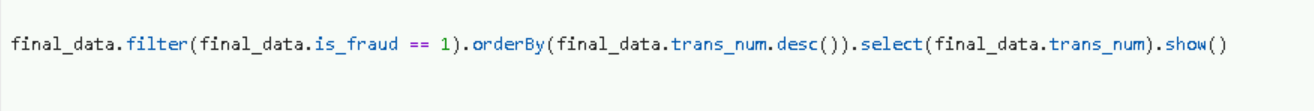
OUTPUT

A black and white text

AI-generated content may be incorrect.

2. QUESTION

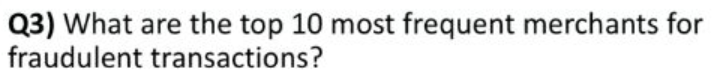
CODE



OUTPUT



3.QUESTION



 CODE



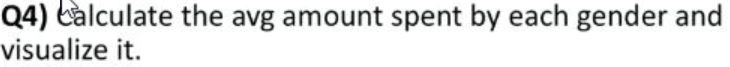
OUTPUT

A screenshot of a computer program

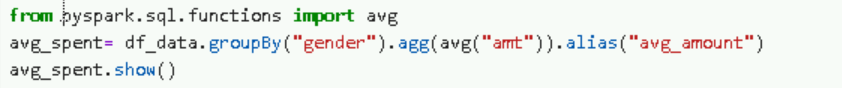
AI-generated content may be incorrect.A black and white text

AI-generated content may be incorrect.

4.QUESTION



CODE

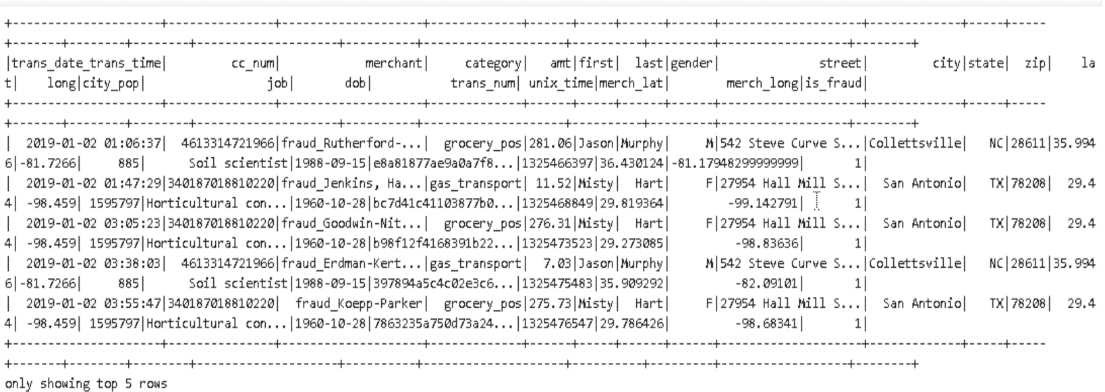
OUTPUTA close-up of a number

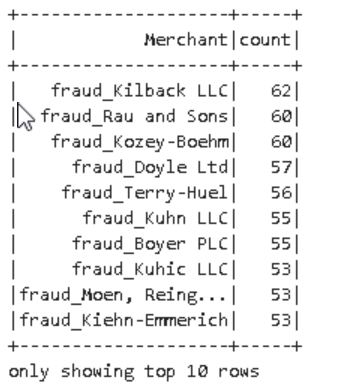
AI-generated content may be incorrect.

A close up of a sign

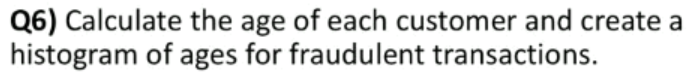
AI-generated content may be incorrect.5.QUESTION

CODE

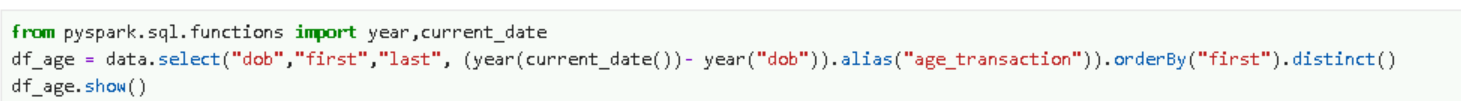
OUTPUT



6.QUESTION



CODE

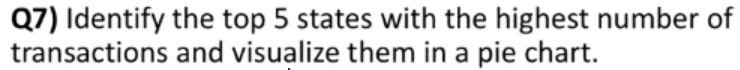


OUTPUT

A screenshot of a document

AI-generated content may be incorrect.

7.QUESTION



 CODE

OUTPUT

A screenshot of a computer

AI-generated content may be incorrect.

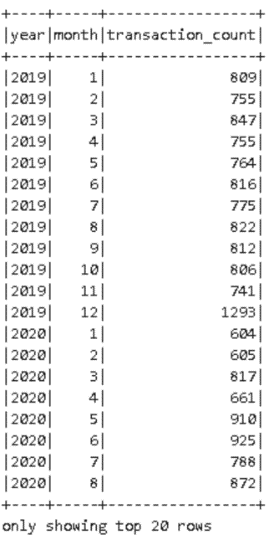
8.QUESTION



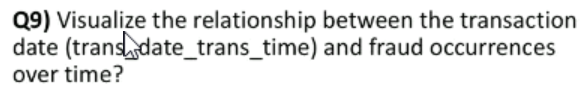
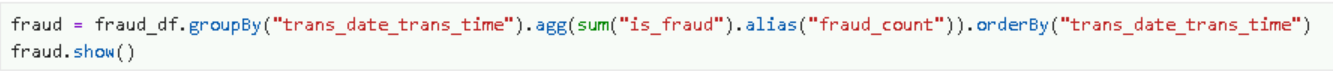
CODE



OUTPUT



9.QUESTION

CODE

OUTPUT

A table of numbers with numbers

AI-generated content may be incorrect.

10.QUESTION

A close-up of a text

AI-generated content may be incorrect.

CODE AND OUPUT



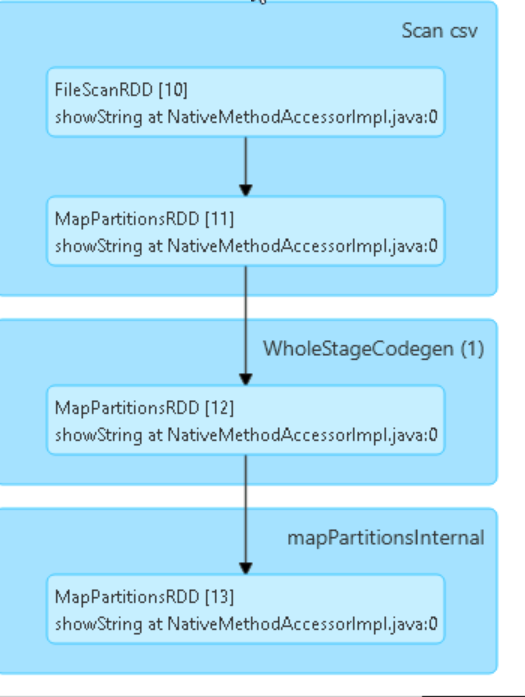
A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**UI ANALYSIS**

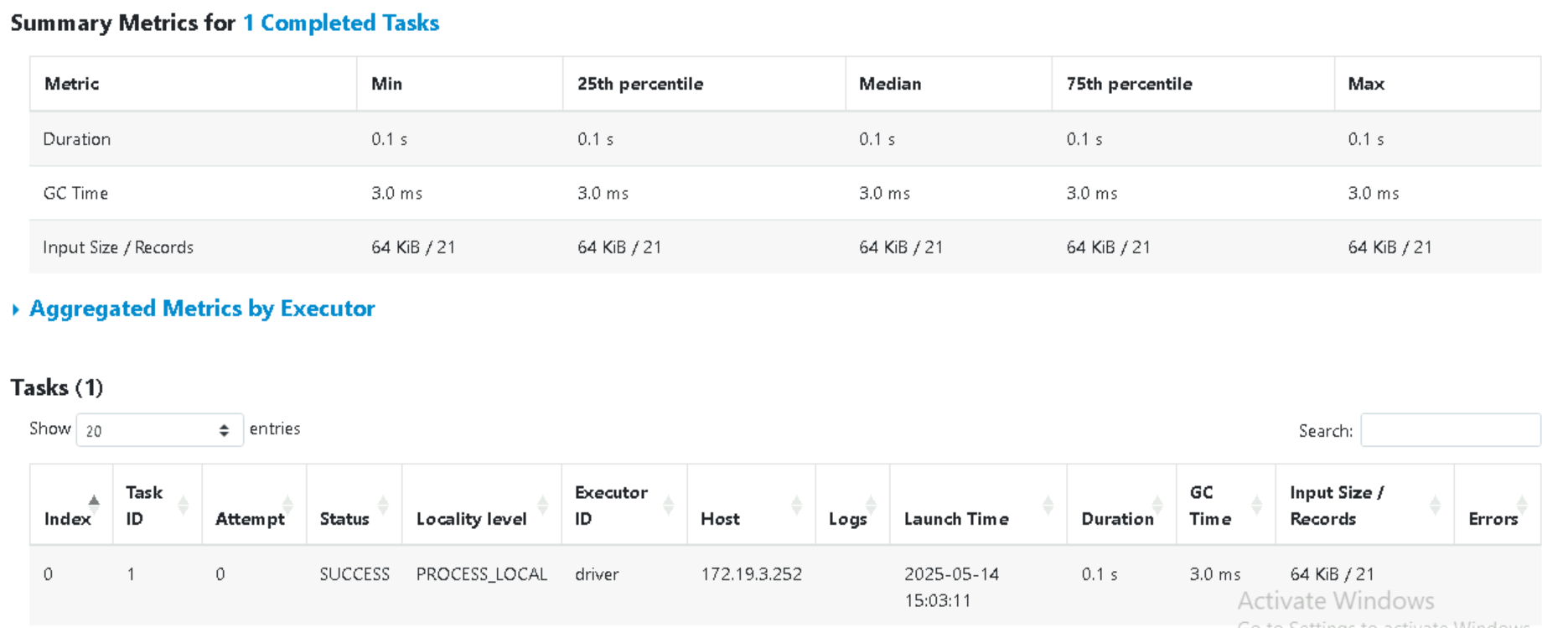
****

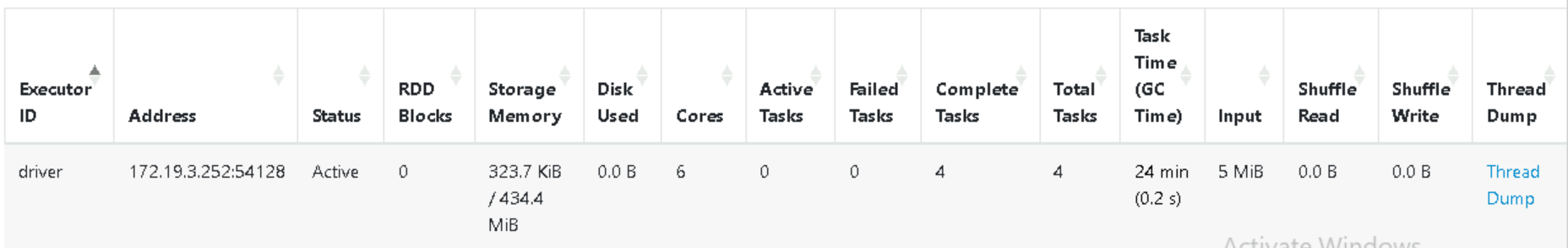
DETAILS FOR

STAGES

**A screenshot of a computer

AI-generated content may be incorrect.**

****

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**MEMORY MANAGEMENT**

Spark is an in-memory processing engine where all of the computation that a task does happens in memory. So, it is important to understand Spark Memory Management. This will help us develop Spark applications and perform performance tuning.

The Spark application includes two JVM processes: driver and executor.

The driver is the main control process, which is responsible for creating the SparkSession/SparkContext, submitting the job, converting the job to a task, and coordinating the task execution between executors.

The executor is mainly responsible for performing specific calculation tasks and returning the results to the driver.

Driver's memory management is relatively simple; Spark does not make specific plans.

2. Executor memory

The executor acts as a JVM process launched on a worker node. So, it is important to understand JVM memory management.

JVM memory management is categorized into two types:

On-Heap Memory Management (In-Heap Memory): Objects are allocated on the JVM Heap and bound by GC.

This refers to memory allocated inside the JVM heap.

Objects created in Java, such as variables, arrays, and instances of classes, are stored in heap memory.

Memory management is handled by Java's Garbage Collector (GC), which automatically frees unused memory to prevent leaks.

Since GC manages heap memory, memory allocation and deallocation happen dynamically, reducing manual overhead but sometimes leading to performance issues due to GC pauses.

Off-Heap Memory Management (External Memory): Objects are allocated in memory outside the JVM by serialization, managed by the application, and are not bound by GC.

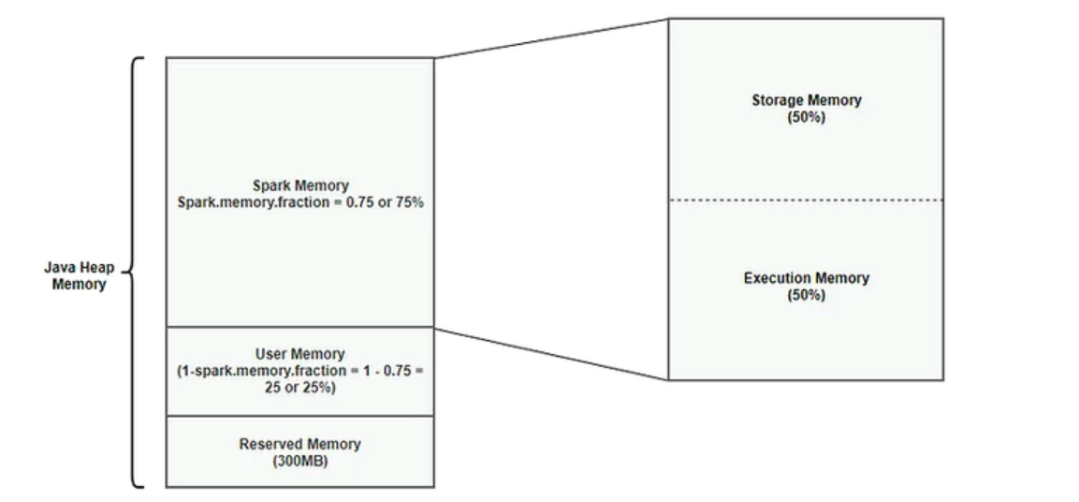
This refers to memory outside the JVM heap, often managed manually.

Objects are serialized and stored in external memory spaces (e.g., DirectByteBuffer, Native memory via JNI, or external disk).

GC does not control off-heap memory, meaning the application has explicit control over allocation and deallocation.

Used when handling large objects (e.g., caching, big data processing in Spark) to reduce GC pressure and improve performance.

**TYPES OF MEMORY IN SPARK**

****

Reserved Memory

* This is the memory reserved by the system, and its size is hardcoded.
* As of Spark 1.6.0, its value is 300MB, which means that this 300MB of RAM cannot be changed unless spark is recompiled.
* This memory stores sparks internal objects.
* Reserved Memory = 300MB
* Calculation for 4GB : Reserved Memory = 300MB

User Memory

* This is the memory area that stores all the user defined data structures, any UDFs created by the user etc,.
* This memory segment is not managed by spark, spark will not be aware of/maintain this memory segment.
* User Memory = (Java Heap — Reserved Memory) \* (1.0 — spark.memory.fraction)
* Calculation for 4GB : User Memory = (4024MB — 300MB) \* (1.0–0.75) = 949MB

Spark Memory

* This memory pool is managed by Spark.
* This is responsible for storing intermediate state while doing task execution like joins or to store the broadcast variables.
* All the cached/persisted data will be stored in this segment, specifically in the storage memory of this segment.
* Formula : (Java Heap — Reserved Memory) \* spark.memory.fraction
* Calculation for 4GB : (4096MB -300MB) \* 0.75 = 2847MB
* This is broken into 2 segments Storage Memory and Execution Memory.
* We will briefly discuss these two segments:

Storage Memory:

* Storage memory is used for storing all of the cached data, broadcast variables are also stored here.
* Storage Memory = (Java Heap — Reserved Memory) \* spark.memory.fraction \* spark.memory.storageFraction
* Storage Memory = (4096MB — 300MB) \* 0.75 \* 0.5 = ~1423MB

Execution Memory:

* This memory region is used by Spark for objects created during execution of a task. For example,it is used to store hash table for hash aggregation step,
* Execution Memory = (Java Heap — Reserved Memory) \* spark.memory.fraction \* (1.0 — spark.memory.storageFraction)
* Calculation for 4GB : Execution Memory = (4096MB — 300MB) \* 0.75 \* (1.0 — 0.5) = ~1423MB

**MEMORY MANAGEMENT IN SPARK**

**1.Unified Memory Management (Spark 1.6 and later – Default)**

* Dynamic sharing: Execution and storage memory can borrow space from each other based on demand.
* If storage (for caching RDDs) doesn’t need much memory, execution (for tasks like joins, aggregations) can use it — and vice versa.
* This improves flexibility and utilization of available memory.

**2.Static Memory Management (Before Spark 1.6)**

* + Fixed memory regions: Execution and storage memory are allocated as fixed percentages of JVM heap at the start.
  + No dynamic sharing — if execution memory is full, and storage is underused, execution still can’t use that free memory.
  + Can lead to inefficient memory usage or OutOfMemory errors during skewed workloads.