PROJECT

TOPIC : IoT Sensor Data Analytics Pipeline using Kafka, AWS Glue, DMS, Kinesis, S3, ClickHouse, Hive, and PySpark.

Use-Case : Vehicle Health Monitoring & Predictive Maintenance

Problem: Unexpected vehicle breakdowns cost businesses money and lead to downtime.

Solution: Build a Glue pipeline that processes RPM, engine load, coolant temp (cTemp), and DTC codes in real-time to:

- Detect early signs of engine trouble.
- Trigger alerts for overheating, frequent high RPMs, low battery, or high eLoad.

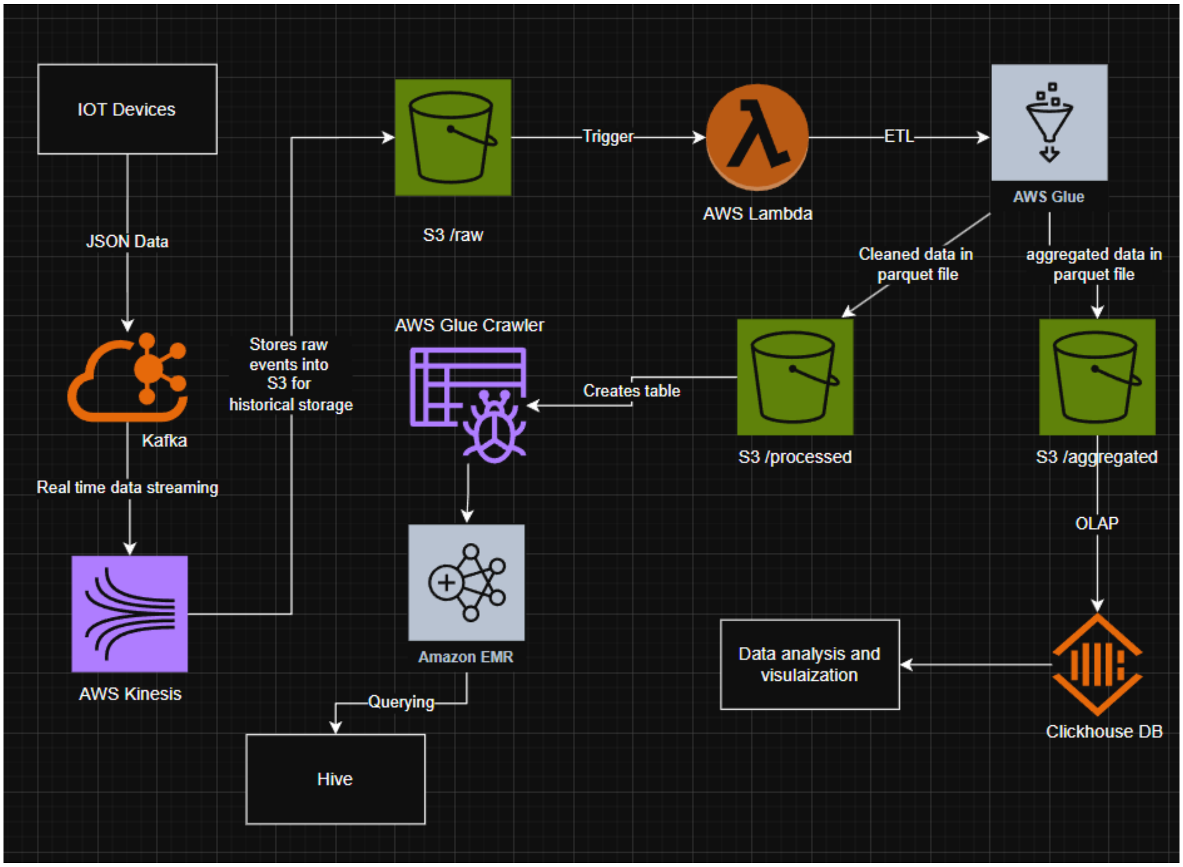
Pipeline Outcome:

- Alerting system on anomalies
- Maintenance recommendations
- Aggregated health reports in ClickHouse for dashboards

Made By,

Leon Jervis Olaprath

Architecture Diagram:



Dataset used:

This dataset contains vehicle telematics data collected in real-time from multiple passenger vehicles using IoT-enabled OBD-II and GPS modules. The data captures various engine parameters, diagnostic metrics, and driving behavior at one-second intervals, suitable for analysis of vehicle health, performance, and driver patterns.

1. Data Ingestion Layer Apache Kafka:

- Downloaded the dataset from kaggle.
- Converted the csv file into JSON using python script.

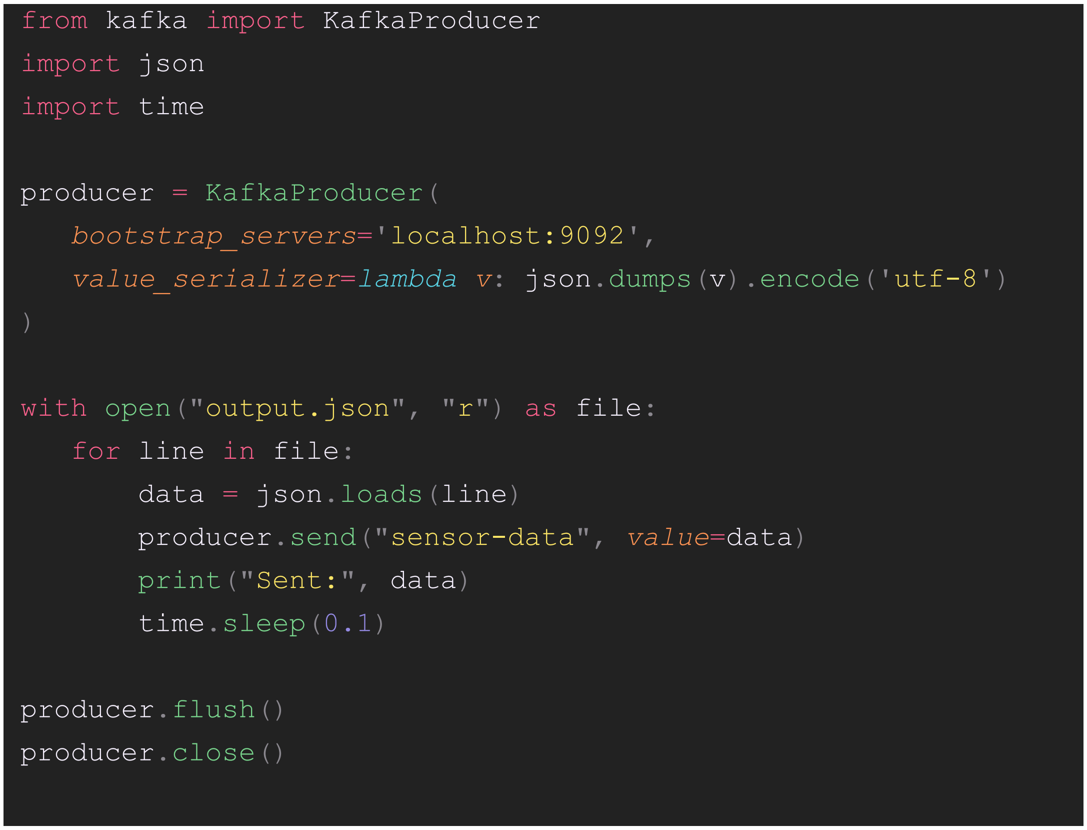
```
import csv
import json

input_csv = r"C:\Users\Leon\Downloads\BL-Project\vehicle.csv"
output_json = r"C:\Users\Leon\Downloads\BL-Project\output.json"

with open(input_csv, mode="r", encoding="utf-8") as csv_file, \
     open(output_json, mode="w", encoding="utf-8") as
json_file:

    reader = csv.DictReader(csv_file)

    for row in reader:
        json_line = json.dumps(row)
        json_file.write(json_line + "\n")

print("Success")
```

Send json file into ec2 through scp
scp -i /path/to/mykey.pem /path/to/data.json ec2-user@ec2-public-ip:/home/ec2-user/

- Downloaded and set up Kafka on EC2.
- Started zookeeper
  bin/zookeeper-server-start.sh config/zookeeper.properties
- Started server in another terminal
  bin/kafka-server-start.sh config/server.properties
  Create topic

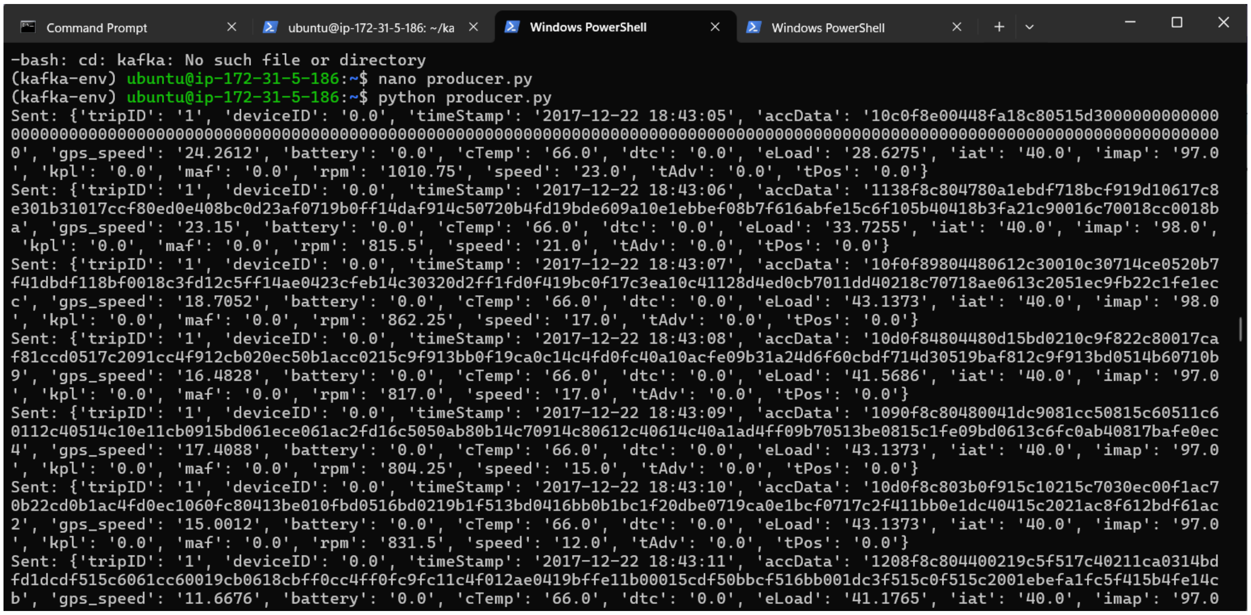
```
 bin/kafka-topics.sh \
  --create \
  --topic sensor-data \
  --bootstrap-server localhost:9092 \
  --partitions 1 \
  --replication-factor 1
```

- Created topic sensor-data.
- Producer code



```python
from kafka import KafkaProducer
import json
import time

producer = KafkaProducer(
   bootstrap_servers='localhost:9092',
   value_serializer=lambda v: json.dumps(v).encode('utf-8')
)

with open("output.json", "r") as file:
   for line in file:
       data = json.loads(line)
       producer.send("sensor-data", value=data)
       print("Sent:", data)
       time.sleep(0.1)

producer.flush()
producer.close()
```

Producer Sending the data:



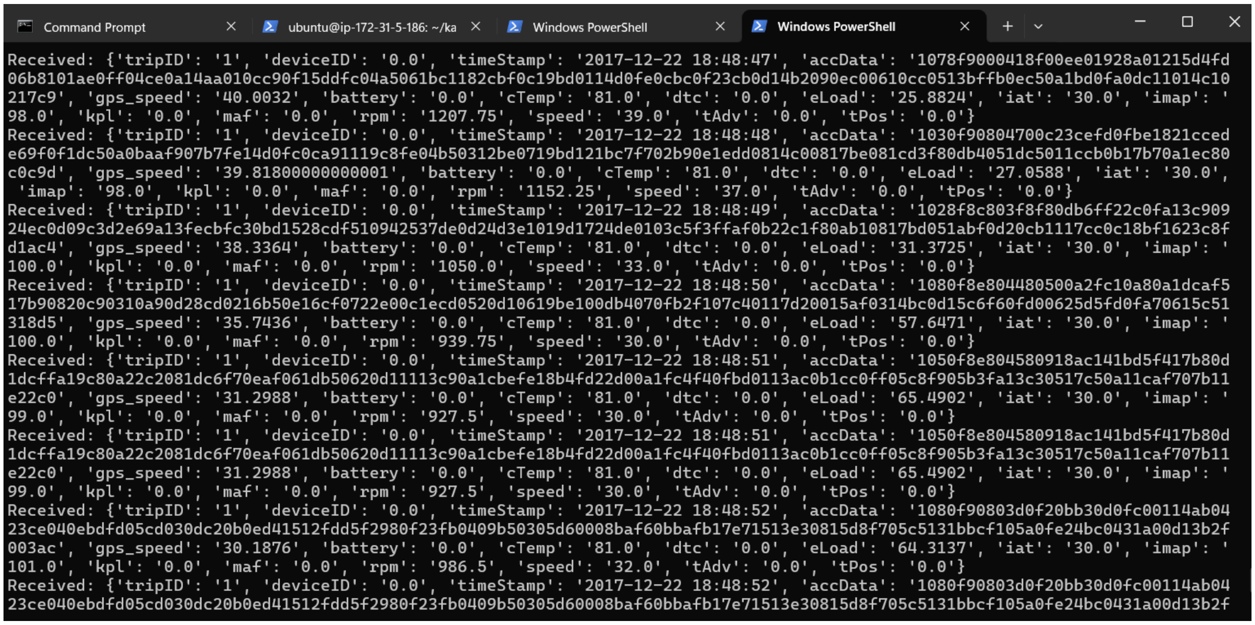
```
-bash: cd: kafka: No such file or directory
(kafka-env) ubuntu@ip-172-31-5-186:~$ nano producer.py
(kafka-env) ubuntu@ip-172-31-5-186:~$ python producer.py
Sent: {'tripID': '1', 'deviceID': '0.0', 'timeStamp': '2017-12-22 18:43:05', 'accData': '10c0f8e00448fa18c80515d3000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0', 'gps_speed': '24.2612', 'battery': '0.0', 'cTemp': '66.0', 'dtc': '0.0', 'eLoad': '28.6275', 'iat': '40.0', 'imap': '97.0
', 'kpl': '0.0', 'maf': '0.0', 'rpm': '1010.75', 'speed': '23.0', 'tAdv': '0.0', 'tPos': '0.0'}
Sent: {'tripID': '1', 'deviceID': '0.0', 'timeStamp': '2017-12-22 18:43:06', 'accData': '1138f8c804780a1ebdf718bcf919d10617c8
e301b31017ccf80ed0e408bc0d23af0719b0ff14daf914c50720b4fd19bde609a10e1ebbef08b7f616abfe15c6f105b40418b3fa21c90016c70018cc0018b
a', 'gps_speed': '23.15', 'battery': '0.0', 'cTemp': '66.0', 'dtc': '0.0', 'eLoad': '33.7255', 'iat': '40.0', 'imap': '98.0',
 'kpl': '0.0', 'maf': '0.0', 'rpm': '815.5', 'speed': '21.0', 'tAdv': '0.0', 'tPos': '0.0'}
Sent: {'tripID': '1', 'deviceID': '0.0', 'timeStamp': '2017-12-22 18:43:07', 'accData': '10f0f89804480612c30010c30714ce0520b7
f41dbdf118bf0018c3fd12c5ff14ae0423cfeb14c30320d2ff1fd0f419bc0f17c3ea10c41128d4ed0cb7011dd40218c70718ae0613c2051ec9fb22c1fe1ec
c', 'gps_speed': '18.7052', 'battery': '0.0', 'cTemp': '66.0', 'dtc': '0.0', 'eLoad': '43.1373', 'iat': '40.0', 'imap': '98.0
', 'kpl': '0.0', 'maf': '0.0', 'rpm': '862.25', 'speed': '17.0', 'tAdv': '0.0', 'tPos': '0.0'}
Sent: {'tripID': '1', 'deviceID': '0.0', 'timeStamp': '2017-12-22 18:43:08', 'accData': '10d0f84804480d15bd0210c9f822c80017ca
f81ccd0517c2091cc4f912cb020ec50b1acc0215c9f913bb0f19ca0c14c4fd0fc40a10acfe09b31a24d6f60cbdf714d30519baf812c9f913bd0514b60710b
9', 'gps_speed': '16.4828', 'battery': '0.0', 'cTemp': '66.0', 'dtc': '0.0', 'eLoad': '41.5686', 'iat': '40.0', 'imap': '97.0
', 'kpl': '0.0', 'maf': '0.0', 'rpm': '817.0', 'speed': '17.0', 'tAdv': '0.0', 'tPos': '0.0'}
Sent: {'tripID': '1', 'deviceID': '0.0', 'timeStamp': '2017-12-22 18:43:09', 'accData': '1090f8c80480041dc9081cc50815c60511c6
0112c40514c10e11cb0915bd061ece061ac2fd16c5050ab80b14c70914c80612c40614c40a1ad4ff09b70513be0815c1fe09bd0613c6fc0ab40817bafe0ec
4', 'gps_speed': '17.4088', 'battery': '0.0', 'cTemp': '66.0', 'dtc': '0.0', 'eLoad': '43.1373', 'iat': '40.0', 'imap': '97.0
', 'kpl': '0.0', 'maf': '0.0', 'rpm': '804.25', 'speed': '15.0', 'tAdv': '0.0', 'tPos': '0.0'}
Sent: {'tripID': '1', 'deviceID': '0.0', 'timeStamp': '2017-12-22 18:43:10', 'accData': '10d0f8c803b0f915c10215c7030ec00f1ac7
0b22cd0b1ac4fd0ec1060fc80413be010fbd0516bd0219b1f513bd0416bb0b1bc1f20dbe0719ca0e1bcf0717c2f411bb0e1dc40415c2021ac8f612bdf61ac
2', 'gps_speed': '15.0012', 'battery': '0.0', 'cTemp': '66.0', 'dtc': '0.0', 'eLoad': '43.1373', 'iat': '40.0', 'imap': '97.0
', 'kpl': '0.0', 'maf': '0.0', 'rpm': '831.5', 'speed': '12.0', 'tAdv': '0.0', 'tPos': '0.0'}
Sent: {'tripID': '1', 'deviceID': '0.0', 'timeStamp': '2017-12-22 18:43:11', 'accData': '1208f8c804400219c5f517c40211ca0314bd
fd1dcdf515c6061cc60019cb0618cbff0cc4ff0fc9fc11c4f012ae0419bffe11b00015cdf50bbcf516bb001dc3f515c0f515c2001ebefa1fc5f415b4fe14c
b', 'gps_speed': '11.6676', 'battery': '0.0', 'cTemp': '66.0', 'dtc': '0.0', 'eLoad': '41.1765', 'iat': '40.0', 'imap': '97.0
```

Testing if it is able to be received through consumer:

Consumer code:

```
from kafka import KafkaConsumer
import json

consumer = KafkaConsumer(
    'sensor-data',
    bootstrap_servers='localhost:9092',
    auto_offset_reset='earliest',
    group_id='my-group',
    value_deserializer=lambda m: json.loads(m.decode('utf-8'))
)

print("Listening to messages...")
for message in consumer:
    print("Received:", message.value)
```

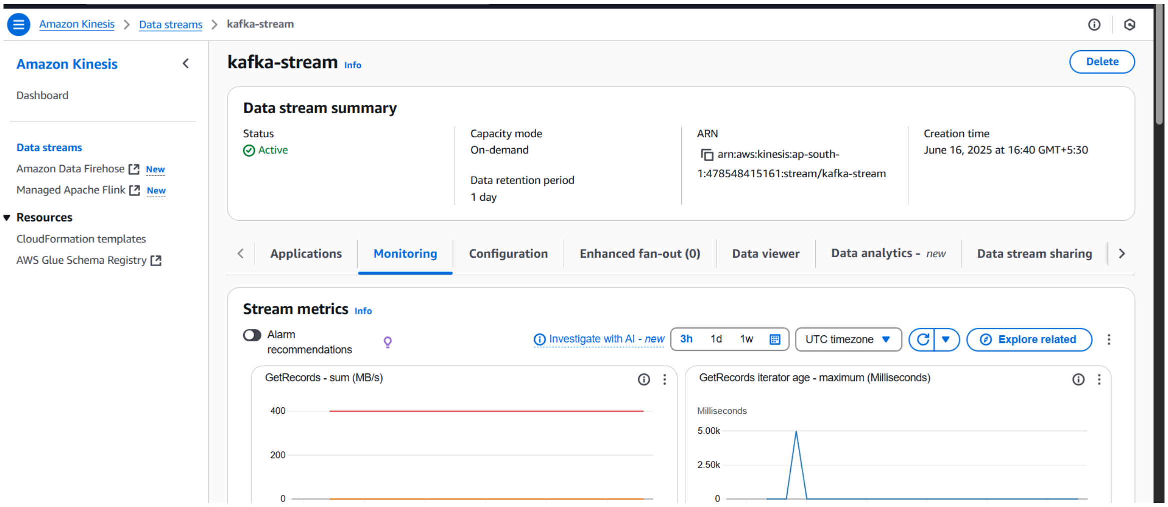
Output:



```
Received: {'tripID': '1', 'deviceID': '0.0', 'timeStamp': '2017-12-22 18:48:47', 'accData': '1078f9000418f00ee01928a01215d4fd
06b8101ae0ff04ce0a14aa010cc90f15ddfc04a5061bc1182cbf0c19bd0114d0fe0cbc0f23cb0d14b2090ec00610cc0513bffb0ec50a1bd0fa0dc11014c10
217c9', 'gps_speed': '40.0032', 'battery': '0.0', 'cTemp': '81.0', 'dtc': '0.0', 'eLoad': '25.8824', 'iat': '30.0', 'imap': '
98.0', 'kpl': '0.0', 'maf': '0.0', 'rpm': '1207.75', 'speed': '39.0', 'tAdv': '0.0', 'tPos': '0.0'}
Received: {'tripID': '1', 'deviceID': '0.0', 'timeStamp': '2017-12-22 18:48:48', 'accData': '1030f90804700c23cefd0fbe1821cced
e69f0f1dc50a0baaf907b7fe14d0fc0ca91119c8fe04b50312be0719bd121bc7f702b90e1edd0814c00817be081cd3f80db4051dc5011ccb0b17b70a1ec80
c0c9d', 'gps_speed': '39.81800000000001', 'battery': '0.0', 'cTemp': '81.0', 'dtc': '0.0', 'eLoad': '27.0588', 'iat': '30.0',
 'imap': '98.0', 'kpl': '0.0', 'maf': '0.0', 'rpm': '1152.25', 'speed': '37.0', 'tAdv': '0.0', 'tPos': '0.0'}
Received: {'tripID': '1', 'deviceID': '0.0', 'timeStamp': '2017-12-22 18:48:49', 'accData': '1028f8c803f8f80db6ff22c0fa13c909
24ec0d09c3d2e69a13fecbfc30bd1528cdf510942537de0d24d3e1019d1724de0103c5f3ffaf0b22c1f80ab10817bd051abf0d20cb1117cc0c18bf1623c8f
d1ac4', 'gps_speed': '38.3364', 'battery': '0.0', 'cTemp': '81.0', 'dtc': '0.0', 'eLoad': '31.3725', 'iat': '30.0', 'imap': '
100.0', 'kpl': '0.0', 'maf': '0.0', 'rpm': '1050.0', 'speed': '33.0', 'tAdv': '0.0', 'tPos': '0.0'}
Received: {'tripID': '1', 'deviceID': '0.0', 'timeStamp': '2017-12-22 18:48:50', 'accData': '1080f8e804480500a2fc10a80a1dcaf5
17b90820c90310a90d28cd0216b50e16cf0722e00c1ecd0520d10619be100db4070fb2f107c40117d20015af0314bc0d15c6f60fd00625d5fd0fa70615c51
318d5', 'gps_speed': '35.7436', 'battery': '0.0', 'cTemp': '81.0', 'dtc': '0.0', 'eLoad': '57.6471', 'iat': '30.0', 'imap': '
100.0', 'kpl': '0.0', 'maf': '0.0', 'rpm': '939.75', 'speed': '30.0', 'tAdv': '0.0', 'tPos': '0.0'}
Received: {'tripID': '1', 'deviceID': '0.0', 'timeStamp': '2017-12-22 18:48:51', 'accData': '1050f8e804580918ac141bd5f417b80d
1dcffa19c80a22c2081dc6f70eaf061db50620d11113c90a1cbefe18b4fd22d00a1fc4f40fbd0113ac0b1cc0ff05c8f905b3fa13c30517c50a11caf707b11
e22c0', 'gps_speed': '31.2988', 'battery': '0.0', 'cTemp': '81.0', 'dtc': '0.0', 'eLoad': '65.4902', 'iat': '30.0', 'imap': '
99.0', 'kpl': '0.0', 'maf': '0.0', 'rpm': '927.5', 'speed': '30.0', 'tAdv': '0.0', 'tPos': '0.0'}
Received: {'tripID': '1', 'deviceID': '0.0', 'timeStamp': '2017-12-22 18:48:51', 'accData': '1050f8e804580918ac141bd5f417b80d
1dcffa19c80a22c2081dc6f70eaf061db50620d11113c90a1cbefe18b4fd22d00a1fc4f40fbd0113ac0b1cc0ff05c8f905b3fa13c30517c50a11caf707b11
e22c0', 'gps_speed': '31.2988', 'battery': '0.0', 'cTemp': '81.0', 'dtc': '0.0', 'eLoad': '65.4902', 'iat': '30.0', 'imap': '
99.0', 'kpl': '0.0', 'maf': '0.0', 'rpm': '927.5', 'speed': '30.0', 'tAdv': '0.0', 'tPos': '0.0'}
Received: {'tripID': '1', 'deviceID': '0.0', 'timeStamp': '2017-12-22 18:48:52', 'accData': '1080f90803d0f20bb30d0fc00114ab04
23ce040ebdfd05cd030dc20b0ed41512fdd5f2980f23fb0409b50305d60008baf60bbafb17e71513e30815d8f705c5131bbcf105a0fe24bc0431a00d13b2f
003ac', 'gps_speed': '30.1876', 'battery': '0.0', 'cTemp': '81.0', 'dtc': '0.0', 'eLoad': '64.3137', 'iat': '30.0', 'imap': '
101.0', 'kpl': '0.0', 'maf': '0.0', 'rpm': '986.5', 'speed': '32.0', 'tAdv': '0.0', 'tPos': '0.0'}
Received: {'tripID': '1', 'deviceID': '0.0', 'timeStamp': '2017-12-22 18:48:52', 'accData': '1080f90803d0f20bb30d0fc00114ab04
23ce040ebdfd05cd030dc20b0ed41512fdd5f2980f23fb0409b50305d60008baf60bbafb17e71513e30815d8f705c5131bbcf105a0fe24bc0431a00d13b2f
```

2. Real-time Streaming with Kinesis Amazon Kinesis Data Streams :

- Go to AWS Kinesis and create a kinesis stream called kafka-stream



kafka-stream Info

Data stream summary

Status: Active

Capacity mode: On-demand

Data retention period: 1 day

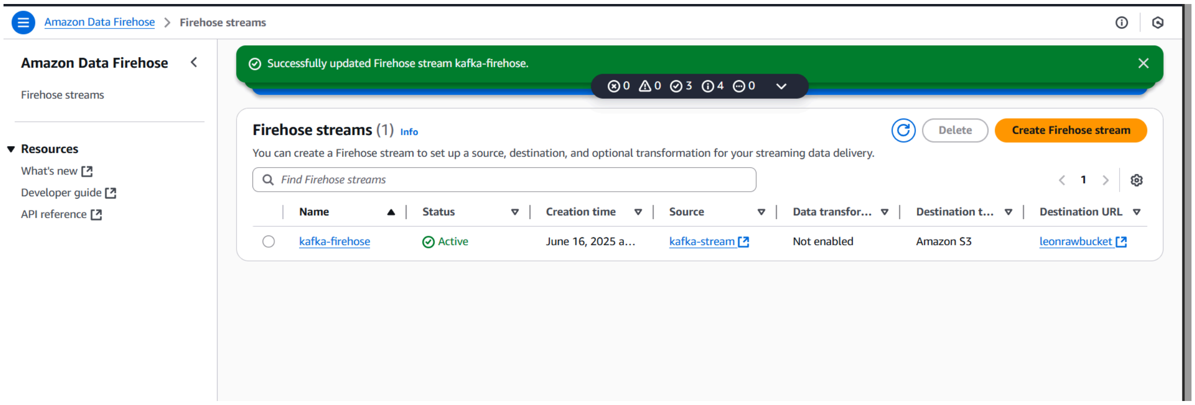
ARN: arn:aws:kinesis:ap-south-1:478548415161:stream/kafka-stream

Creation time: June 16, 2025 at 16:40 GMT+5:30

Stream metrics Info

- Create a kafka firehose with source as kafka-stream and destination as S3 bucket

- Firehose is used to take data from kinesis and store in in s3



Successfully updated Firehose stream kafka-firehose.

Firehose streams (1) Info

You can create a Firehose stream to set up a source, destination, and optional transformation for your streaming data delivery.

	Name	Status	Creation time	Source	Data transfor...	Destination t...	Destination URL
○	kafka-firehose	✓ Active	June 16, 2025 a...	kafka-stream	Not enabled	Amazon S3	leonrawbucket

- Type aws configure and enter all necessary details in ec2 where kafka is running.

- Clone the connector plugin

```
git clone https://github.com/awslabs/kinesis-kafka-connector.git
cd kinesis-kafka-connector
./gradlew shadowJar
cp target/kinesis-kafka-connector.jar ~/kafka_2.13-3.7.0/libs/
```

- In kinesis-sink-properties add

```
name=kinesis-sink-connector
connector.class=io.lenses.streamreactor.connect.kinesis.sink.KinesisSinkConnector
tasks.max=1
topics=sensor-data
aws.kinesis.stream=kafka-stream
aws.region=ap-south-1
key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=org.apache.kafka.connect.storage.StringConverter
aws.access.key.id=ACCESS_KEY
aws.secret.access.key=SECRET_KEY
```

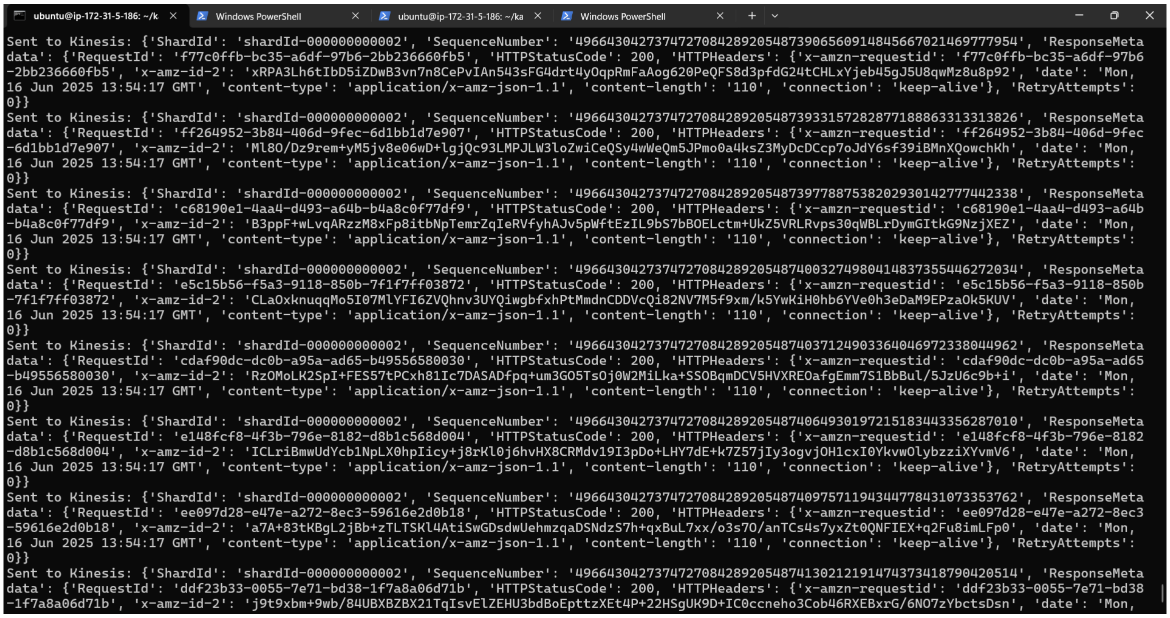
- Start the connect worker

```
bin/connect-standalone.sh config/connect-standalone.properties kinesis-sink.properties
```

- Update consumer code to send data to kinesis stream

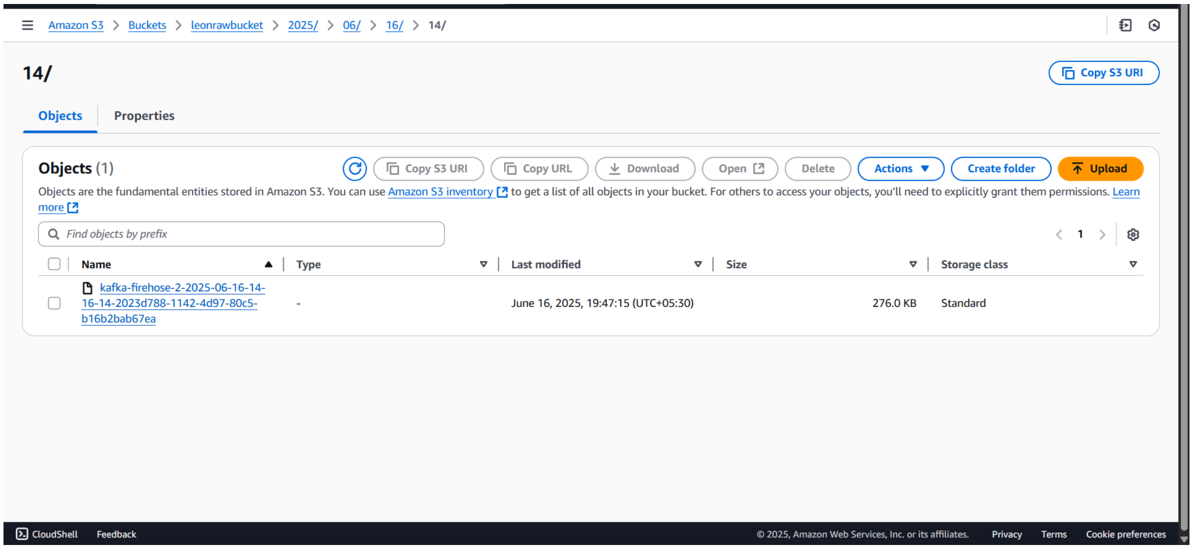
```
from kafka import KafkaConsumer
import boto3
import json

aws_region = 'ap-south-1'
kinesis_stream_name = 'kafka-stream'

kinesis_client = boto3.client('kinesis', region_name=aws_region)

consumer = KafkaConsumer(
   'sensor-data',
   bootstrap_servers=['localhost:9092'],
   auto_offset_reset='latest',
   enable_auto_commit=True,
   group_id='my-group',
   value_deserializer=lambda m: json.loads(m.decode('utf-8'))
)

for message in consumer:
   data = message.value

   payload = json.dumps(data).encode('utf-8')
   response = kinesis_client.put_record(
       StreamName=kinesis_stream_name,
       Data=payload,
       PartitionKey="partitionKey"
   )
   print(f"Sent to Kinesis: {response}")
```

- Run the zookeeper and then the producer as well as the consumer.



```
Sent to Kinesis: {'ShardId': 'shardId-000000000002', 'SequenceNumber': '49664304273747270842892054873906560914845667021469777954', 'ResponseMeta
data': {'RequestId': 'f77c0ffb-bc35-a6df-97b6-2bb236660fb5', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': 'f77c0ffb-bc35-a6df-97b6
-2bb236660fb5', 'x-amz-id-2': 'xRPA3Lh6tIbD5iZDwB3vn7n8CePvIAn543sFG4drt4yOqpRmFaAog620PeQFS8d3pfdG24tCHLxYjeb45gJ5U8qwMz8u8p92', 'date': 'Mon,
16 Jun 2025 13:54:17 GMT', 'content-type': 'application/x-amz-json-1.1', 'content-length': '110', 'connection': 'keep-alive'}, 'RetryAttempts':
0}}
Sent to Kinesis: {'ShardId': 'shardId-000000000002', 'SequenceNumber': '49664304273747270842892054873933157282877188863313313826', 'ResponseMeta
data': {'RequestId': 'ff264952-3b84-406d-9fec-6d1bb1d7e907', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': 'ff264952-3b84-406d-9fec
-6d1bb1d7e907', 'x-amz-id-2': 'Ml8O/Dz9rem+yM5jv8e06wD+lgjQc93LMPJLW3loZwiCeQSy4wWeQm5JPmo0a4ksZ3MyDcDCcp7oJdY6sf39iBMnXQowchKh', 'date': 'Mon,
16 Jun 2025 13:54:17 GMT', 'content-type': 'application/x-amz-json-1.1', 'content-length': '110', 'connection': 'keep-alive'}, 'RetryAttempts':
0}}
Sent to Kinesis: {'ShardId': 'shardId-000000000002', 'SequenceNumber': '49664304273747270842892054873977887538202930142777442338', 'ResponseMeta
data': {'RequestId': 'c68190e1-4aa4-d493-a64b-b4a8c0f77df9', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': 'c68190e1-4aa4-d493-a64b
-b4a8c0f77df9', 'x-amz-id-2': 'B3ppF+wLvqARzzM8xFp8itbNpTemrZqIeRVfyhAJv5pWftEzIL9bS7bBOELctm+UkZ5VRLRvps30qWBLrDymGItkG9NzjXEZ', 'date': 'Mon,
16 Jun 2025 13:54:17 GMT', 'content-type': 'application/x-amz-json-1.1', 'content-length': '110', 'connection': 'keep-alive'}, 'RetryAttempts':
0}}
Sent to Kinesis: {'ShardId': 'shardId-000000000002', 'SequenceNumber': '49664304273747270842892054874003274980414837355446272034', 'ResponseMeta
data': {'RequestId': 'e5c15b56-f5a3-9118-850b-7f1f7ff03872', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': 'e5c15b56-f5a3-9118-850b
-7f1f7ff03872', 'x-amz-id-2': 'CLaOxknuqqMo5I07MlYFI6ZVQhnv3UYQiwgbfxhPtMmdnCDDVcQi82NV7M5f9xm/k5YwKiH0hb6YVe0h3eDaM9EPzaOk5KUV', 'date': 'Mon,
16 Jun 2025 13:54:17 GMT', 'content-type': 'application/x-amz-json-1.1', 'content-length': '110', 'connection': 'keep-alive'}, 'RetryAttempts':
0}}
Sent to Kinesis: {'ShardId': 'shardId-000000000002', 'SequenceNumber': '49664304273747270842892054874037124903364046972338044962', 'ResponseMeta
data': {'RequestId': 'cdaf90dc-dc0b-a95a-ad65-b49556580030', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': 'cdaf90dc-dc0b-a95a-ad65
-b49556580030', 'x-amz-id-2': 'RzOMoLK2SpI+FES57tPCxh81Ic7DASADfpq+um3GO5TsOj0W2MiLka+SSOBqmDCV5HVXREOafgEmm7S1BbBul/5JzU6c9b+i', 'date': 'Mon,
16 Jun 2025 13:54:17 GMT', 'content-type': 'application/x-amz-json-1.1', 'content-length': '110', 'connection': 'keep-alive'}, 'RetryAttempts':
0}}
Sent to Kinesis: {'ShardId': 'shardId-000000000002', 'SequenceNumber': '49664304273747270842892054874064930197215183443356287010', 'ResponseMeta
data': {'RequestId': 'e148fcf8-4f3b-796e-8182-d8b1c568d004', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': 'e148fcf8-4f3b-796e-8182
-d8b1c568d004', 'x-amz-id-2': 'ICLriBmwUdYcb1NpLX0hpIicy+j8rKl0j6hvHX8CRMdv19I3pDo+LHY7dE+k7Z57jIy3ogvjOH1cxI0YkvwOlybzziXYvmV6', 'date': 'Mon,
16 Jun 2025 13:54:17 GMT', 'content-type': 'application/x-amz-json-1.1', 'content-length': '110', 'connection': 'keep-alive'}, 'RetryAttempts':
0}}
Sent to Kinesis: {'ShardId': 'shardId-000000000002', 'SequenceNumber': '49664304273747270842892054874097571194344778431073353762', 'ResponseMeta
data': {'RequestId': 'ee097d28-e47e-a272-8ec3-59616e2d0b18', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': 'ee097d28-e47e-a272-8ec3
-59616e2d0b18', 'x-amz-id-2': 'a7A+83tKBgL2jBb+zTLTSKl4AtiSwGDsdwUehmzqaDSNdzS7h+qxBuL7xx/o3s7O/anTCs4s7yxZt0QNFIEX+q2Fu8imLFp0', 'date': 'Mon,
16 Jun 2025 13:54:17 GMT', 'content-type': 'application/x-amz-json-1.1', 'content-length': '110', 'connection': 'keep-alive'}, 'RetryAttempts':
0}}
Sent to Kinesis: {'ShardId': 'shardId-000000000002', 'SequenceNumber': '49664304273747270842892054874130212191474373418790420514', 'ResponseMeta
data': {'RequestId': 'ddf23b33-0055-7e71-bd38-1f7a8a06d71b', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': 'ddf23b33-0055-7e71-bd38
-1f7a8a06d71b', 'x-amz-id-2': 'j9t9xbm+9wb/84UBXBZBX21TqIsvElZEHU3bdBoEpttzXEt4P+22HSgUK9D+IC0ccneho3Cob46RXEBxrG/6NO7zYbctsDsn', 'date': 'Mon,
```

- Check S3 bucket if the files are present.

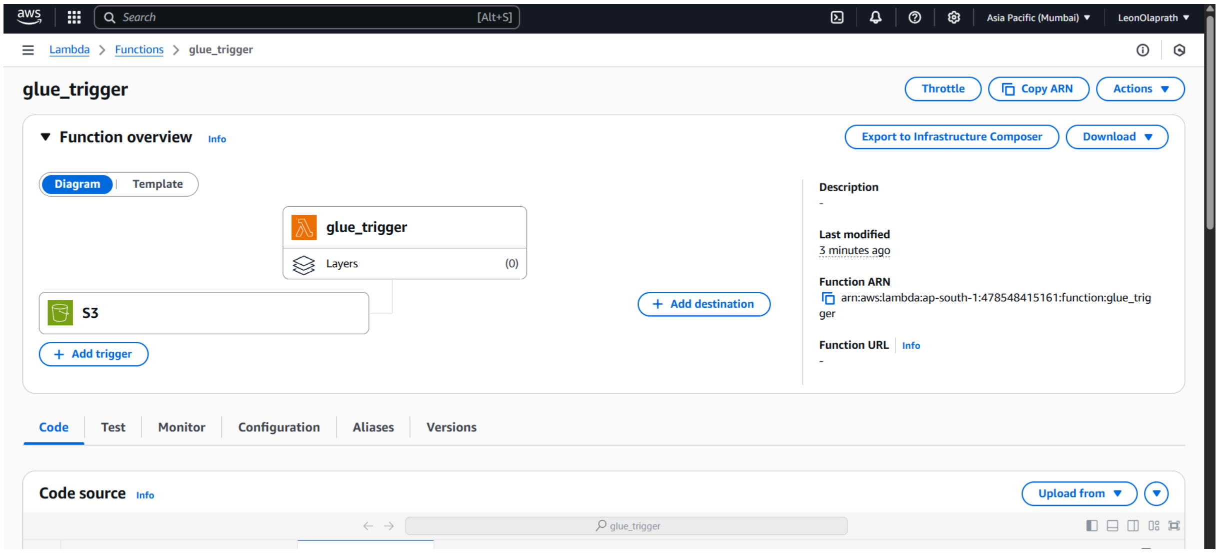


Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

□	Name	Type	Last modified	Size	Storage class
□	kafka-firehose-2-2025-06-16-14-16-14-2023d788-1142-4d97-80c5-b16b2bab67ea	-	June 16, 2025, 19:47:15 (UTC+05:30)	276.0 KB	Standard

3. Processing with AWS Glue + PySpark AWS Glue Jobs

Create a lambda function to send the files into glue automatically immediately after it enters the s3 bucket from kinesis.



Description
-

Last modified
3 minutes ago

Function ARN
arn:aws:lambda:ap-south-1:478548415161:function:glue_trigger

Function URL | Info
-

In the lambda IAM role include

```
{
  "Effect": "Allow",
  "Action": [
    "glue:StartJobRun"
  ],
  "Resource": "arn:aws:glue:region:account-id:job/glue-job-name"
}
```

```
import boto3
import urllib.parse

glue = boto3.client('glue')

def lambda_handler(event, context):
    bucket = event['Records'][0]['s3']['bucket']['name']
    key =
urllib.parse.unquote_plus(event['Records'][0]['s3']['object']['key'])

    print(f"New file in S3: s3://{bucket}/{key}")
    response = glue.start_job_run(
        JobName='Vehicle-stream',
        Arguments={
            '--source_s3_path': f's3://{bucket}/{key}'
        }
    )

    print(f"Glue job started: JobRunId = {response['JobRunId']}")
    return {
        'statusCode': 200,
        'body': f"Triggered Glue job for s3://{bucket}/{key}"
    }
```

Setup AWS- Glue:

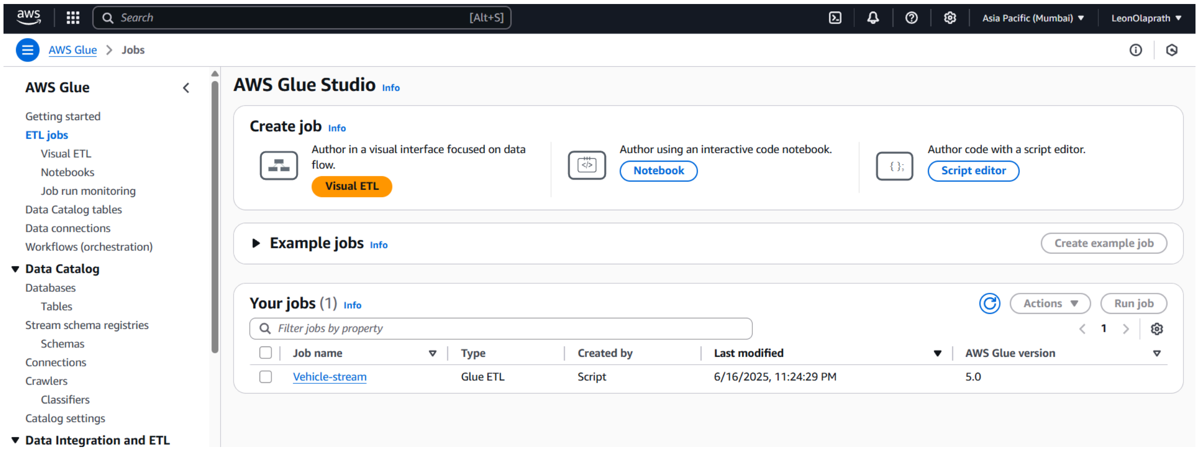
- Go to ETL Jobs inside Glue
- Select Python and then choose Spark
- Then in script write the pyspark script to take data from s3 and then clean and send the processed data as parquet file back to s3 in processed and aggregated forms

```
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from pyspark.sql.functions import (
    col, avg, max, count, window, to_timestamp
)
from pyspark.sql.types import StructType, StringType, DoubleType,
IntegerType

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session

schema = StructType() \
    .add("tripID", StringType()) \
    .add("deviceID", StringType()) \
    .add("timeStamp", StringType()) \
    .add("accData", StringType()) \
    .add("gps_speed", DoubleType()) \
    .add("battery", DoubleType()) \
    .add("cTemp", DoubleType()) \
    .add("dtc", IntegerType()) \
    .add("eLoad", DoubleType()) \
    .add("iat", IntegerType()) \
    .add("imap", IntegerType()) \
    .add("kpl", DoubleType()) \
    .add("maf", DoubleType()) \
    .add("rpm", DoubleType()) \
    .add("speed", DoubleType()) \
    .add("tAdv", IntegerType()) \
    .add("tPos", IntegerType())


df = spark.readStream \
    .schema(schema) \
    .json("s3://leonrawbucket/2025/06/16/14/")

df_cleaned = df.fillna({
    "tripID": "unknown",
    "deviceID": "unknown",
    "timeStamp": "1970-01-01T00:00:00",
    "accData": "unknown",
    "gps_speed": 0.0,
    "battery": 0.0,
    "cTemp": 0.0,
    "dtc": 0,
    "eLoad": 0.0,
    "iat": 0,
    "imap": 0,
    "kpl": 0.0,
    "maf": 0.0,
    "rpm": 0.0,
    "speed": 0.0,
    "tAdv": 0,
    "tPos": 0
}).withColumn("event_time", to_timestamp("timeStamp",
"yyyy-MM-dd'T'HH:mm:ss"))

df_cleaned.writeStream \
    .format("parquet") \
    .option("checkpointLocation",
"s3://leonprocessed/vehicleprocessed/_checkpoints/") \
    .option("path", "s3://leonprocessed/vehicleprocessed/") \
    .outputMode("append") \
    .start()

df_aggregated = df_cleaned.groupBy("deviceID").agg(
    avg("rpm").alias("avg_rpm"),
    avg("speed").alias("avg_speed"),
    avg("gps_speed").alias("avg_gps_speed"),
    avg("battery").alias("avg_battery"),
    avg("cTemp").alias("avg_coolant_temp"),
    avg("eLoad").alias("avg_engine_load"),
    avg("iat").alias("avg_intake_air_temp"),
    avg("imap").alias("avg_intake_pressure"),
    avg("kpl").alias("avg_kmpl"),
    avg("maf").alias("avg_mass_air_flow"),
    max("speed").alias("max_speed"),
    count("*").alias("record_count")
)

df_aggregated.writeStream \
    .format("parquet") \
    .option("checkpointLocation",
"s3://leonaggregated/vehiclehealth/_checkpoints/") \
    .option("path", "s3://leonaggregated/vehiclehealth/") \
    .outputMode("append") \
    .start() \
    .awaitTermination()
```

The processing done by this Glue script is :

- Set up the Spark and Glue runtime environment.
- Defines schema for incoming JSON files.

- Reads streaming JSON telemetry data from the raw S3 path.
- Defines schema and fills missing rows with values
- Converts timestamp to timestamp type
- Sends cleaned data to s3
- Performs aggregation on the data and sends to s3



AWS Glue Studio Info

Create job Info

Author in a visual interface focused on data flow. Visual ETL

Author using an interactive code notebook. Notebook

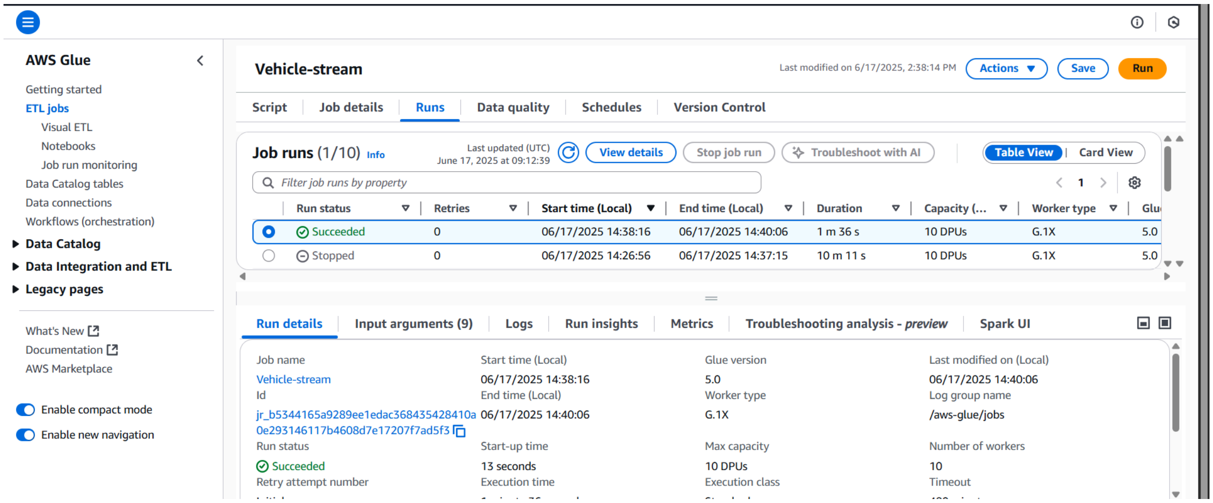
Author code with a script editor. Script editor

▶ Example jobs Info — Create example job

Your jobs (1) Info — Actions ▼ — Run job

Filter jobs by property

□	Job name	Type	Created by	Last modified	AWS Glue version
□	Vehicle-stream	Glue ETL	Script	6/16/2025, 11:24:29 PM	5.0



Vehicle-stream — Last modified on 6/17/2025, 2:38:14 PM — Actions ▼ — Save — Run

Script | Job details | Runs | Data quality | Schedules | Version Control

Job runs (1/10) Info — Last updated (UTC) June 17, 2025 at 09:12:39 — View details — Stop job run — Troubleshoot with AI — Table View | Card View

Filter job runs by property

	Run status	Retries	Start time (Local)	End time (Local)	Duration	Capacity (...	Worker type	Glu
●	✓ Succeeded	0	06/17/2025 14:38:16	06/17/2025 14:40:06	1 m 36 s	10 DPUs	G.1X	5.0
○	Stopped	0	06/17/2025 14:26:56	06/17/2025 14:37:15	10 m 11 s	10 DPUs	G.1X	5.0

Run details | Input arguments (9) | Logs | Run insights | Metrics | Troubleshooting analysis - *preview* | Spark UI

Job name
Vehicle-stream

Id
jr_b5344165a9289ee1edac368435428410a0e293146117b4608d7e17207f7ad5f3

Run status
✓ Succeeded

Retry attempt number

Start time (Local)
06/17/2025 14:38:16

End time (Local)
06/17/2025 14:40:06

Start-up time
13 seconds

Execution time

Glue version
5.0

Worker type
G.1X

Max capacity
10 DPUs

Execution class

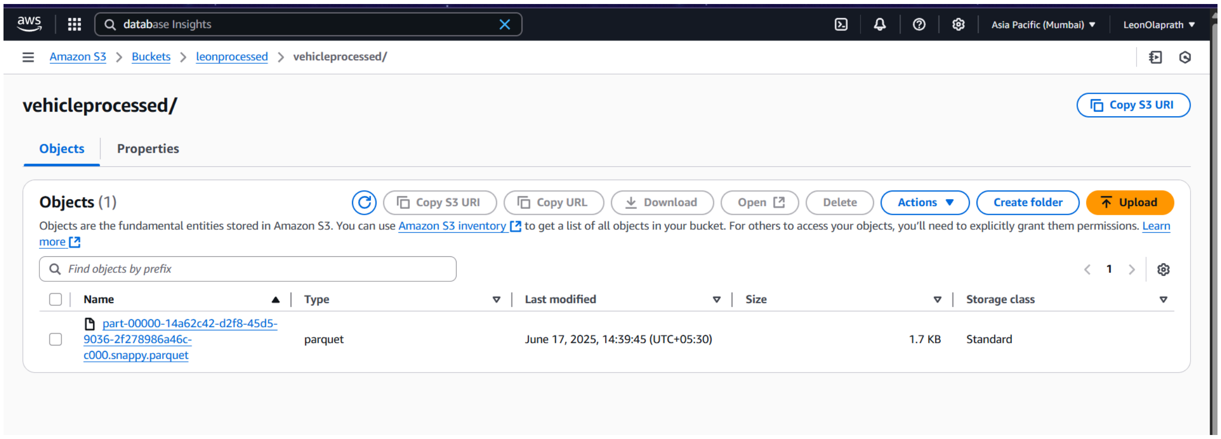
Last modified on (Local)
06/17/2025 14:40:06

Log group name
/aws-glue/jobs

Number of workers
10

Timeout

Cleaned Parquet file has been generated in s3



Amazon S3 > Buckets > leonprocessed > vehicleprocessed/

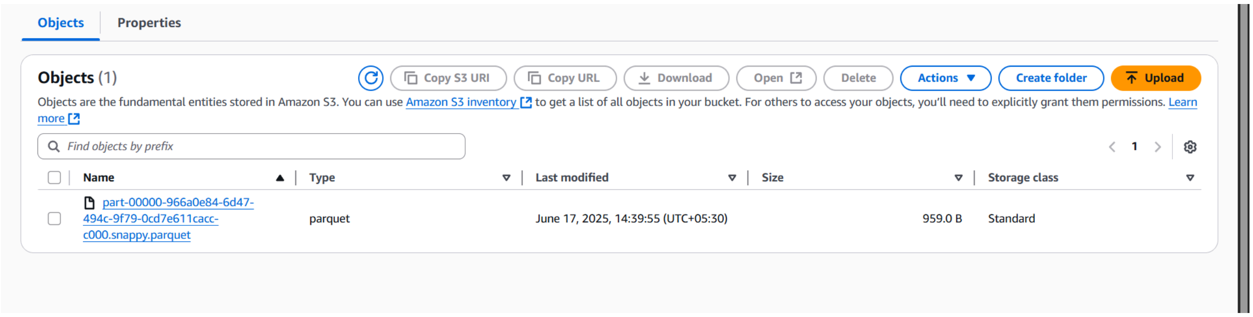
vehicleprocessed/

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
part-00000-14a62c42-d2f8-45d5-9036-2f278986a46c-c000.snappy.parquet	parquet	June 17, 2025, 14:39:45 (UTC+05:30)	1.7 KB	Standard

Aggregated data is stored in vehicleaggregated/



Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

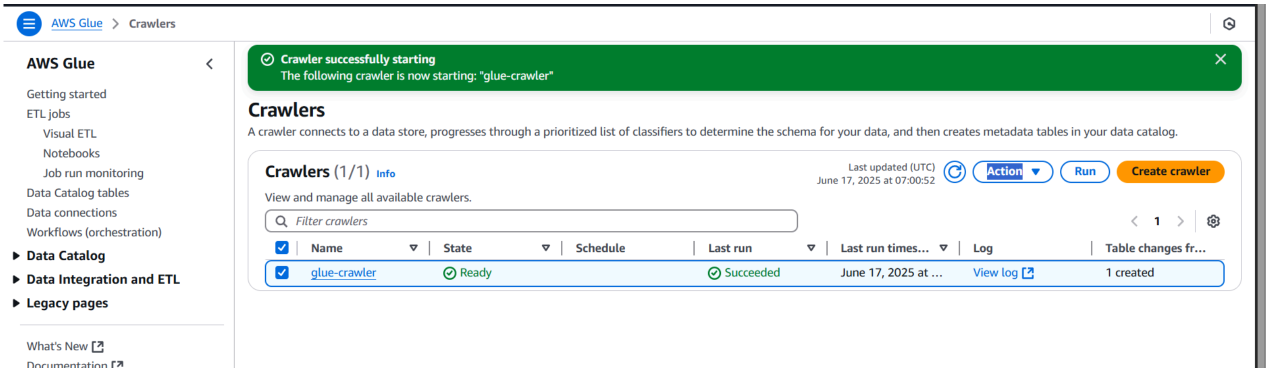
Name	Type	Last modified	Size	Storage class
part-00000-966a0e84-6d47-494c-9f79-0cd7e611cacc-c000.snappy.parquet	parquet	June 17, 2025, 14:39:55 (UTC+05:30)	959.0 B	Standard

4. Data Storage Amazon S3

/raw : for storing the data from kinesis
/processed : for storing the processed data from glue
/aggregated : for aggregated data

5. Hive + Glue Catalog AWS Glue Crawler

- Create a Glue crawler.
- Choose the S3 bucket where the processed data is stored
- Give IAM role as per required.
- Create a database for storing the data.
- Run the crawler.



AWS Glue > Crawlers

Crawler successfully starting
The following crawler is now starting: "glue-crawler"

Crawlers

A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in your data catalog.

Crawlers (1/1) Info

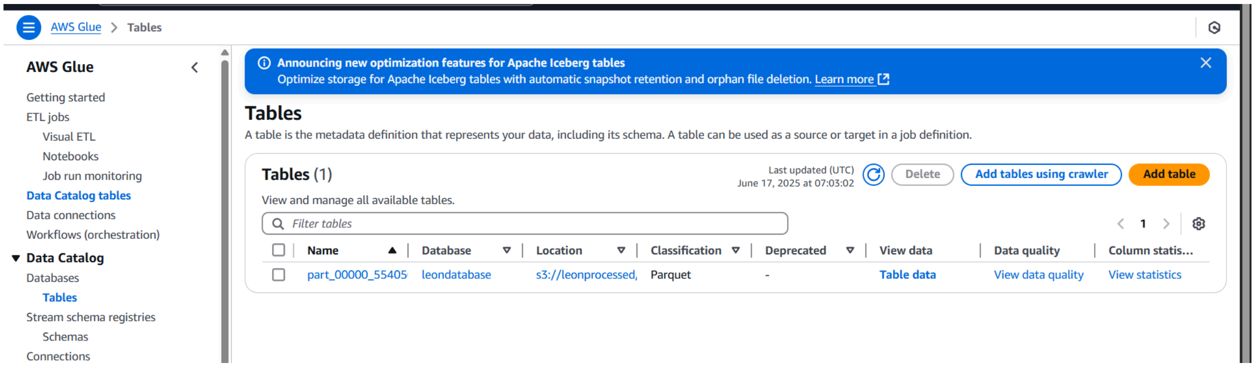
Last updated (UTC) June 17, 2025 at 07:00:52

View and manage all available crawlers.

Filter crawlers

☑	Name	State	Schedule	Last run	Last run times...	Log	Table changes fr...
☑	glue-crawler	Ready		Succeeded	June 17, 2025 at ...	View log	1 created

- Table will be created



AWS Glue > Tables

Announcing new optimization features for Apache Iceberg tables
Optimize storage for Apache Iceberg tables with automatic snapshot retention and orphan file deletion. Learn more

Tables

A table is the metadata definition that represents your data, including its schema. A table can be used as a source or target in a job definition.

Tables (1)

Last updated (UTC) June 17, 2025 at 07:03:02

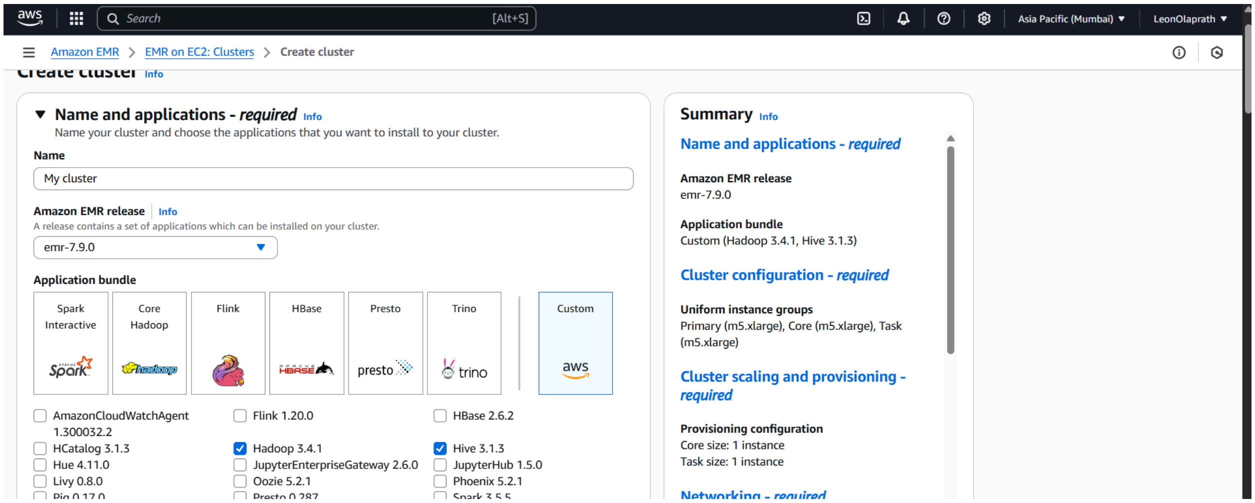
View and manage all available tables.

Filter tables

□	Name	Database	Location	Classification	Deprecated	View data	Data quality	Column statis...
□	part_00000_55405	leondatabase	s3://leonprocessed,	Parquet	-	Table data	View data quality	View statistics

Go to AWS EMR

- Create cluster
- Choose Hive and hadoop in application bundle
- Create the required IAM roles
- Choose security groups and key pair



Amazon EMR > EMR on EC2: Clusters > Create cluster

Create cluster Info

Name and applications - *required* Info

Name your cluster and choose the applications that you want to install to your cluster.

Name: My cluster

Amazon EMR release Info
A release contains a set of applications which can be installed on your cluster.
emr-7.9.0

Application bundle

□ AmazonCloudWatchAgent 1.300032.2
□ HCatalog 3.1.3
□ Hue 4.11.0
□ Livy 0.8.0
□ Pig 0.17.0
□ Flink 1.20.0
☑ Hadoop 3.4.1
□ JupyterEnterpriseGateway 2.6.0
□ Oozie 5.2.1
□ Presto 0.287
□ HBase 2.6.2
☑ Hive 3.1.3
□ JupyterHub 1.5.0
□ Phoenix 5.2.1
□ Spark 3.5.5

Summary Info

Name and applications - required

Amazon EMR release
emr-7.9.0

Application bundle
Custom (Hadoop 3.4.1, Hive 3.1.3)

Cluster configuration - required

Uniform instance groups
Primary (m5.xlarge), Core (m5.xlarge), Task (m5.xlarge)

Cluster scaling and provisioning - required

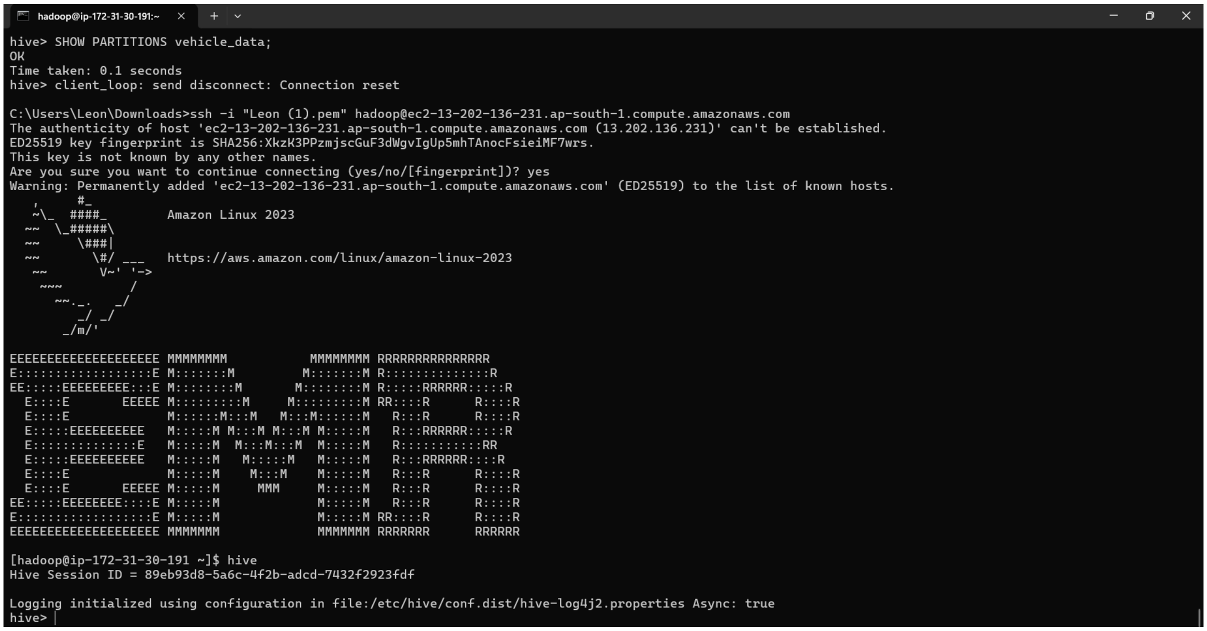
Provisioning configuration
Core size: 1 instance
Task size: 1 instance

Networking - required

- In edit software settings add this so that it tells Hive to use the AWS Glue Data Catalog as the metastore instead of the default Hive metastore.

```
[
  {
    "classification": "hive-site",
    "properties": {
      "hive.metastore.client.factory.class":
"com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory"
    }
  }
]
```

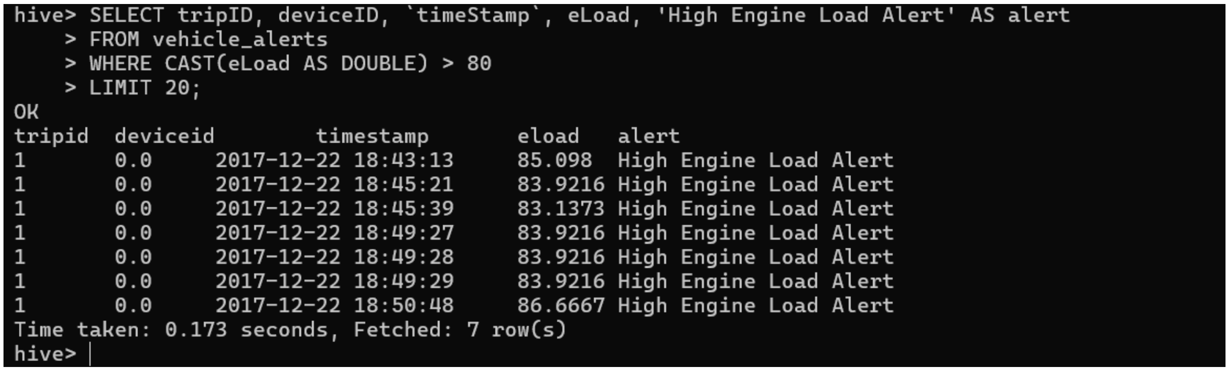
- In IAM permissions give glue full access
- After cluster is running connect into it through ssh
- Type hive
- Use database databasename
- Show tables ; will show the tables
- Query on the table created by the crawler



```
hive> SHOW PARTITIONS vehicle_data;
OK
Time taken: 0.1 seconds
hive> client_loop: send disconnect: Connection reset

C:\Users\Leon\Downloads>ssh -i "Leon (1).pem" hadoop@ec2-13-202-136-231.ap-south-1.compute.amazonaws.com
The authenticity of host 'ec2-13-202-136-231.ap-south-1.compute.amazonaws.com (13.202.136.231)' can't be established.
ED25519 key fingerprint is SHA256:XkzK3PPzmjscGuF3dWgvIgUp5mhTAnocFsieiMF7wrs.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-13-202-136-231.ap-south-1.compute.amazonaws.com' (ED25519) to the list of known hosts.
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'

EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-30-191 ~]$ hive
Hive Session ID = 89eb93d8-5a6c-4f2b-adcd-7432f2923fdf

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j2.properties Async: true
hive> 
```

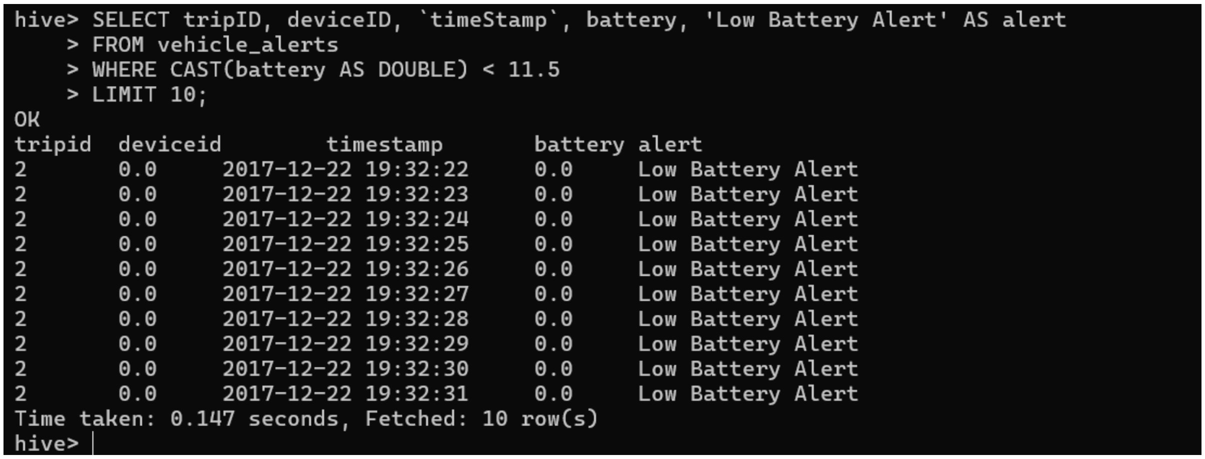
Now querying based on my use case i.e., focusing on vehicle health monitoring and predictive maintenance by analyzing historical telematics data such as RPM, engine load, coolant temperature, and battery. By identifying patterns like frequent overheating, high engine strain, or recurring faults, it enables the generation of maintenance recommendations and detection of long-term anomalies.

1. Detect High Engine Load



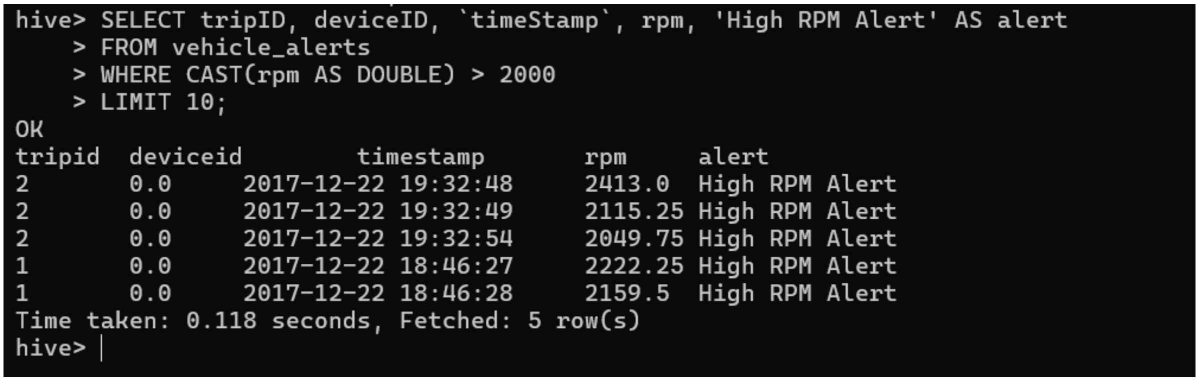
```
hive> SELECT tripID, deviceID, `timeStamp`, eLoad, 'High Engine Load Alert' AS alert
    > FROM vehicle_alerts
    > WHERE CAST(eLoad AS DOUBLE) > 80
    > LIMIT 20;
OK
tripid  deviceid        timestamp       eload   alert
1       0.0     2017-12-22 18:43:13     85.098  High Engine Load Alert
1       0.0     2017-12-22 18:45:21     83.9216 High Engine Load Alert
1       0.0     2017-12-22 18:45:39     83.1373 High Engine Load Alert
1       0.0     2017-12-22 18:49:27     83.9216 High Engine Load Alert
1       0.0     2017-12-22 18:49:28     83.9216 High Engine Load Alert
1       0.0     2017-12-22 18:49:29     83.9216 High Engine Load Alert
1       0.0     2017-12-22 18:50:48     86.6667 High Engine Load Alert
Time taken: 0.173 seconds, Fetched: 7 row(s)
hive> |
```

2. Detect Low Battery



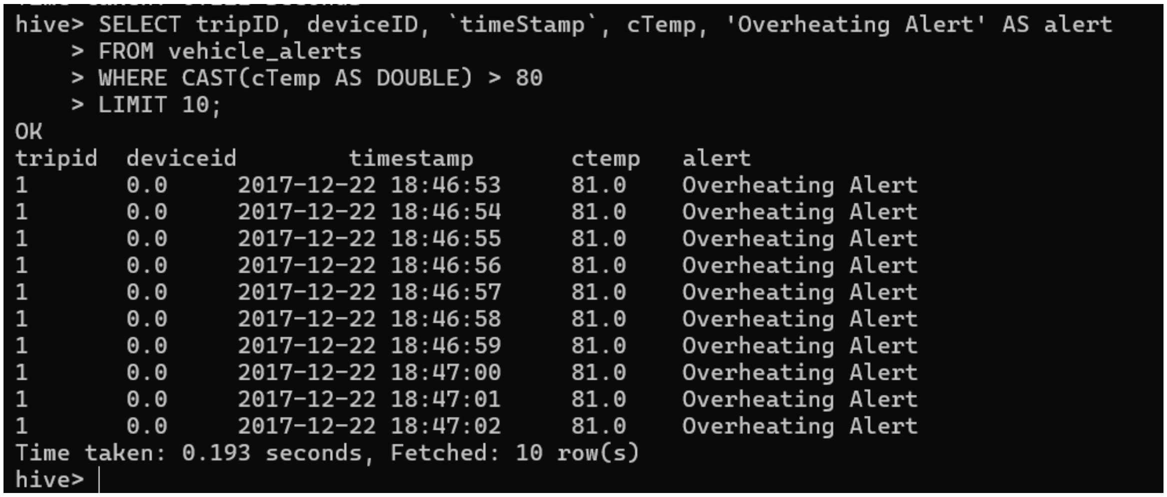
```
hive> SELECT tripID, deviceID, `timeStamp`, battery, 'Low Battery Alert' AS alert
    > FROM vehicle_alerts
    > WHERE CAST(battery AS DOUBLE) < 11.5
    > LIMIT 10;
OK
tripid  deviceid        timestamp       battery alert
2       0.0     2017-12-22 19:32:22     0.0     Low Battery Alert
2       0.0     2017-12-22 19:32:23     0.0     Low Battery Alert
2       0.0     2017-12-22 19:32:24     0.0     Low Battery Alert
2       0.0     2017-12-22 19:32:25     0.0     Low Battery Alert
2       0.0     2017-12-22 19:32:26     0.0     Low Battery Alert
2       0.0     2017-12-22 19:32:27     0.0     Low Battery Alert
2       0.0     2017-12-22 19:32:28     0.0     Low Battery Alert
2       0.0     2017-12-22 19:32:29     0.0     Low Battery Alert
2       0.0     2017-12-22 19:32:30     0.0     Low Battery Alert
2       0.0     2017-12-22 19:32:31     0.0     Low Battery Alert
Time taken: 0.147 seconds, Fetched: 10 row(s)
hive> |
```

3. Detect High RPM



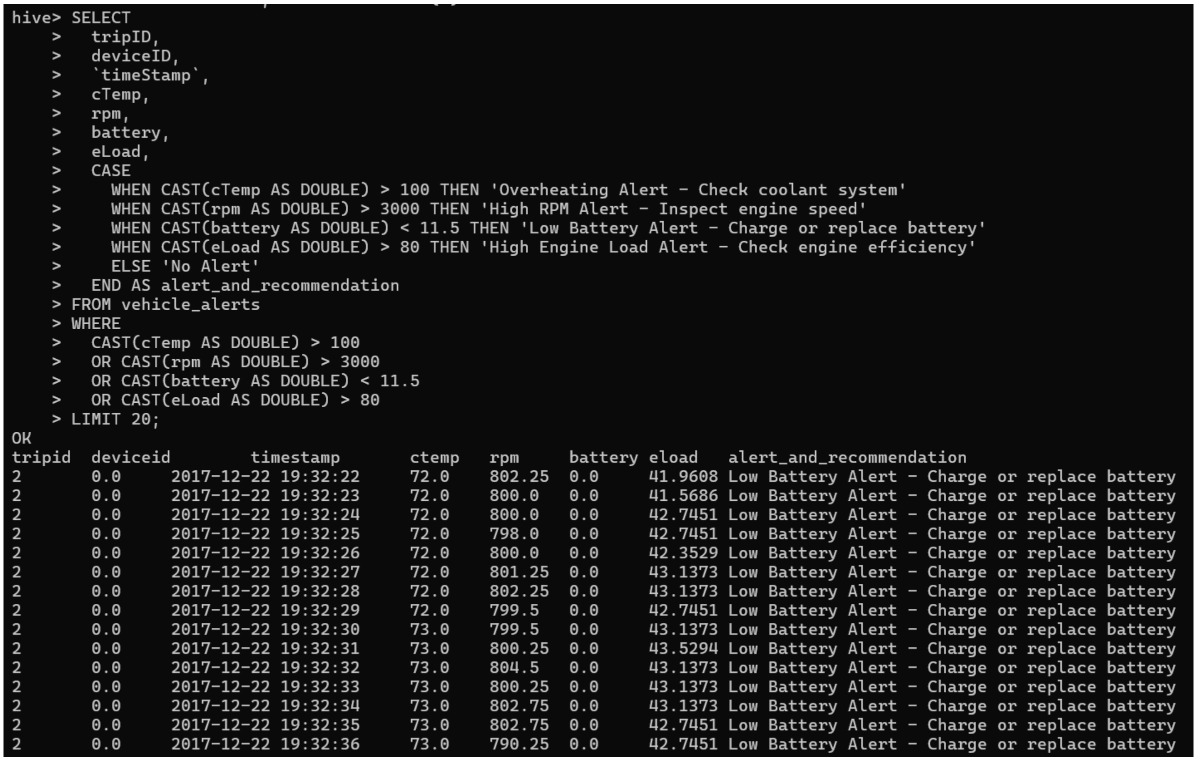
```
hive> SELECT tripID, deviceID, `timeStamp`, rpm, 'High RPM Alert' AS alert
    > FROM vehicle_alerts
    > WHERE CAST(rpm AS DOUBLE) > 2000
    > LIMIT 10;
OK
tripid  deviceid        timestamp       rpm     alert
2       0.0     2017-12-22 19:32:48     2413.0  High RPM Alert
2       0.0     2017-12-22 19:32:49     2115.25 High RPM Alert
2       0.0     2017-12-22 19:32:54     2049.75 High RPM Alert
1       0.0     2017-12-22 18:46:27     2222.25 High RPM Alert
1       0.0     2017-12-22 18:46:28     2159.5  High RPM Alert
Time taken: 0.118 seconds, Fetched: 5 row(s)
hive> |
```

4. Detect overheating



```
hive> SELECT tripID, deviceID, `timeStamp`, cTemp, 'Overheating Alert' AS alert
    > FROM vehicle_alerts
    > WHERE CAST(cTemp AS DOUBLE) > 80
    > LIMIT 10;
OK
tripid  deviceid        timestamp       ctemp   alert
1       0.0     2017-12-22 18:46:53     81.0    Overheating Alert
1       0.0     2017-12-22 18:46:54     81.0    Overheating Alert
1       0.0     2017-12-22 18:46:55     81.0    Overheating Alert
1       0.0     2017-12-22 18:46:56     81.0    Overheating Alert
1       0.0     2017-12-22 18:46:57     81.0    Overheating Alert
1       0.0     2017-12-22 18:46:58     81.0    Overheating Alert
1       0.0     2017-12-22 18:46:59     81.0    Overheating Alert
1       0.0     2017-12-22 18:47:00     81.0    Overheating Alert
1       0.0     2017-12-22 18:47:01     81.0    Overheating Alert
1       0.0     2017-12-22 18:47:02     81.0    Overheating Alert
Time taken: 0.193 seconds, Fetched: 10 row(s)
hive> 
```

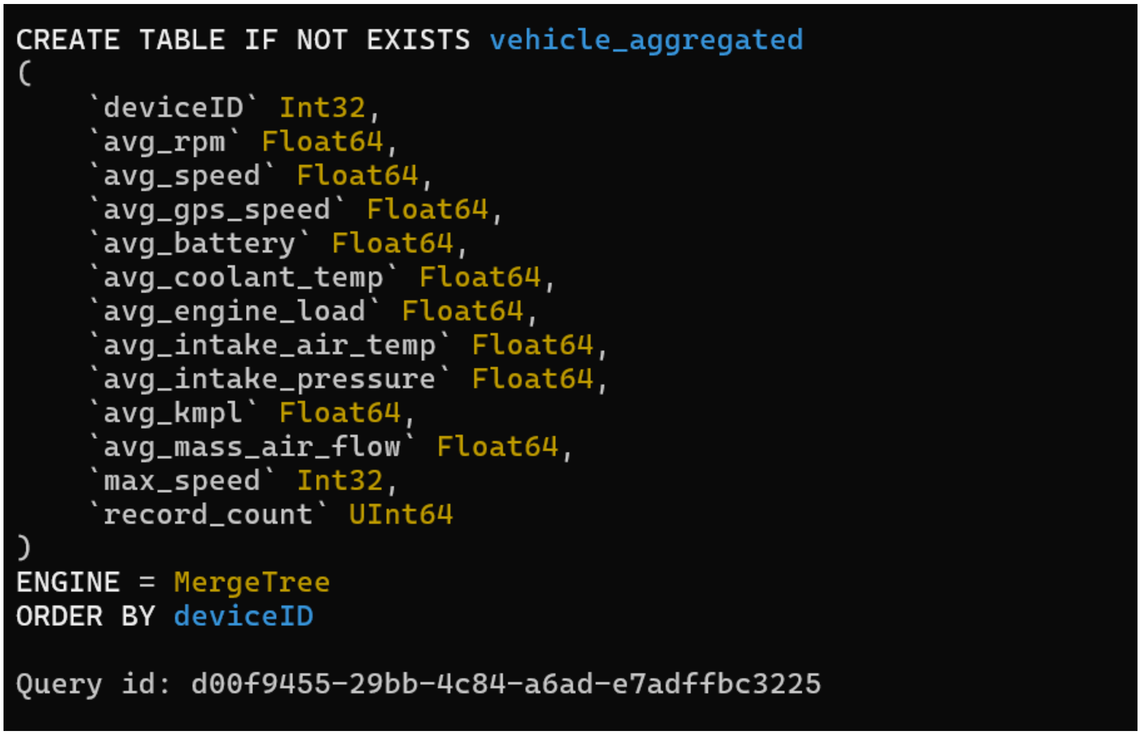
5. Total anomaly detection and maintenance recommendations



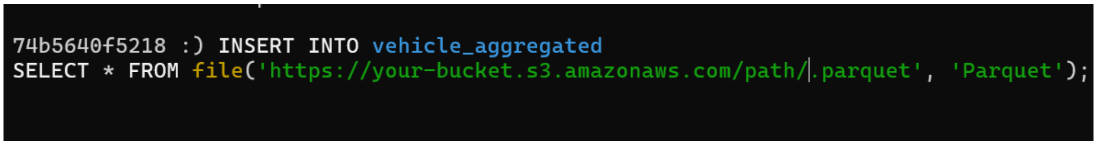
```
hive> SELECT
    >   tripID,
    >   deviceID,
    >   `timeStamp`,
    >   cTemp,
    >   rpm,
    >   battery,
    >   eLoad,
    >   CASE
    >     WHEN CAST(cTemp AS DOUBLE) > 100 THEN 'Overheating Alert - Check coolant system'
    >     WHEN CAST(rpm AS DOUBLE) > 3000 THEN 'High RPM Alert - Inspect engine speed'
    >     WHEN CAST(battery AS DOUBLE) < 11.5 THEN 'Low Battery Alert - Charge or replace battery'
    >     WHEN CAST(eLoad AS DOUBLE) > 80 THEN 'High Engine Load Alert - Check engine efficiency'
    >     ELSE 'No Alert'
    >   END AS alert_and_recommendation
    > FROM vehicle_alerts
    > WHERE
    >   CAST(cTemp AS DOUBLE) > 100
    >   OR CAST(rpm AS DOUBLE) > 3000
    >   OR CAST(battery AS DOUBLE) < 11.5
    >   OR CAST(eLoad AS DOUBLE) > 80
    > LIMIT 20;
OK
tripid  deviceid        timestamp       ctemp   rpm     battery eload   alert_and_recommendation
2       0.0     2017-12-22 19:32:22     72.0    802.25  0.0     41.9608 Low Battery Alert - Charge or replace battery
2       0.0     2017-12-22 19:32:23     72.0    800.0   0.0     41.5686 Low Battery Alert - Charge or replace battery
2       0.0     2017-12-22 19:32:24     72.0    800.0   0.0     42.7451 Low Battery Alert - Charge or replace battery
2       0.0     2017-12-22 19:32:25     72.0    798.0   0.0     42.7451 Low Battery Alert - Charge or replace battery
2       0.0     2017-12-22 19:32:26     72.0    800.0   0.0     42.3529 Low Battery Alert - Charge or replace battery
2       0.0     2017-12-22 19:32:27     72.0    801.25  0.0     43.1373 Low Battery Alert - Charge or replace battery
2       0.0     2017-12-22 19:32:28     72.0    802.25  0.0     43.1373 Low Battery Alert - Charge or replace battery
2       0.0     2017-12-22 19:32:29     72.0    799.5   0.0     42.7451 Low Battery Alert - Charge or replace battery
2       0.0     2017-12-22 19:32:30     73.0    799.5   0.0     43.1373 Low Battery Alert - Charge or replace battery
2       0.0     2017-12-22 19:32:31     73.0    800.25  0.0     43.5294 Low Battery Alert - Charge or replace battery
2       0.0     2017-12-22 19:32:32     73.0    804.5   0.0     43.1373 Low Battery Alert - Charge or replace battery
2       0.0     2017-12-22 19:32:33     73.0    800.25  0.0     43.1373 Low Battery Alert - Charge or replace battery
2       0.0     2017-12-22 19:32:34     73.0    802.75  0.0     43.1373 Low Battery Alert - Charge or replace battery
2       0.0     2017-12-22 19:32:35     73.0    802.75  0.0     42.7451 Low Battery Alert - Charge or replace battery
2       0.0     2017-12-22 19:32:36     73.0    790.25  0.0     42.7451 Low Battery Alert - Charge or replace battery
```

6. ClickHouse: Fast OLAP store

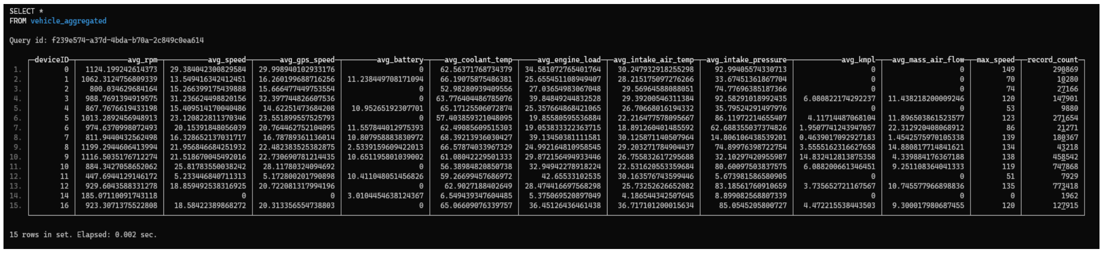
- Using clickhouse inside docker in local machine
- Open cmd and enter docker exec -it clickhouse-server clickhouse-client
- Create table and give s3 as source which has the aggregated data as parquet file



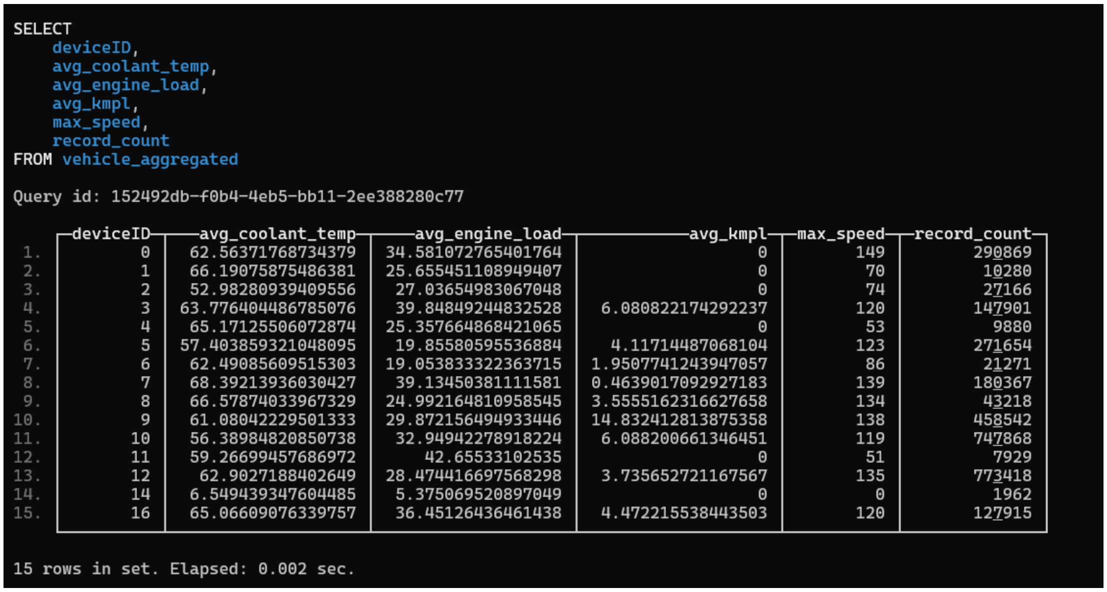
```
CREATE TABLE IF NOT EXISTS vehicle_aggregated
(
    `deviceID` Int32,
    `avg_rpm` Float64,
    `avg_speed` Float64,
    `avg_gps_speed` Float64,
    `avg_battery` Float64,
    `avg_coolant_temp` Float64,
    `avg_engine_load` Float64,
    `avg_intake_air_temp` Float64,
    `avg_intake_pressure` Float64,
    `avg_kmpl` Float64,
    `avg_mass_air_flow` Float64,
    `max_speed` Int32,
    `record_count` UInt64
)
ENGINE = MergeTree
ORDER BY deviceID

Query id: d00f9455-29bb-4c84-a6ad-e7adffbc3225
```



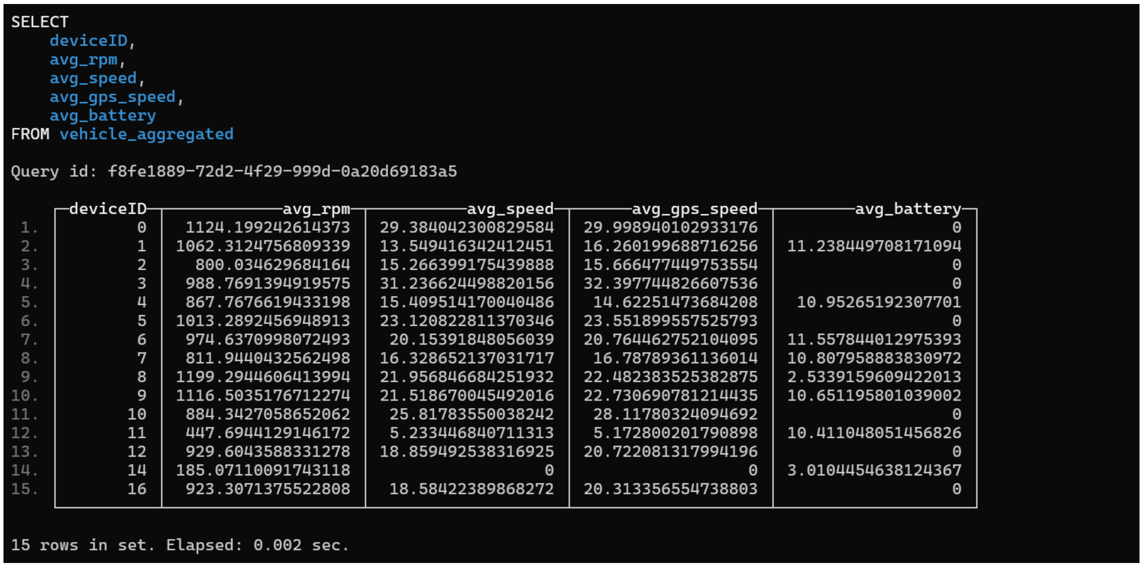
```
74b5640f5218 :) INSERT INTO vehicle_aggregated
SELECT * FROM file('https://your-bucket.s3.amazonaws.com/path/|.parquet', 'Parquet');
```



```
SELECT *
FROM vehicle_aggregated

Query id: f239e574-a37d-4bda-b70a-2c849c0ea614

    deviceID             avg_rpm           avg_speed       avg_gps_speed         avg_battery    avg_coolant_temp     avg_engine_load  avg_intake_air_temp  avg_intake_pressure            avg_kmpl   avg_mass_air_flow  max_speed  record_count
 1.        0   1124.199242614373  29.384042300829584  29.998940102933176                   0   62.56371768734379  34.581072765401764   30.247932918255298    92.99405574330713                   0                   0        149        290869
 2.        1  1062.3124756809339  13.549416342412451  16.260199688716256  11.238449708171094   66.19075875486381  25.655451108949407   28.215175097276266    33.67451361867704                   0                   0         70         10280
 3.        2    800.034629684164  15.266399175439888  15.666477449753554                   0   52.98280939409556   27.03654983067048    29.56964588088051    74.77696385187366                   0                   0         74         27166
 4.        3   988.7691394919575  31.236624498820156  32.397744826607536                   0  63.776404486785076   39.84849244832528    29.39200546311384    92.58291018992435   6.080822174292237  11.438218200009246        120        147901
 5.        4   867.7676619433198  15.409514170040486   14.62251473684208   10.95265192307701   65.17125506072874  25.357664868421065    26.70668016194332    35.79524291497976                   0                   0         53          9880
 6.        5  1013.2892456948913  23.120822811370346  23.551899557525793                   0  57.403859321048095   19.85580595536884   22.216477578095667    86.11972214655407    4.11714487068104  11.896503861523577        123        271654
 7.        6   974.6370998072493   20.15391848056039  20.764462752104095  11.557844012975393   62.49085609515303  19.053833322363715   18.891260401485592   62.688355037374826  1.9507741243947057  22.312920408068912         86         21271
 8.        7   811.9440432562498  16.328652137031717   16.78789361136014  10.807958883830972   68.39213936030427   39.13450381111581   30.125871140507964   14.806106438539201  0.4639017092927183  1.4542575970105338        139        180367
 9.        8  1199.2944606413994  21.956846684251932  22.482383525382875  2.5339159609422013   66.57874033967329  24.992164810958545   29.203271784904437    74.89976398722754  3.5555162316627658  14.880817714841621        134         43218
10.        9  1116.5035176712274  21.518670045492016  22.730690781214435  10.651195801039002   61.08042229501333  29.872156494933446   26.755832617295688    32.10297420955987  14.832412813875358   4.339884176367188        138        458542
11.       10   884.3427058652062   25.81783550038242   28.11780324094692                   0   56.38984820850738   32.94942278918224   22.531620553359684    80.60097503837575   6.088200661346451   9.251108364041333        119        747868
12.       11   447.6944129146172   5.233446840711313   5.172800201790898  10.411048051456826   59.26699457686972      42.65533102535   30.163576743599446    5.673981586580905                   0                   0         51          7929
13.       12   929.6043588331278  18.859492538316925  20.722081317994196                   0    62.9027188402649  28.474416697568298    25.73252626652082    83.18561760910659   3.735652721167567  10.745577966898836        135        773418
14.       14  185.07110091743118                   0                   0  3.0104454638124367   6.549439347604485   5.375069520897049    4.186544342507645    8.899082568807339                   0                   0          0          1962
15.       16   923.3071375522808   18.58422389868272  20.313356554738803                   0   65.06609076339757   36.45126436461438   36.717101200015634     85.0545205800727   4.472215538443503   9.300017980687455        120        127915

15 rows in set. Elapsed: 0.002 sec.
```



```
SELECT
    deviceID,
    avg_coolant_temp,
    avg_engine_load,
    avg_kmpl,
    max_speed,
    record_count
FROM vehicle_aggregated

Query id: 152492db-f0b4-4eb5-bb11-2ee388280c77

    ┌─deviceID─┬─────avg_coolant_temp─┬────avg_engine_load─┬────────────avg_kmpl─┬─max_speed─┬─record_count─┐
 1. │        0 │    62.56371768734379 │ 34.581072765401764 │                    0 │       149 │       290869 │
 2. │        1 │    66.19075875486381 │ 25.655451108949407 │                    0 │        70 │        10280 │
 3. │        2 │    52.98280939409556 │  27.03654983067048 │                    0 │        74 │        27166 │
 4. │        3 │   63.776404486785076 │  39.84849244832528 │    6.080822174292237 │       120 │       147901 │
 5. │        4 │    65.17125506072874 │ 25.357664868421065 │                    0 │        53 │         9880 │
 6. │        5 │   57.403859321048095 │  19.85580595536884 │     4.11714487068104 │       123 │       271654 │
 7. │        6 │    62.49085609515303 │ 19.053833322363715 │   1.9507741243947057 │        86 │        21271 │
 8. │        7 │    68.39213936030427 │  39.13450381111581 │   0.4639017092927183 │       139 │       180367 │
 9. │        8 │    66.57874033967329 │ 24.992164810958545 │   3.5555162316627658 │       134 │        43218 │
10. │        9 │    61.08042229501333 │ 29.872156494933446 │   14.832412813875358 │       138 │       458542 │
11. │       10 │    56.38984820850738 │  32.94942278918224 │    6.088200661346451 │       119 │       747868 │
12. │       11 │    59.26699457686972 │     42.65533102535 │                    0 │        51 │         7929 │
13. │       12 │     62.9027188402649 │ 28.474416697568298 │    3.735652721167567 │       135 │       773418 │
14. │       14 │    6.549439347604485 │  5.375069520897049 │                    0 │         0 │         1962 │
15. │       16 │    65.06609076339757 │  36.45126436461438 │    4.472215538443503 │       120 │       127915 │
    └──────────┴──────────────────────┴────────────────────┴──────────────────────┴───────────┴──────────────┘

15 rows in set. Elapsed: 0.002 sec.
```

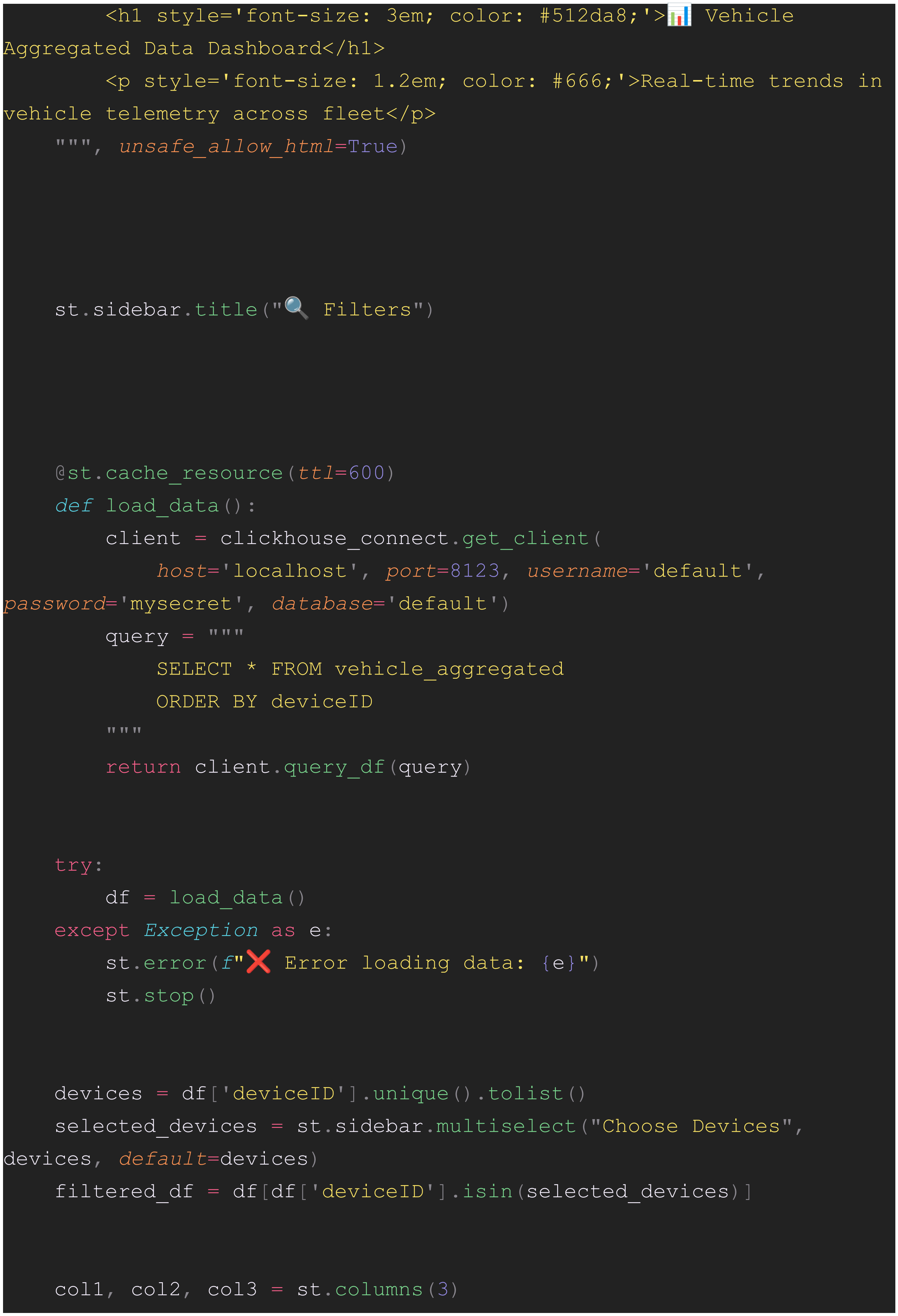


```
SELECT
    deviceID,
    avg_rpm,
    avg_speed,
    avg_gps_speed,
    avg_battery
FROM vehicle_aggregated

Query id: f8fe1889-72d2-4f29-999d-0a20d69183a5

    ┌─deviceID─┬────────────avg_rpm─┬──────────avg_speed─┬──────avg_gps_speed─┬────────avg_battery─┐
 1. │        0 │  1124.199242614373 │ 29.384042300829584 │ 29.998940102933176 │                  0 │
 2. │        1 │ 1062.3124756809339 │ 13.549416342412451 │ 16.260199688716256 │ 11.238449708171094 │
 3. │        2 │   800.034629684164 │ 15.266399175439888 │ 15.666477449753554 │                  0 │
 4. │        3 │  988.7691394919575 │ 31.236624498820156 │ 32.397744826607536 │                  0 │
 5. │        4 │  867.7676619433198 │ 15.409514170040486 │  14.62251473684208 │  10.95265192307701 │
 6. │        5 │ 1013.2892456948913 │ 23.120822811370346 │ 23.551899557525793 │                  0 │
 7. │        6 │  974.6370998072493 │  20.15391848056039 │ 20.764462752104095 │ 11.557844012975393 │
 8. │        7 │  811.9440432562498 │ 16.328652137031717 │  16.78789361136014 │ 10.807958883830972 │
 9. │        8 │ 1199.2944606413994 │ 21.956846684251932 │ 22.482383525382875 │  2.533915960942201 3│
10. │        9 │ 1116.5035176712274 │ 21.518670045492016 │ 22.730690781214435 │ 10.651195801039002 │
11. │       10 │  884.3427058652062 │  25.81783550038242 │  28.11780324094692 │                  0 │
12. │       11 │  447.6944129146172 │  5.233446840711313 │  5.172800201790898 │  10.41104805145682 6│
13. │       12 │  929.6043588331278 │ 18.859492538316925 │ 20.722081317994196 │                  0 │
14. │       14 │ 185.07110091743118 │                  0 │                  0 │ 3.0104454638124367 │
15. │       16 │  923.3071375522808 │  18.58422389868272 │ 20.313356554738803 │                  0 │
    └──────────┴────────────────────┴────────────────────┴────────────────────┴────────────────────┘

15 rows in set. Elapsed: 0.002 sec.
```

7. Dashboard using streamlit in python

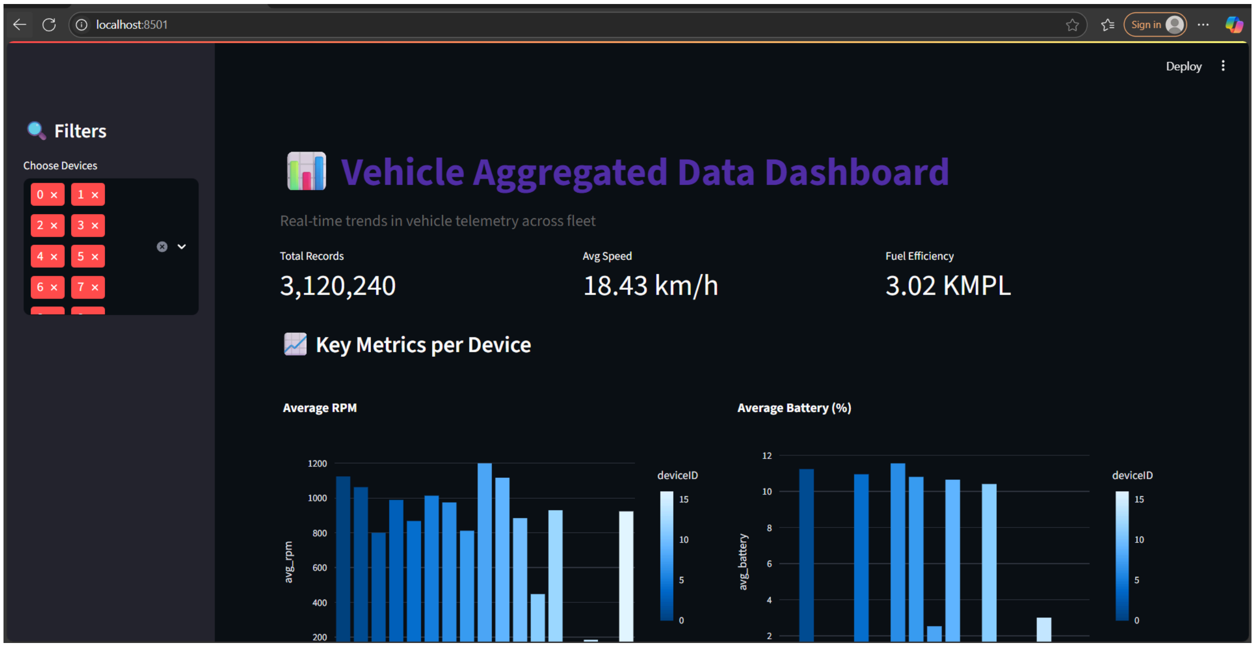
- Install streamlit and clickhouse-connect in python
- Connect to clickhouse client and the required table
- Use plotly for visualization

```
import streamlit as st
import clickhouse_connect
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go


st.set_page_config(
    page_title="🚗 Vehicle Dashboard",
    layout="wide",
    initial_sidebar_state="expanded"
)


st.markdown("""
    <style>
        body {
            background-color: #f8f8fb;
            font-family: 'Segoe UI', sans-serif;
        }
        .css-1aumxhk, .css-ffhzg2 {
            background-color: #fff;
            border-radius: 12px;
            padding: 2rem;
            box-shadow: 0 2px 10px rgba(0,0,0,0.05);
        }
    </style>
""", unsafe_allow_html=True)




st.markdown("""
```



```
        <h1 style='font-size: 3em; color: #512da8;'>📊 Vehicle Aggregated Data Dashboard</h1>
        <p style='font-size: 1.2em; color: #666;'>Real-time trends in vehicle telemetry across fleet</p>
    """, unsafe_allow_html=True)



    st.sidebar.title("🔍 Filters")



    @st.cache_resource(ttl=600)
    def load_data():
        client = clickhouse_connect.get_client(
            host='localhost', port=8123, username='default', password='mysecret', database='default')
        query = """
            SELECT * FROM vehicle_aggregated
            ORDER BY deviceID
        """
        return client.query_df(query)


    try:
        df = load_data()
    except Exception as e:
        st.error(f"❌ Error loading data: {e}")
        st.stop()


    devices = df['deviceID'].unique().tolist()
    selected_devices = st.sidebar.multiselect("Choose Devices", devices, default=devices)
    filtered_df = df[df['deviceID'].isin(selected_devices)]


    col1, col2, col3 = st.columns(3)
    col1.metric("Total Records",
f"{filtered_df['record_count'].sum():,}")
    col2.metric("Avg Speed", f"{filtered_df['avg_speed'].mean():.2f}
km/h")
    col3.metric("Fuel Efficiency",
f"{filtered_df['avg_kmpl'].mean():.2f} KMPL")


    st.markdown("### 📈 Key Metrics per Device")
    col4, col5 = st.columns(2)


    with col4:
        fig = px.bar(filtered_df, x='deviceID', y='avg_rpm',
color='deviceID',
                    color_discrete_sequence=['#ff6f61', '#6a1b9a'],
                    title="Average RPM")
        st.plotly_chart(fig, use_container_width=True)


    with col5:
        fig = px.bar(filtered_df, x='deviceID', y='avg_battery',
color='deviceID',
                    color_discrete_sequence=['#ffa600', '#a05195'],
                    title="Average Battery (%)")
        st.plotly_chart(fig, use_container_width=True)


    st.markdown("### 📉 Trends & Comparisons")
    col6, col7 = st.columns(2)


    with col6:
        fig = px.line(filtered_df, x='deviceID',
y='avg_coolant_temp', markers=True,
                    color_discrete_sequence=['#ff7c43'],
                    title="Coolant Temp Trend")
        st.plotly_chart(fig, use_container_width=True)
    with col7:
        fig = px.line(filtered_df, x='deviceID', y='avg_kmpl', markers=True,
                      color_discrete_sequence=['#2f4b7c'],
                      title="Fuel Efficiency (KMPL)")
        st.plotly_chart(fig, use_container_width=True)


    st.markdown("### ⚙️ Intake Pressure vs Fuel Efficiency")
    fig = px.scatter(filtered_df, x='avg_intake_pressure', y='avg_kmpl',
                      color='deviceID', size='avg_engine_load',
color_discrete_sequence=px.colors.qualitative.Prism,
                      title="Fuel Efficiency vs Intake Pressure",
                      labels={'avg_intake_pressure': 'Intake Pressure', 'avg_kmpl': 'KMPL'})
    fig.update_traces(marker=dict(opacity=0.8, line=dict(width=1, color='DarkSlateGrey')))
    st.plotly_chart(fig, use_container_width=True)


    st.markdown("### 🧱 Engine Load Distribution")
    fig = px.box(filtered_df, x='deviceID', y='avg_engine_load', color='deviceID',
                color_discrete_sequence=px.colors.qualitative.Bold,
                title="Engine Load Distribution per Device",
                points="all")
    st.plotly_chart(fig, use_container_width=True)


    st.markdown("---")
    st.markdown("<p style='text-align:center;'>Designed by Leon Jervis Olaprath</p>", unsafe_allow_html=True)
```

- Run streamlit using streamlit run file.py
- Make sure the clickhouse client is running
- The dashboard will be visible in localhost:8501



🔍 Filters

Choose Devices

0 × 1 ×
2 × 3 ×
4 × 5 ×
6 × 7 ×

📊 Vehicle Aggregated Data Dashboard

Real-time trends in vehicle telemetry across fleet

Total Records
3,120,240

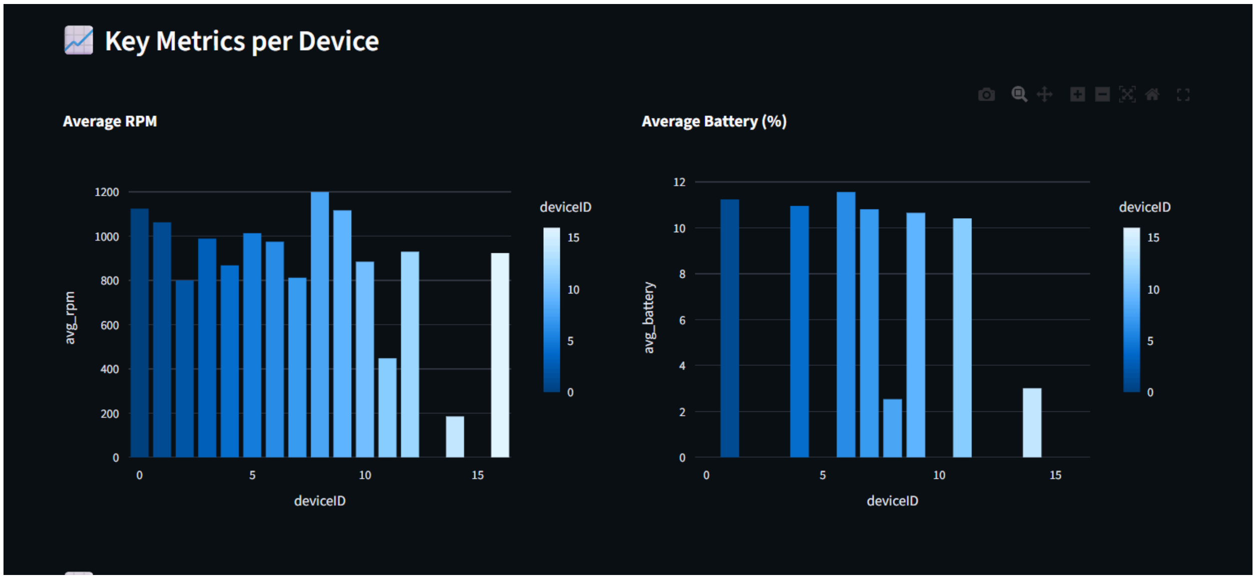
Avg Speed
18.43 km/h

Fuel Efficiency
3.02 KMPL

📈 Key Metrics per Device

Average RPM

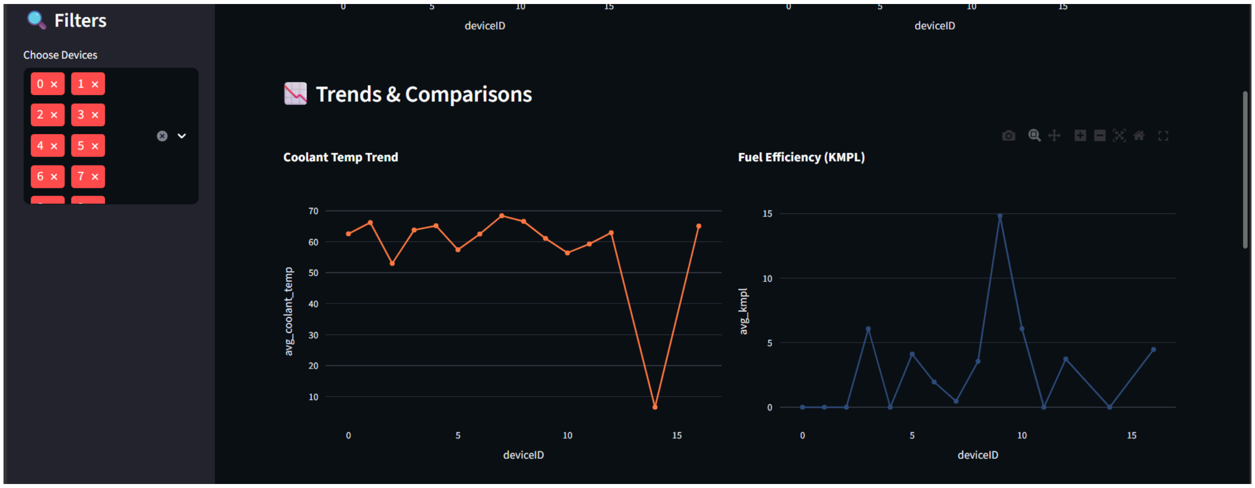
Average Battery (%)



📈 Key Metrics per Device

Average RPM

Average Battery (%)



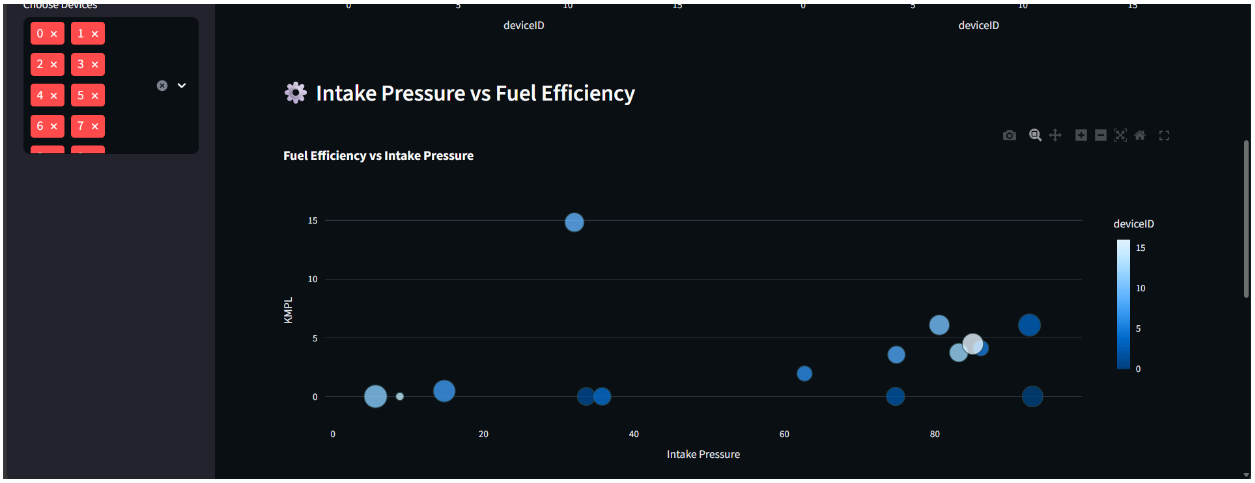
🔍 Filters

Choose Devices

📉 Trends & Comparisons

Coolant Temp Trend

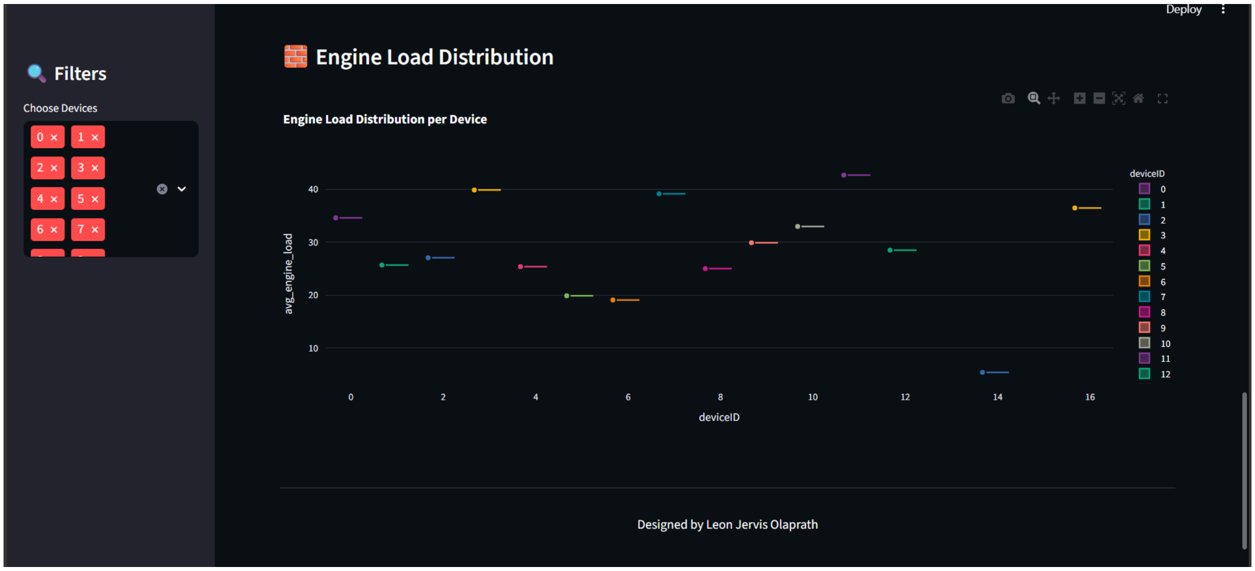
Fuel Efficiency (KMPL)



Choose Devices

⚙️ Intake Pressure vs Fuel Efficiency

Fuel Efficiency vs Intake Pressure



🔍 Filters

Choose Devices

🧱 Engine Load Distribution

Engine Load Distribution per Device

Designed by Leon Jervis Olaprath

REFERENCES

Kaggle :- https://www.kaggle.com/datasets/yunlevin/levin-vehicle-telematics