REPORT

Names : Supraja Rao
Jaya Surya Pagadi
Student ID : 1317617
1318805

Storage of Data:

In our project, we decided to use a "data lake" to store our data. This is a smart choice because a data lake has some clear benefits when compared to a regular "data warehouse".

- With a data lake, we can store data that might not be organized neatly, sort of like a jigsaw puzzle with pieces from different boxes. This is important because our project involves working with different kinds of data that don't all fit the same pattern
- Another good thing about a data lake is that it can handle a lot of information without getting too expensive

Schema Evolution:

- Data lakes enable schema-on-read, meaning that you can apply structure to the data during analysis rather than enforcing a fixed schema on ingest. This flexibility is beneficial when dealing with evolving data sources and schema changes over time.

Cost-Efficiency for Storage:

- Data lakes, like Amazon S3, offer cost-effective storage options for large volumes of data. Since the Chicago taxi fare data might grow over time, you can leverage a pay-as-you-go pricing model, storing the data without incurring significant costs.

Handling High Volume and Velocity:

- If you're dealing with large volumes of data or high data velocity (frequent updates), data lakes can handle the scale more effectively. They're designed to handle big data scenarios and can accommodate rapid growth.

Scalability and Futureproofing:

- Data lakes offer high scalability and can adapt to future data needs. As new data sources emerge and analytical requirements evolve, a data lake can provide a more scalable and adaptable solution

Connection to a Distributed Cloud Service (AWS):

We opted to connect our data lake to Amazon Web Services (AWS) due to its robust cloud infrastructure and as we already worked on it during the summer course it felt more flexible for us. AWS offers extensive services for data storage, processing, and analytics. We set up an S3 bucket to store our data, leveraging its durability and accessibility features. This integration with AWS also provides us with the flexibility to scale resources based on the project's evolving requirements.

Running Spark Application on AWS EMR:

To process and analyze the data stored in the AWS S3 data lake, we utilized Amazon EMR (Elastic MapReduce), a cloud-native big data platform. We launched a Spark cluster on EMR, leveraging its distributed processing capabilities to handle large datasets efficiently. This approach ensures parallel processing, significantly reducing computation time.

Big Data Project on Predicting Taxi Fare Price in city of Chicago using Linear Regression.

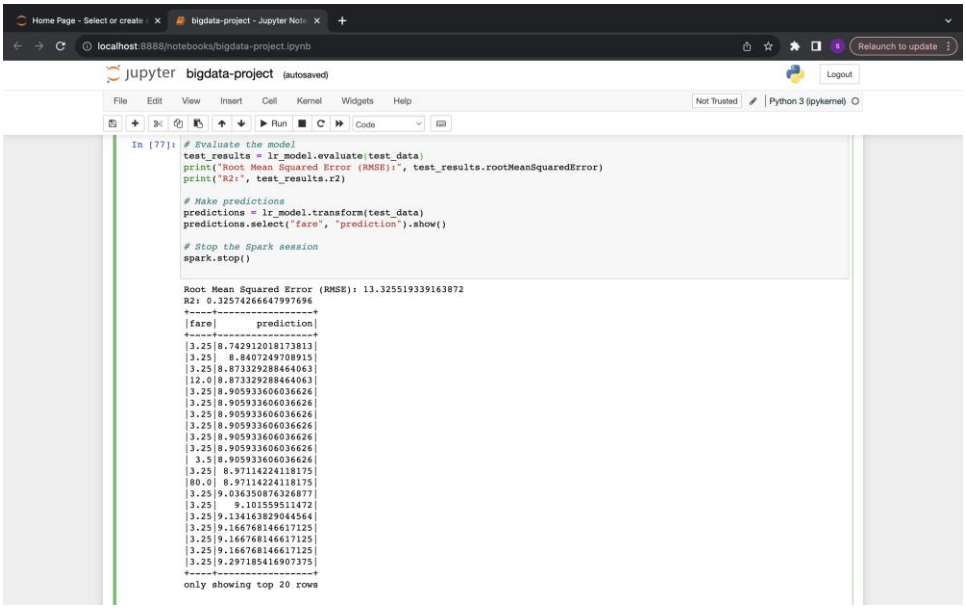
Getting the Data Get the data from https://console.cloud.google.com/bigquery?p=bigquery-public-data&d=chicago_taxi_trips&page=dataset&project=big-data-project-396823&supportedpurview=project&ws=!1m9!1m4!4m3!1sbigquery-public-data!2schicago_taxi_trips!3staxi_trips!1m3!3m2!1sbigquery-public-data!2schicago_taxi_trips

The data has the following:

- Total logical bytes: 75.75 GB
- Number of rows: 208,943,621

For the sake of this project, we extracted only 5000 rows from the data.

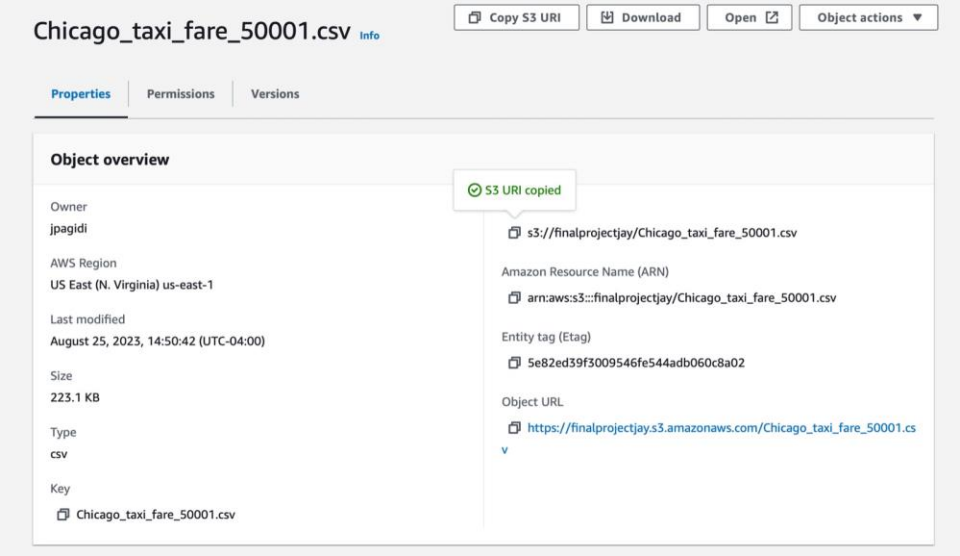
1. We cleaned the data and handled the inconsistencies it had such as null values and feature columns
2. Then proceeded with splitting the data into training and validation sets at the ratio of [0.8, 0.2]
3. Trained the data with linear regression model and my class label as fare
4. The next step we took was to evaluate the model and make the predictions



Our model was able to predict taxi fare prices based on factors like distance and time of day. This was the outcome of our project.

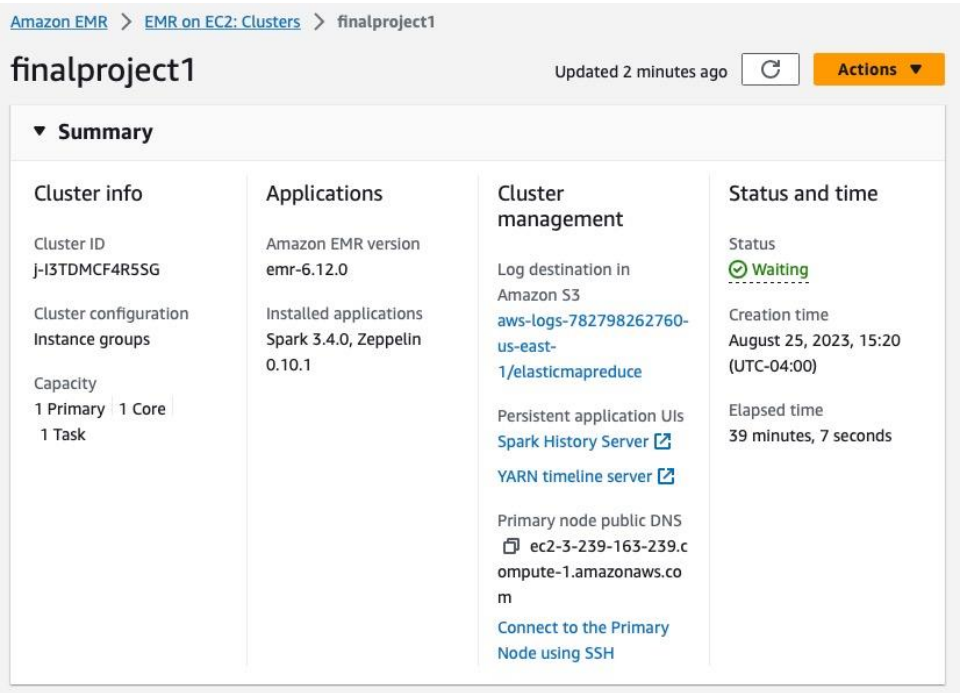
Next steps:

1. As we choose to do this in AWS, we uploaded the data to S3 bucket and here's how it looks



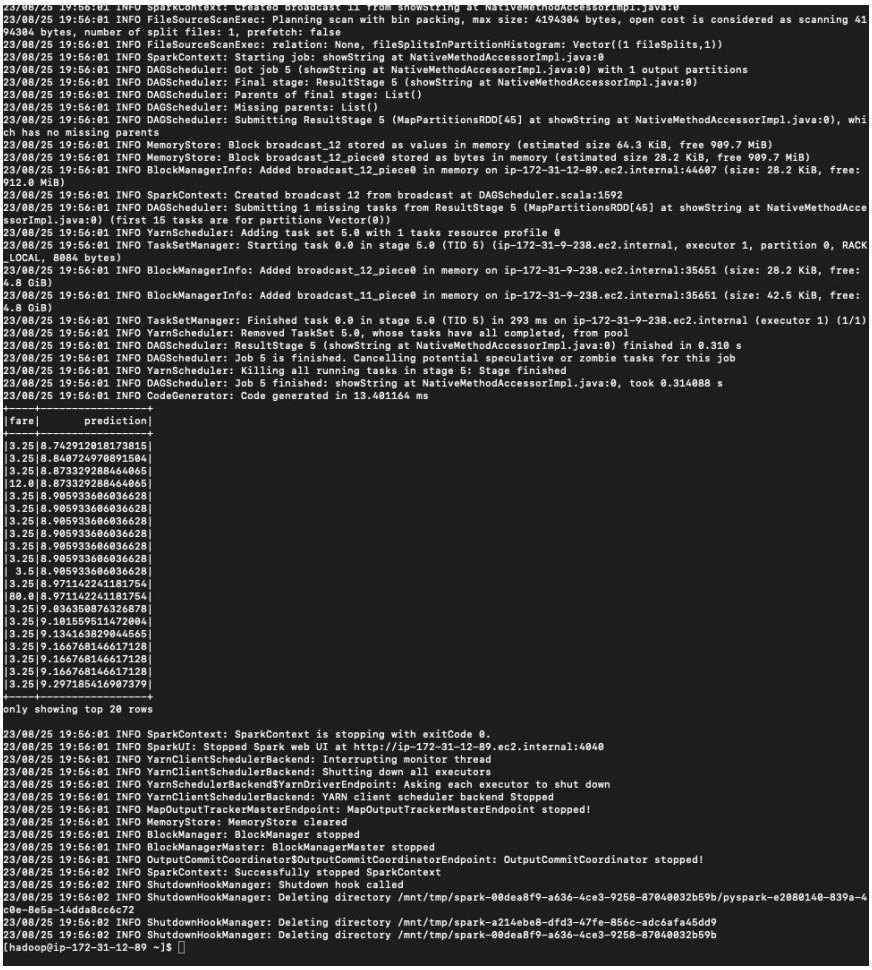
2. We used AWS EMR cluster

This Image below is our EMR cluster with its status:



3. We executed our Spark application from the command line interface, initiating a job that performed data analysis and prediction using the Linear Regression model. As the job ran, we monitored its progress and outcome through Amazon Web Services (AWS) tools.

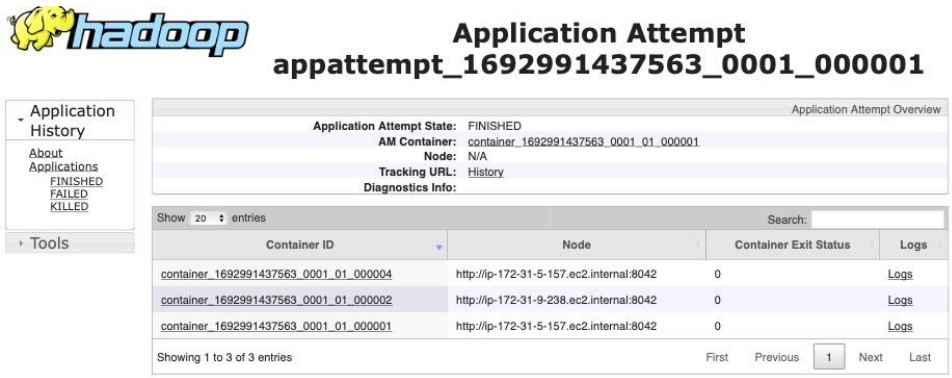
This Image shows our spark application from the terminal:



```
23/08/25 19:56:01 INFO SparkContext: Created broadcast 11 from showString at NativeMethodAccessorImpl.java:0
23/08/25 19:56:01 INFO FileSourceScanExec: Planning scan with bin packing, max size: 4194304 bytes, open cost is considered as scanning 41
94304 bytes, number of split files: 1, prefetch: false
23/08/25 19:56:01 INFO FileSourceScanExec: relation: None, fileSplitsInPartitionHistogram: Vector((1 fileSplits,1))
23/08/25 19:56:01 INFO SparkContext: Starting job: showString at NativeMethodAccessorImpl.java:0
23/08/25 19:56:01 INFO DAGScheduler: Got job 5 (showString at NativeMethodAccessorImpl.java:0) with 1 output partitions
23/08/25 19:56:01 INFO DAGScheduler: Final stage: ResultStage 5 (showString at NativeMethodAccessorImpl.java:0)
23/08/25 19:56:01 INFO DAGScheduler: Parents of final stage: List()
23/08/25 19:56:01 INFO DAGScheduler: Missing parents: List()
23/08/25 19:56:01 INFO DAGScheduler: Submitting ResultStage 5 (MapPartitionsRDD[45] at showString at NativeMethodAccessorImpl.java:0), whi
ch has no missing parents
23/08/25 19:56:01 INFO MemoryStore: Block broadcast_12 stored as values in memory (estimated size 64.3 KiB, free 909.7 MiB)
23/08/25 19:56:01 INFO MemoryStore: Block broadcast_12_piece0 stored as bytes in memory (estimated size 28.2 KiB, free 909.7 MiB)
23/08/25 19:56:01 INFO BlockManagerInfo: Added broadcast_12_piece0 in memory on ip-172-31-12-89.ec2.internal:44607 (size: 28.2 KiB, free:
912.0 MiB)
23/08/25 19:56:01 INFO SparkContext: Created broadcast 12 from broadcast at DAGScheduler.scala:1592
23/08/25 19:56:01 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 5 (MapPartitionsRDD[45] at showString at NativeMethodAcce
ssorImpl.java:0) (first 15 tasks are for partitions Vector(0))
23/08/25 19:56:01 INFO YarnScheduler: Adding task set 5.0 with 1 tasks resource profile 0
23/08/25 19:56:01 INFO TaskSetManager: Starting task 0.0 in stage 5.0 (TID 5) (ip-172-31-9-238.ec2.internal, executor 1, partition 0, RACK
_LOCAL, 8084 bytes)
23/08/25 19:56:01 INFO BlockManagerInfo: Added broadcast_12_piece0 in memory on ip-172-31-9-238.ec2.internal:35651 (size: 28.2 KiB, free:
4.8 GiB)
23/08/25 19:56:01 INFO BlockManagerInfo: Added broadcast_11_piece0 in memory on ip-172-31-9-238.ec2.internal:35651 (size: 42.5 KiB, free:
4.8 GiB)
23/08/25 19:56:01 INFO TaskSetManager: Finished task 0.0 in stage 5.0 (TID 5) in 293 ms on ip-172-31-9-238.ec2.internal (executor 1) (1/1)
23/08/25 19:56:01 INFO YarnScheduler: Removed TaskSet 5.0, whose tasks have all completed, from pool
23/08/25 19:56:01 INFO DAGScheduler: ResultStage 5 (showString at NativeMethodAccessorImpl.java:0) finished in 0.310 s
23/08/25 19:56:01 INFO DAGScheduler: Job 5 is finished. Cancelling potential speculative or zombie tasks for this job
23/08/25 19:56:01 INFO YarnScheduler: Killing all running tasks in stage 5: Stage finished
23/08/25 19:56:01 INFO DAGScheduler: Job 5 finished: showString at NativeMethodAccessorImpl.java:0, took 0.314088 s
23/08/25 19:56:01 INFO CodeGenerator: Code generated in 13.401164 ms
+----+-----------------+
|fare|       prediction|
+----+-----------------+
|3.25|8.742912018173815|
|3.25|8.840724970891504|
|3.25|8.873329288464065|
|12.0|8.873329288464065|
|3.25|8.905933606036628|
|3.25|8.905933606036628|
|3.25|8.905933606036628|
|3.25|8.905933606036628|
|3.25|8.905933606036628|
|3.25|8.905933606036628|
| 3.5|8.905933606036628|
|9.25|8.971142241181754|
|80.0|8.971142241181754|
|3.25|9.036350876326878|
|3.25|9.101559511472004|
|3.25|9.134163829044565|
|3.25|9.166768146617128|
|3.25|9.166768146617128|
|3.25|9.166768146617128|
|3.25|9.297185416907379|
+----+-----------------+
only showing top 20 rows

23/08/25 19:56:01 INFO SparkContext: SparkContext is stopping with exitCode 0.
23/08/25 19:56:01 INFO SparkUI: Stopped Spark web UI at http://ip-172-31-12-89.ec2.internal:4040
23/08/25 19:56:01 INFO YarnClientSchedulerBackend: Interrupting monitor thread
23/08/25 19:56:01 INFO YarnClientSchedulerBackend: Shutting down all executors
23/08/25 19:56:01 INFO YarnSchedulerBackend$YarnDriverEndpoint: Asking each executor to shut down
23/08/25 19:56:01 INFO YarnClientSchedulerBackend: YARN client scheduler backend Stopped
23/08/25 19:56:01 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
23/08/25 19:56:01 INFO MemoryStore: MemoryStore cleared
23/08/25 19:56:01 INFO BlockManager: BlockManager stopped
23/08/25 19:56:01 INFO BlockManagerMaster: BlockManagerMaster stopped
23/08/25 19:56:01 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
23/08/25 19:56:02 INFO SparkContext: Successfully stopped SparkContext
23/08/25 19:56:02 INFO ShutdownHookManager: Shutdown hook called
23/08/25 19:56:02 INFO ShutdownHookManager: Deleting directory /mnt/tmp/spark-00dea8f9-a636-4ce3-9258-87040032b59b/pyspark-e2080140-839a-4
c0e-8e5a-14dda8cc6c72
23/08/25 19:56:02 INFO ShutdownHookManager: Deleting directory /mnt/tmp/spark-a214ebe8-dfd3-47fe-856c-adc6afa45dd9
23/08/25 19:56:02 INFO ShutdownHookManager: Deleting directory /mnt/tmp/spark-00dea8f9-a636-4ce3-9258-87040032b59b
[hadoop@ip-172-31-12-89 ~]$
```

And this image shows the Application History of the Application :



Conclusion:

The seamless integration of our data lake with AWS showcased the power of distributed cloud services. By connecting our data lake to AWS, we harnessed the cloud's robust infrastructure and access to a wide array of tools. This integration enabled us to efficiently process and analyze our data using Amazon EMR and Apache Spark, culminating in accurate predictions through the Linear Regression model. Through a systematic and well-defined approach, we were able to achieve our goal of building a predictive model that offers valuable insights into taxi fare estimation.