Project Report

Introduction

I have explored a dataset called nuscenes dataset which is used in autonomous vehicles.In this project, I have understood the structure of the dataset and I have loaded the data into csv files and I have implemented 33 queries out of which I have used streaming API's on 13 queries on this dataset using pyspark.

Dataset

Nuscenes dataset is a public large-scale dataset for autonomous vehicles.It enables researchers to study challenging urban driving situations using the full sensor suite of a real self-driving car.Motional is making driverless vehicles a safe, reliable, and accessible reality. By releasing a subset of our data to the public, Motional aims to support public research into computer vision and autonomous driving.For this purpose it has been collected 1000 driving scenes in Boston and Singapore, two cities that are known for their dense traffic and highly challenging driving situations. The scenes of 20 second length are manually selected to show a diverse and interesting set of driving maneuvers, traffic situations and unexpected behaviours. The rich complexity of nuScenes will encourage development of methods that enable safe driving in urban areas with dozens of objects per scene. Gathering data on different continents further allows us to study the generalization of computer vision algorithms across different locations, weather conditions, vehicle types, vegetation, road markings and left versus right hand traffic.For the nuScenes dataset approximately 15h of driving data in Boston and Singapore is collected.. Driving routes are carefully chosen to capture challenging scenarios

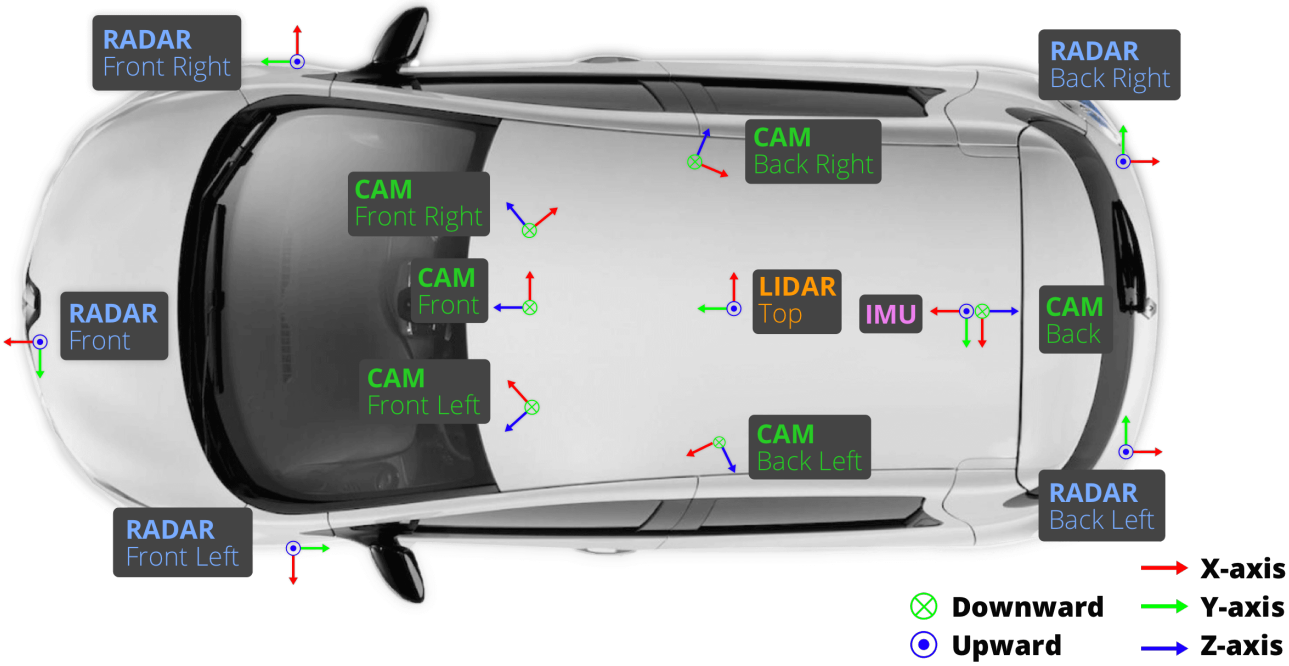
The nuScenes dataset is a large-scale autonomous driving dataset with **3d object annotations.** It features:

- Full sensor suite (1x LIDAR, 5x RADAR, 6x camera, IMU, GPS)
- 1000 scenes of 20s each
- 1,400,000 camera images
- 390,000 lidar sweeps
- Two diverse cities: Boston and Singapore

- Left versus right hand traffic
- Detailed map information
- 1.4M 3D bounding boxes manually annotated for 23 object classes
- Attributes such as visibility, activity and pose

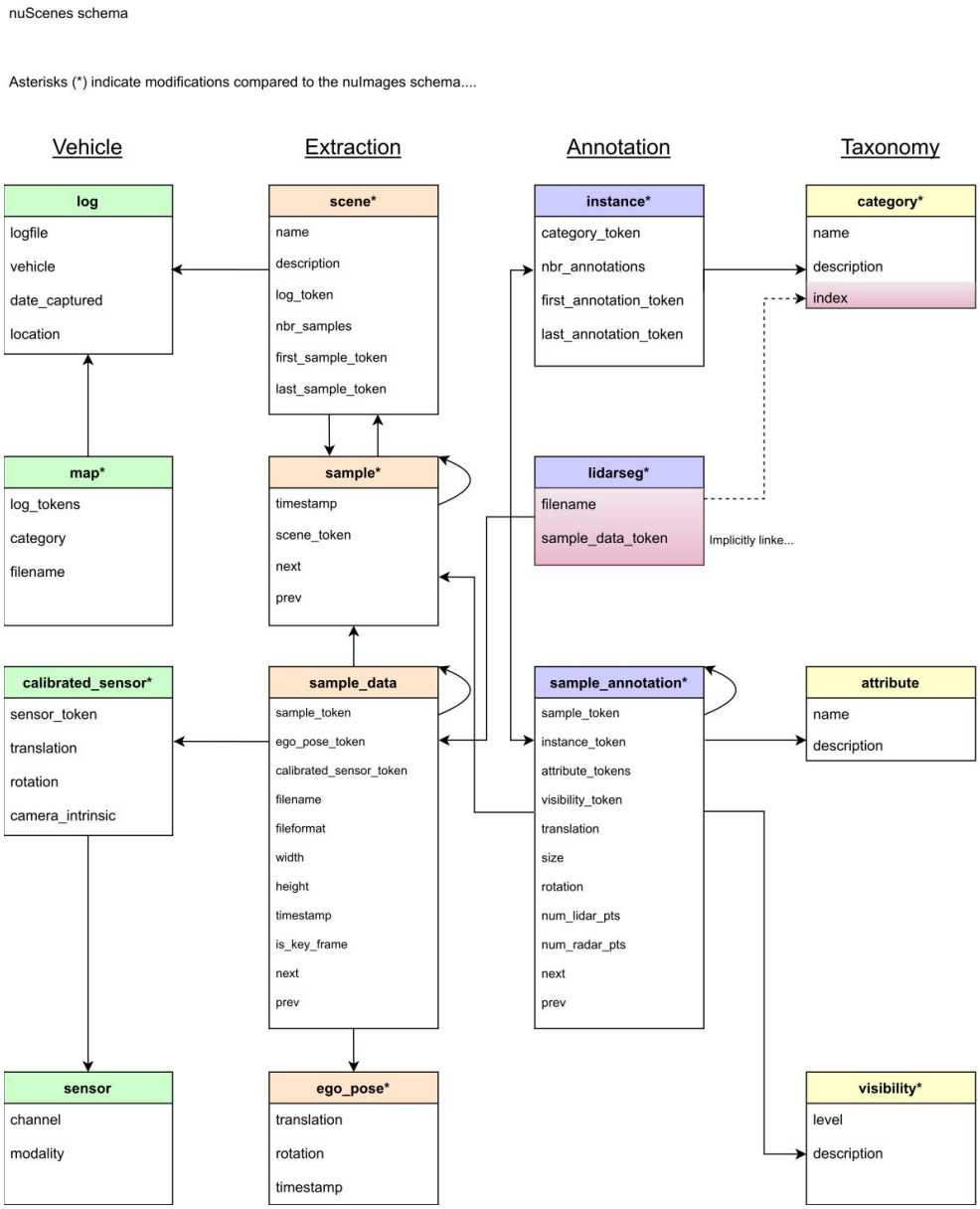
Car Setup

Two Renault Zoe cars with an identical sensor layout to drive in Boston and Singapore. The data was gathered from a research platform and is not indicative of the setup used in Motional products. Data released from the following sensors:



Structure of the Dataset

Nuscenes Schema :



- log : - Log information from which the data was extracted.
- scene : - 20 second snippet of a car's journey.
- sample : - An annotated snapshot of a scene at a particular timestamp.

- sample_data : - Data collected from a particular sensor.
- ego_pose : - Ego vehicle poses at a particular timestamp.
- sensor : - A specific sensor type.
- calibrated_sensor : - Definition of a particular sensor as calibrated on a particular vehicle.
- Instance : - Enumeration of all object instance we observed.
- category : - Taxonomy of object categories (e.g. vehicle, human).
- Attribute : - Property of an instance that can change while the category remains the same.
- visibility : - Fraction of pixels visible in all the images collected from 6 different cameras.
- sample_annotation : - An annotated instance of an object within our interest.
- map : - Map data that is stored as binary semantic masks from a top-down view.

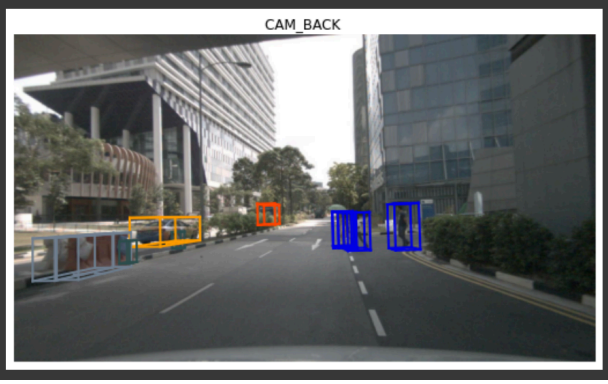
Queries

1. Query1 : -

Image you got from sensor ‘CAM_BACK’ in a particular sample.

Code : -

```
[ ] sample1=nusc.sample[0]
    query2=spark.read.csv('samples/sample1.csv', sep=',',
                          inferSchema=True, header=True)
    query3=query2.select('token','data').where(query2.token==sample1["token"])
    answer=[data1[0] for data1 in query3.select('data').collect()]
    cam="CAM_BACK"
    answer=(eval(answer[0]))
    nusc.render_sample_data(answer[cam])
```



2. Query2 : -

Print the token,first_sample_token into a csv file.Code : -

```
scene=spark.readStream.format("csv").schema(schema1).option("header",True
).option("maxFilesPerTrigger",1).load("scenes")

query2=scene.select('token','first_sample_token')q=query2.writeStream.forma
t("csv").option("path","/query1").option ("checkpointLocation", "/
checkpoint_path").outputMode("append").start()
```

3. Query3 -

Given sample annotation tokens, return what kind of category they belong to.

Code :

```
sample_ann1=nusc.sample_annotation[0]
sample_ann2=nusc.sample_annotation[60]
query3=spark.read.csv('sample_annotation/sample_annotation1.csv', sep=',',
                      inferSchema=True, header=True)
answer1=query3.select('category_name').where(query3.token == sample_ann1['token'])
answer2=query3.select('category_name').where(query3.token == sample_ann2['token'])
result1=[data[0] for data in answer1.select('category_name').collect()]
result2=[data[0] for data in answer2.select('category_name').collect()]
print(result1)
print(result2)
```

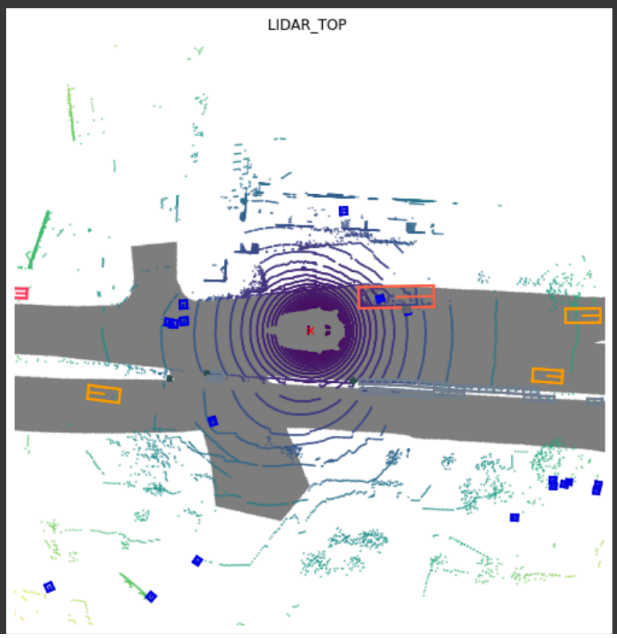
```
['human.pedestrian.adult']
['human.pedestrian.adult']
```

4.Query4-

Given a sample token return data from sensor LIDAR_TOP.

Code :-

```
sample1=nusc.sample[1]
query4=spark.read.csv('samples/sample1.csv', sep=',',
                      inferSchema=True, header=True)
query4_2=query2.select('token','data').where(query4.token==sample1["token"])
answer=[data1[0] for data1 in query4_2.select('data').collect()]
cam="LIDAR_TOP"
answer=(eval(answer[0]))
nusc.render_sample_data(answer[cam])
```



5.Query5 -

Return number of sample_annotations whose category is vehicle.

Code : -

```
query5=spark.read.csv('sample_annotation/sample_annotation1.csv', sep=',',
                      inferSchema=True, header=True)
result1=[data[0] for data in query5.select('category_name').collect()]
answer=0
for i in result1:
  result2=i.split('.')
  if (result2[0]=="vehicle"):
     answer=answer+1
answer
```

```
9648
```

6.Query6 -

Return number of sample_annotations whose category is pedestrian.

Code : -

```
query6=spark.read.csv('sample_annotation/sample_annotation1.csv', sep=',',
                      inferSchema=True, header=True)
result1=[data[0] for data in query6.select('category_name').collect()]
answer=0
for i in result1:
  result2=i.split('.')
  if (result2[1]=="pedestrian"):
     answer=answer+1
answer
```

```
5040
```

7.Query7 : -

Given an sample_annotation,calculate number of instances of that annotation.

Code : -

```
[ ] query7=spark.read.csv('sample_annotation/sample_annotation1.csv', sep=',',
                          inferSchema=True, header=True)
    sample_ann=nusc.sample_annotation[20]
    query7=query7.select('instance_token').where(query7.token ==  sample_ann['token'])
    result1=[data[0] for data in query7.select('instance_token').collect()]
    query7=spark.read.csv('instance/instance1.csv', sep=',',
                          inferSchema=True, header=True)
    query7=query7.select('nbr_annotations').where(query7.token == result1[0]);
    result2=[data[0] for data in query7.select('nbr_annotations').collect()]
    result2[0]

    39
```

8.Query8 : -

Given a sample annotation,return the third instance of that annotation.

Code : -

```
query8=spark.read.csv('sample_annotation/sample_annotation1.csv', sep=',',
                      inferSchema=True, header=True)
sample_ann=nusc.sample_annotation[20]
curr_ann=sample_ann['token']
next_ann=sample_ann['next']
i=2
while(i!=0):
  query8.select('next').where(query8.token == curr_ann)
  result1=[data[0] for data in query8.select('next').collect()]
  curr_ann=result1[0]
  i=i-1
nusc.render_annotation(curr_ann)
```

9.Query9 : -

Given a sample,number of annotations in that sample.

Code : -

```
query9=spark.read.csv('samples/sample1.csv', sep=',',
                      inferSchema=True, header=True)
sample=nusc.sample[15]
query9=query9.select('anns').where(query9.token == sample['token'])
result1=[data[0] for data in query9.select('anns').collect()]
answer=eval(result1[0])
len(answer)
```

```
143
```

10. Query10 : -

Given a sample_annotation,return last instance.

Code : -

```
query10=spark.read.csv('sample_annotation/sample_annotation1.csv', sep=',',
                       inferSchema=True, header=True)
sample_ann=nusc.sample_annotation[20]
query10=query10.select('instance_token').where(query10.token == sample_ann['token'])
result1=[data[0] for data in query10.select('instance_token').collect()]
query10=spark.read.csv('instance/instance1.csv', sep=',',
                       inferSchema=True, header=True)
query10=query10.select("last_annotation_token").where(query10.token == result1[0])
result1=[data[0] for data in query10.select('last_annotation_token').collect()]
nusc.render_annotation(result1[0])
```

11.Query11 : -

Given a sample annotation,list all activities that annotation has done .

Code : -

```
query11=spark.read.csv('sample_annotation/sample_annotation1.csv', sep=',',
                       inferSchema=True, header=True)
sample_ann=nusc.sample_annotation[9]
query11=query11.select('instance_token').where(query11.token == sample_ann['token'])
result1=[data[0] for data in query11.select('instance_token').collect()]
query11=spark.read.csv('instance/instance1.csv', sep=',',
                       inferSchema=True, header=True)
query11=query11.select('category_token').where(query11.token == result1[0])
result1=[data[0] for data in query11.select('category_token').collect()]
query11=spark.read.csv('category/category1.csv', sep=',',
                       inferSchema=True, header=True)
query11=query11.select('description').where(query11.token==result1[0])
result1=[data[0] for data in query11.select('description').collect()]
result1[0]
```

```
'Adult subcategory.'
```

12. Query12 : -

Check whether given annotation is vehicle or not.

Code : -

```
query12=spark.read.csv('sample_annotation/sample_annotation1.csv', sep=',',
                       inferSchema=True, header=True)
sample_ann=nusc.sample_annotation[100]
query12=query12.select('category_name').where(query12.token == sample_ann['token'])
result1=[data[0] for data in query12.select('category_name').collect()]
result1=result1[0].split('.')
if (result1[0]=='vehicle'):
   print(True)
else:
  print(False)
```

```
True
```

13. Query13 : -

Check whether given annotation is pedestrain or not.

Code : -

```
query13=spark.read.csv('sample_annotation/sample_annotation1.csv', sep=',',
                       inferSchema=True, header=True)
sample_ann=nusc.sample_annotation[40]
query13=query13.select('category_name').where(query13.token == sample_ann['token'])
result1=[data[0] for data in query13.select('category_name').collect()]
result1=result1[0].split('.')
if (result1[0]=='human'):
   print(True)
else:
  print(False)
#print(result1[0])
```

```
True
```

14. Query14 : -

How many vehicles are there in an sample?

Code : -

```
query14=spark.read.csv('sample_annotation/sample_annotation1.csv', sep=',',
                       inferSchema=True, header=True)
query14_2=spark.read.csv('samples/sample1.csv', sep=',',
                       inferSchema=True, header=True)
sample=nusc.sample[10]
answer=0
query14_2=query14_2.select('anns').where(query14_2.token == sample['token'])
result1=[data[0] for data in query14_2.select('anns').collect()]
result1=eval(result1[0])
#print(result1)
for ann in result1:
 # print(ann)
  query15=query14.select('category_name').where(query14.token == ann)
  result2=[data[0] for data in query15.select('category_name').collect()]
  result2=result2[0].split('.')
  if (result2[0]=='human'):
    answer=answer+1
answer
```

```
39
```

15. Query15 : -

How many pedestrains are there in an annotation?

Code : -

```
query14=spark.read.csv('sample_annotation/sample_annotation1.csv', sep=',',
                       inferSchema=True, header=True)
query14_2=spark.read.csv('samples/sample1.csv', sep=',',
                       inferSchema=True, header=True)
sample=nusc.sample[10]
answer=0
query14_2=query14_2.select('anns').where(query14_2.token == sample['token'])
result1=[data[0] for data in query14_2.select('anns').collect()]
result1=eval(result1[0])
#print(result1)
for ann in result1:
 # print(ann)
  query15=query14.select('category_name').where(query14.token == ann)
  result2=[data[0] for data in query15.select('category_name').collect()]
  result2=result2[0].split('.')
  if (result2[0]=='vehicle'):
    answer=answer+1
answer
```

```
22
```

16. Query16 : -

number of cycle sample annotations who has a rider ?

Code : -

```
query16=spark.read.csv('sample_annotation/sample_annotation1.csv', sep=',',
                       inferSchema=True, header=True)


result1=[data[0] for data in query16.select('attribute_tokens').collect()]
query16_2=spark.read.csv('attribute/attribute1.csv', sep=',',
                       inferSchema=True, header=True)

result2=[]
for i in result1:
 result2.append(eval(i))
result3=[]
for i in result2:
  if (len(i)==1):
   result3.append(i[0])
#print(result3)
answer=0

for cat in result3:
  query16_3=query16_2.select('name').where(query16_2.token==cat)
  result2=[data[0] for data in query16_3.select('name').collect()]
  #print(result2[0])
  if (result2[0]=='cycle.with_rider'):
     answer=answer+1
answer
```

17 .Query17

Number of instances who are adult and child?

Code : -

```
[ ]  query17=spark.read.csv('instance/instance1.csv', sep=',',
                            inferSchema=True, header=True)
     query17_2=spark.read.csv('category/category1.csv', sep=',',
                            inferSchema=True, header=True)
     result1=[data[0] for data in query17.select('category_token').collect()]
     adult=0
     child=0
     for cat in result1:
         result2=[data[0] for data in query17_2.select('name').collect()]
        # print(result2[0])
         if (result2[0]=='human.pedestrian.adult'):
           adult=adult+1
         if (result2[0]=='human.pedestrian.child'):
           child=child+1
     print('adult:'+str(adult))
     print('child:'+ str(child))
```

```
adult:911
child:0
```

18. Query18 : -

Given an instance, return description of that instance.

Code : -

```
[ ] query18=spark.read.csv('instance/instance1.csv', sep=',',
                           inferSchema=True, header=True)
    query18_2=spark.read.csv('category/category1.csv', sep=',',
                           inferSchema=True, header=True)
    inst=nusc.instance[2]
    query18=query18.select('category_token').where(query18.token == inst['token'])
    result1=[data[0] for data in query18.select('category_token').collect()]
    query18=query18_2.select('description').where(query18_2.token==result1[0])
    result2=[data[0] for data in query18.select('description').collect()]
    result2[0]

    'Vehicle designed primarily for personal use, e.g. sedans, hatch-backs, wagons, vans, mini-vans, SUVs and jeeps. If the vehicle is designed to carry more
    than 10 people use vehicle.bus. If it is primarily designed to haul cargo use vehicle.truck. '
```

19. Query19 : -

Given sample annotation, return visibility.

Code : -

```
query19=spark.read.csv('sample_annotation/sample_annotation1.csv', sep=',',
                       inferSchema=True, header=True)
sample_ann=nusc.sample_annotation[5]
query19=query19.select('visibility_token').where(query19.token == sample_ann['token'])
result1=[data[0] for data in query19.select('visibility_token').collect()]
result1[0]

1
```

20.Query20 : -

Return number of sample_annotations whose visibility is less than 60%.

Code : -

```
query19=spark.read.csv('sample_annotation/sample_annotation1.csv', sep=',',
                       inferSchema=True, header=True)
result1=[data[0] for data in query19.select('visibility_token').collect()]
answer=0
for i in result1:
  if ((int(i))<=2):
     answer=answer+1
answer
```

```
7116
```

Spark streaming

Structured streaming

Structured Streaming is built on top of SparkSQL engine of Apache Spark which will deal with running the stream as the data continues to receive. Just like the other engines of Spark, it is scalable as well as it is fault-tolerant. Structured Streaming enhances Spark DataFrame APIs with streaming features.Structured Streaming also ensures recovery of any fault as soon as possible with the help of checkpoints and caching options. In summary, Structured Streaming is a scalable, fault-tolerant and nearly instant operations.

Structured Streaming queries are processed using a *micro-batch processing* engine, which processes data streams as a series of small batch jobs thereby achieving end-to-end latencies as low as 100 milliseconds and exactly-once fault-tolerance guarantees. However, since Spark 2.3, new low-latency processing mode called Continuous Processing has been introduced, which can achieve end-to-end latencies as low as 1 millisecond with at-least-once guarantees. Without changing the Dataset/DataFrame operations in your queries, you will be able to choose the mode based on your application requirements.

So in this case I have used Python.I have implemented 12 queries using spark streaming API's.

Queries with streaming

Query1 : -

We are loading sample_annotation csv files in batches like in batch streaming using readStream and writeStream API's and calculating what is the count of each category name in the dataset(example: let's say there are 300 sample_annotations belonging to pedestrian.adult).

Code :

```
from pyspark.sql.functions import *
from pyspark.sql.types import *
schema1=StructType([StructField('token',StringType(),True),StructField('sample_token',StringType(),True),StructField('instance_token',StringType(),True
scene=spark.readStream.format("csv").schema(schema1).option("header",True).option("maxFilesPerTrigger",1).load("sample_annotation")
query22=scene.groupBy("category_name").count()
q=query22.writeStream.queryName("query22_6").format("memory").outputMode("complete").start()
for i in range(0,5):
  _df=spark.sql("SELECT * FROM query22_6")
  _df.show()
  time.sleep(2)
```

Query2: -

We are loading sample_annotation csv files in batches like in batch streaming using readStream API and calculating unique category_name dataset contains.

Code

```
+ Code  + Text

schema2=StructType([StructField('token',StringType(),True),StructField('sample_token',StringType(),True),StructField('instance_token',StringType
scene=spark.readStream.format("csv").schema(schema2).option("header",True).option("maxFilesPerTrigger",1).load("sample_annotation")
query23=scene.select("category_name")
q=query23.writeStream.queryName("query22_7").format("memory").outputMode("append").start()
for i in range(0,5):
  _df=spark.sql("SELECT DISTINCT(category_name) FROM query22_7")
  _df.show()
  time.sleep(2)
```

Query3 : -

We are loading sample_annotation csv files in batches like in batch streaming using readStream API and calculating how many sample annotations which belongs to category human.pedestrian.child.

Code : -

Query4 : -

We are loading sample_annotation csv files in batches like in batch streaming using readStream API and calculating number of samples in whole dataset

Code : -

Query5 : -

We are loading sample_data csv files in batches like in batch streaming using readStream API and calculating count of each channel(like RADAR_FRONT ,CAM_BACK) in the dataset.

Code : -

```
schema5=StructType([StructField('token',StringType(),True),StructField('sample_token',StringType(),True),StructField('ego_pose_token',StringType(),Tru
scene=spark.readStream.format("csv").schema(schema5).option("header",True).option("maxFilesPerTrigger",1).load("sample_data1")
query24=scene.groupBy("channel").count()
q=query24.writeStream.queryName("query22_29").format("memory").outputMode("complete").start()
for i in range(0,5):
  _df=spark.sql("SELECT * FROM query22_29")
  _df.show()
  time.sleep(2)
```

Query6 : -

We are loading sample_data csv files in batches like in batch streaming using readStream API and calculating number of samples belong to channel 'RADAR_FRONT'.

Code : -

```
+ Code + Text
schema6=StructType([StructField('token',StringType(),True),StructField('sample_token',StringType(),True),StructField('ego_pose_token',StringType(),Tr
scene=spark.readStream.format("csv").schema(schema6).option("header",True).option("maxFilesPerTrigger",1).load("sample_data1")
query24=scene.select("channel").where(scene['channel'] == 'RADAR_FRONT')
q=query24.writeStream.queryName("query22_32").format("memory").outputMode("append").start()
for i in range(0,5):
  _df=spark.sql("SELECT COUNT(channel) FROM query22_32")
  _df.show()
  time.sleep(2)
```

Query7 : -

We are loading sample_data csv files in batches like in batch streaming using readStream API and calculating count of each distinct visible token in the dataset.

Code : -

```
schema6=StructType([StructField('token',StringType(),True),StructField('sample_token',StringType(),True),StructField('instance_token',StringType(),True
scene=spark.readStream.format("csv").schema(schema6).option("header",True).option("maxFilesPerTrigger",1).load("sample_annotation")
query24=scene.groupBy("visibility_token").count()
q=query24.writeStream.queryName("query22_33").format("memory").outputMode("complete").start()
for i in range(0,5):
  _df=spark.sql("SELECT * FROM query22_33")
  _df.show()
  time.sleep(2)
```

Query8 : -

We are loading sample_data csv files in batches like in batch streaming using readStream API and calculating number of sample annotations whose visibility_token is 2.

Code : -

```
schema7=StructType([StructField('token',StringType(),True),StructField('sample_token',StringType(),True),StructField('instance_token',StringType(),Tr
scene=spark.readStream.format("csv").schema(schema6).option("header",True).option("maxFilesPerTrigger",1).load("sample_annotation")
query24=scene.select("visibility_token").where(scene['visibility_token']== 2)
q=query24.writeStream.queryName("query22_34").format("memory").outputMode("append").start()
for i in range(0,5):
  _df=spark.sql("SELECT COUNT(visibility_token) FROM query22_34")
  _df.show()
  time.sleep(2)
```

Query9 :-

We are loading instance folder csv files in batches like in batch streaming using readStream API and calculating count of each distinct ‘name’ key in the dataset.

Code : -

```
schema8=StructType([StructField('token',StringType(),True),StructField('name',StringType(),True),StructField('description',StringType(),True)])
scene=spark.readStream.format("csv").schema(schema8).option("header",True).option("maxFilesPerTrigger",1).load("instance")
query24=scene.groupBy("name").count()
q=query24.writeStream.queryName("query22_36").format("memory").outputMode("complete").start()
for i in range(0,5):
  time.sleep(10)
  _df=spark.sql("SELECT * FROM query22_36")
  _df.show()
  time.sleep(10)
```

Query10 : -

We are loading log folder csv files in batches like in batch streaming using readStream API and calculating all distinct locations data collected from in the dataset.

Code : -

```
schema9=StructType([StructField('token',StringType(),True),StructField('logfile',StringType(),True),StructField('vehicle',StringType(),True),StructFi
scene=spark.readStream.format("csv").schema(schema9).option("header",True).option("maxFilesPerTrigger",1).load("log")
query24=scene.select("location")
q=query24.writeStream.queryName("query22_38").format("memory").outputMode("append").start()
for i in range(0,5):
  _df=spark.sql("SELECT DISTINCT(location) FROM query22_38")
  _df.show()
  time.sleep(10)
```

Query11 : -

We are loading scenes folder csv files in batches like in batch streaming using readStream API and calculating average number of samples a scene contains.

Code : -

```
schema10=StructType([StructField('token',StringType(),True),StructField('log_token',StringType(),True),StructField('nbr_samples',StringType(),True),St
scene=spark.readStream.format("csv").schema(schema10).option("header",True).option("maxFilesPerTrigger",1).load("scenes")
query24=scene.select("nbr_samples")
q=query24.writeStream.queryName("query22_39").format("memory").outputMode("append").start()
for i in range(0,5):
  _df=spark.sql("SELECT AVG(nbr_samples) FROM query22_39")
  _df.show()
  time.sleep(10)
```

Query12 : -

We are loading sample_data folder csv files in batches like in batch streaming using readStream API and calculating count of each sensor_modality(like RADAR) in the dataset.

Code : -

```
schema11=StructType([StructField('token',StringType(),True),StructField('sample_token',StringType(),True),StructField('ego_pose_token',StringType(),T
scene=spark.readStream.format("csv").schema(schema11).option("header",True).option("maxFilesPerTrigger",1).load("sample_data1")
query24=scene.groupBy('sensor_modality').count()
q=query24.writeStream.queryName("query22_40").format("memory").outputMode("complete").start()
for i in range(0,5):
  _df=spark.sql("SELECT * FROM query22_40")
  _df.show()
  time.sleep(10)
```

Query13 : -

We are loading sample_data folder csv files in batches like in batch streaming using readStream API and calculating distinct sensor_modality(like RADAR) in the dataset.

Code : -

Code

I have created separate folders for sample, sample_data, sample_annotation Instance, Visibility, scenes, instance, category, attribute, sensor, calibrated_sensor ,ego pose, map and log to store respective data in that folders in the form of csv files.Then I have implemented queries on those files. I have implemented 33 queries for this project out of which I have used streaming API's for 13 queries.

Tools used :

- Pyspark
- Python
- Pyspark streaming API's

Conclusion : -

Nuscenes dataset is a very vast and complex dataset .It contains data of 1000 scenes .But only small portion of the data is released.They released data of 10 scenes.So in this project I explored only 10 scenes of nuscenes dataset and implemented queries on that.

Which category this project belongs to?

I would say this project is a very good project because of the dataset and vast content and the structure present in the dataset . It is a complex dataset which is used in autonomous vehicles .Querying on this dataset is a complex task.

Efforts I have put in?

As it is a large and complex dataset.It took me a lot of time to understand the structure of the dataset.After that in order to apply queries using pyspark,I cannot directly apply queries on this dataset.For that purpose,I have loaded every building block data into csv files.Each building block has one folder.After that I have written 33 queries using pyspark and streaming API's. I have thought about each query a lot before writing the code for it.I have implemented 20 queries without using streaming API and 13 queries using streaming API.Total I have written around 700 lines of code for this project.

GitHub Repository

https://github.com/maheedhar1208/SDS-PROJECT