ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503

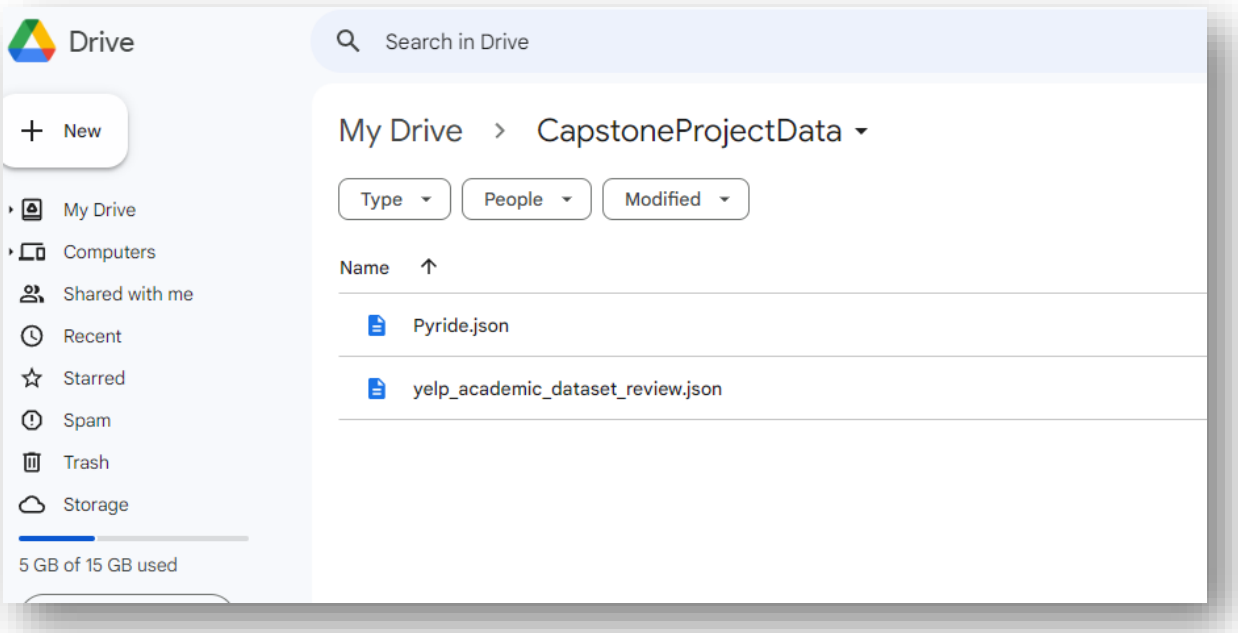
Data Requirement:-

Prior to commencing the capstone project, it is essential to load the necessary data for the Kafka implementation and Pyspark streaming. Given data for the project is stored in the Google Drive repository and after mounting the drive data is loaded in google colab.

Mounting Google Drive

```
[3] # Check if Google Drive is already mounted
    if not os.path.exists('/content/drive'):
        # Mount Google Drive if it's not already mounted
        drive.mount('/content/drive')
    else:
        print("Google Drive is already mounted.")
```

```
Google Drive is already mounted.
```

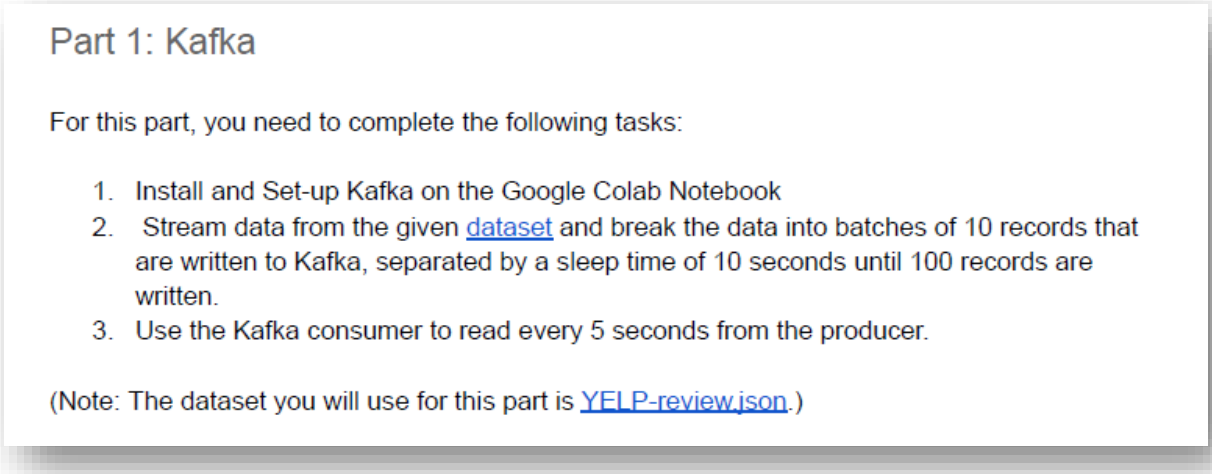


ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503



Part 1: Kafka

For this part, you need to complete the following tasks:

1. Install and Set-up Kafka on the Google Colab Notebook
2. Stream data from the given dataset and break the data into batches of 10 records that are written to Kafka, separated by a sleep time of 10 seconds until 100 records are written.
3. Use the Kafka consumer to read every 5 seconds from the producer.

(Note: The dataset you will use for this part is YELP-review.json.)

Brief Kafka Architecture Explanation:

Kafka is a distributed streaming platform that provides a robust and scalable architecture for building real-time data pipelines and streaming applications. The core components of Kafka include producers, brokers, topics, partitions, and consumers, which work together to create a highly efficient and fault-tolerant messaging system.

Here's an overview of the Kafka producer-consumer architecture:

- **Producers:** Producers are applications or components that generate and publish data to Kafka topics. They send records (messages) to Kafka brokers, which act as intermediaries between producers and consumers. Producers can be various data sources, such as web servers, sensors, log files, or any application that generates data.
- **Brokers:** Kafka brokers are servers responsible for handling incoming messages from producers and serving them to consumers. They are the core components of the Kafka cluster and maintain the publish-subscribe model. Brokers store the incoming records in persistent storage on disk.
- **Topics:** Topics are logical channels or categories to which records are published. They act as message queues where producers write data, and consumers read from them. Each topic can have multiple partitions to store the data, and they are replicated across different brokers for fault tolerance.
- **Partitions:** Topics are divided into partitions to distribute the load and provide parallelism. Each partition is an ordered and immutable sequence of records. Producers can specify a key while sending a record, which helps determine the partition to which the record will be written.
- **Consumers:** Consumers are applications or components that read and process data from Kafka topics. They subscribe to one or more topics and consume records from

the partitions within those topics. Consumers can be part of a consumer group, which enables them to share the load and work in parallel.

- **Consumer Groups:** Consumer groups are logical sets of consumers that work together to consume data from Kafka topics. Each consumer group is assigned to one or more partitions of a topic. Each partition can be consumed by only one consumer from a consumer group at any given time, which allows for parallel processing of messages.
- **Offset Management:** Kafka keeps track of the position of each consumer in each partition using offsets. Offsets represent the position of the last record consumed by a consumer in a partition. It allows consumers to pick up from where they left off in case of failure or rebalancing.
- **ZooKeeper (Optional, for older versions of Kafka):** In older versions of Kafka, ZooKeeper was used for managing cluster state, but in newer versions, Kafka now uses its internal metadata management system.

The Kafka producer-consumer architecture offers several advantages, including high scalability, fault tolerance, and real-time data processing capabilities. It has become a popular choice for building data pipelines, event-driven applications, log aggregation systems, and other streaming data solutions.

Explanation of Yelp Dataset :

Yelp review dataset contains below fields,

- "**review_id**": This is a unique identifier for each review.
- "**user_id**": This field represents the ID of the user who wrote the review.
- "**business_id**": This field contains the ID of the business that was reviewed.
- "**stars**": This is the rating the user gave to the business, on a scale from 1 (worst) to 5 (best).
- "**useful**": This is a count of the number of people who found the review useful.
- "**funny**": This is a count of the number of people who found the review funny.
- "**cool**": This is a count of the number of people who found the review cool.
- "**text**": This field contains the actual text of the review.

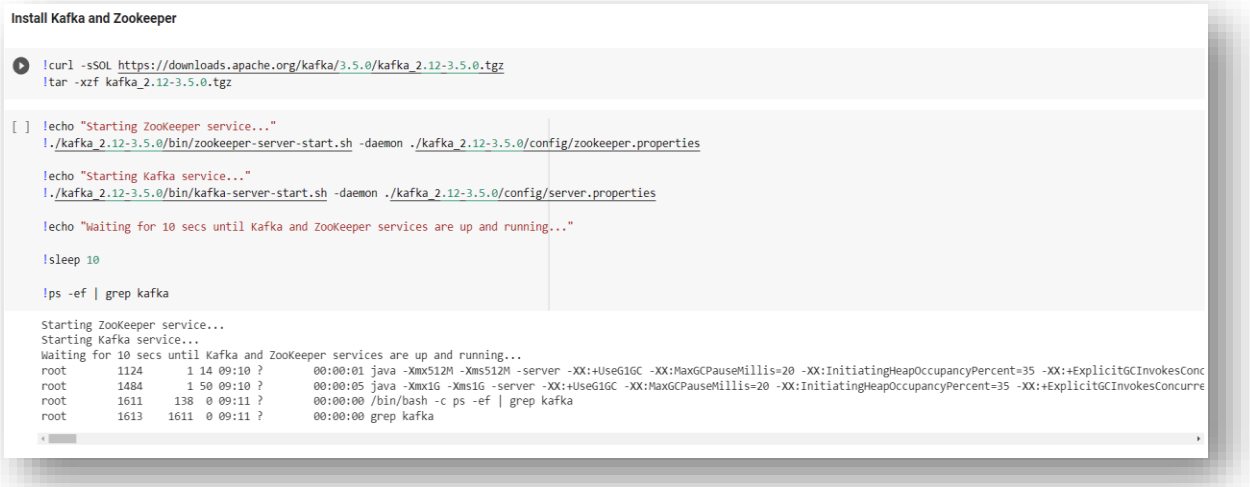
Kafka Installation and Set-up on Google Colab Notebook:

Step 1: As a steps 1, we have installed the Kafka and Zookeeper using following sets of command as mentioned below,

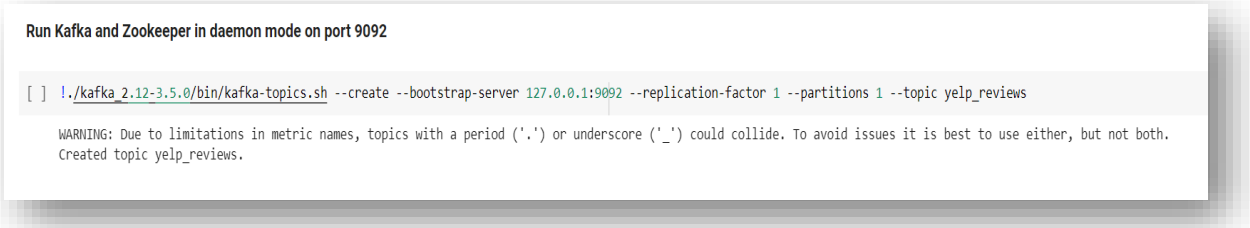
ID5003W:Industrial AI at Scale Laboratory

Capstone Project

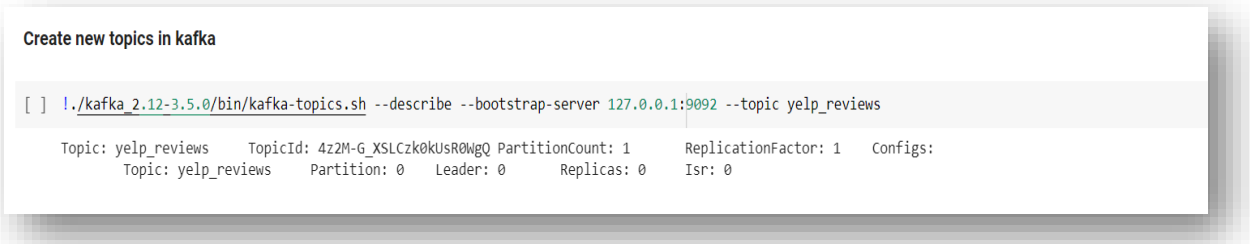
Prepared By: Aloy Banerjee
Roll Number: CH22M503



Step 2: In this step, we have run the Kafka and Zookeeper in port 9092 as mentioned by professor in notebook.



Step 3: In this step, we have created a topic named as **'yelp_reviews'** as our given dataset deals with yelp review data.



Step 4: In further steps we have open jdk and kafka python for performing the given task.

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee
Roll Number: CH22M503

Install OpenJDK

```
[ ]  !echo "Installing OpenJDK 8 JDK..."
     !apt-get install openjdk-8-jdk-headless -qq > /dev/null

     Installing OpenJDK 8 JDK...
```

Install Kafka's client

```
[ ]  !pip install kafka-python

     Collecting kafka-python
       Downloading kafka_python-2.0.2-py2.py3-none-any.whl (246 kB)
          ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 246.5/246.5 kB 5.4 MB/s eta 0:00:00
     Installing collected packages: kafka-python
     Successfully installed kafka-python-2.0.2
```

For performing all the task, we have made a logical segregation in the code for better maintainability and future reusability.

Part 1: Importing Library:

Importing Library

```
import json
import time
import numpy as np
import pandas as pd
import random
from pandas import Timestamp
from datetime import datetime
import threading
from kafka import KafkaProducer, KafkaConsumer
from kafka.errors import KafkaError
```

Explanation and use of the imported library:

- **import json**: This imports the Python JSON module, which allows you to work with JSON data. It provides methods to encode Python objects into JSON strings and decode JSON strings into Python objects.
- **import time**: This imports the Python time module, which provides various time-related functions. It allows you to work with time values, sleep the program for a specific duration, and measure time intervals.

ID5003W:Industrial AI at Scale Laboratory

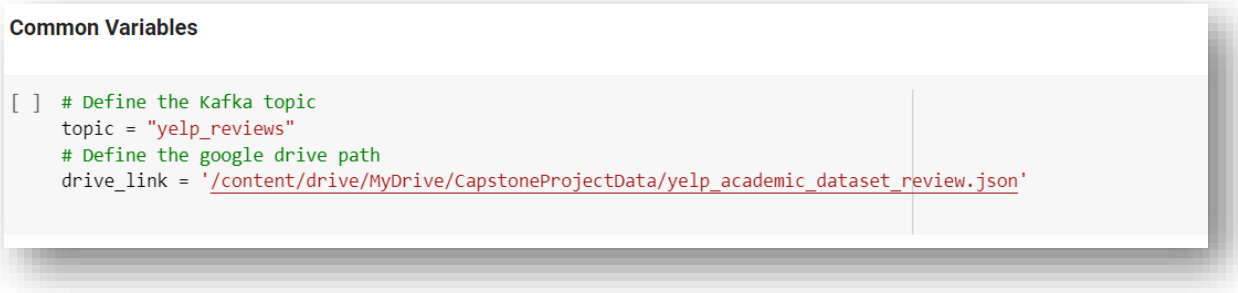
Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503

- **import numpy as np:** This imports the NumPy library and assigns it the alias "np." NumPy is a powerful library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, as well as a collection of mathematical functions to operate on these arrays efficiently.
- **import pandas as pd**: This imports the Pandas library and assigns it the alias "pd." Pandas is a widely used library for data manipulation and analysis. It offers data structures like DataFrame and Series, along with various functions for data cleaning, filtering, grouping, and merging.
- **import random**: This imports Python's built-in random module, which provides functions to generate random numbers, select random elements from a sequence, and shuffle elements in a list.
- **from pandas import Timestamp**: This imports the Timestamp class from the Pandas library. Timestamp is a specific data type in Pandas that represents a single timestamp, similar to a Python datetime object, but with additional functionalities for time series operations.
- **from datetime import datetime**: This imports the datetime class from the Python datetime module. The datetime class allows you to work with date and time values in Python, including formatting and arithmetic operations.
- **import threading**: This imports Python's threading module, which provides support for multi-threading in your programs. Threads allow you to execute multiple tasks concurrently, potentially improving performance and responsiveness.
- **from kafka import KafkaProducer, KafkaConsumer**: This imports the KafkaProducer and KafkaConsumer classes from the Kafka library. Kafka is a distributed streaming platform used for building real-time data pipelines and streaming applications.
- **from kafka.errors import KafkaError**: This imports the KafkaError class from the kafka.errors module. KafkaError represents common errors that can occur when working with Kafka, such as connection errors or message handling issues.

Part 2: Common Variables: Required common variables for storing the topic name used in Kafka and yelp academic review file path in the google drive link is stored in two variables which will be used in further part of the code



ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503

Sequential Kafka Producer and Consumer Call:

Part 3: Common Method:

```python
def read_json_data(file_path, num_lines=None):
    """
    Reads data from a specified JSON file and converts it to a list of dictionaries.
    Each dictionary corresponds to a line in the JSON file. If num_lines is not specified,
    the function will attempt to read all lines in the file.

    Args:
        file_path (str): The path of the JSON file to read.
        num_lines (int, optional): The maximum number of lines to read from the file.
            Defaults to None, which means all lines will be read.

    Returns:
        list: A list of dictionaries representing the JSON data read from the file.

    Raises:
        IOError: If the file cannot be read.
        json.JSONDecodeError: If there is an error decoding the JSON data.
    """
    json_data = []
    try:
        with open(file_path, 'r') as f:
            for i, line in enumerate(f):
                if num_lines is not None and i >= num_lines:
                    break
                record = json.loads(line)
                json_data.append(record)
    except IOError:
        raise IOError(f"Could not read file at path: {file_path}")
    except json.JSONDecodeError:
        raise json.JSONDecodeError(f"Error decoding JSON data in file: {file_path}")

    return json_data
```

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503

```python
def send_to_kafka(producer, topic, data, iteration):
    """
    Sends data to Kafka topic using the specified producer.

    Parameters:
        producer (KafkaProducer): The Kafka producer instance.
        topic (str): The name of the Kafka topic to send data to.
        data (list): A list of dictionaries representing the data to send.
        iteration(int) : Iteration Number

    Returns:
        None
    """
    try:
        print(f'yelp data length : {len(data)}')
        for record in data:
            if 'timestamp' in record and isinstance(record['timestamp'], str):
                record['timestamp'] = datetime.strptime(record['timestamp'], "%Y-%m-%d %H:%M:%S")
            message = json.dumps(record).encode("utf-8")
            print(message)
            producer.send(topic, value=message)
        print(f"Iteration:{iteration} data sent successfully to Kafka.")
    except KafkaError as e:
        print(f"Error sending data to Kafka: {e}")
```

```python
def consume_messages(consumer):
    """
    Consumes messages from Kafka in a separate thread.

    Parameters:
        consumer (KafkaConsumer): The Kafka consumer instance.

    Returns:
        None
    """
    try:
        while True:  # Run indefinitely to keep consuming messages
            for message in consumer:
                record = message.value
                print(record)
                # Perform further processing of the record here if needed
            time.sleep(5)  # Sleep for 5 seconds before reading more messages
    except Exception as e:
        print(f"Error occurred while consuming messages: {e}")
    finally:
        consumer.close()
```

Explanation and the parameter details are mentioned clearly at the top of each common methods in details.

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503

Part 4: Kafka Operation: This section has two parts, in first part we are creating **a Kafka Producer** and pushing the messages from the yelp dataset to Kafka and in next part we are creating **Kafka Consumer** for consuming those messages.

```python
def kafka_producer_call(drive_link, topic):
    """
    Establishes a Kafka producer, loads Yelp data, and sends it to Kafka in batches.

    Args:
        drive_link (str): Path to the Yelp data JSON file.
        topic (str): Kafka topic to which data will be sent.
    """
    # Create a Kafka producer
    producer = KafkaProducer(bootstrap_servers="localhost:9092")
    print('Producer sending data:')
    try:
        # Load the Yelp data and select randomly 1000 records for sending to Kafka
        yelp_reviews_data = read_json_data(drive_link, num_lines=1000)
        num_lines_to_select = 100
        yelp_reviews_data_ops = random.sample(yelp_reviews_data, num_lines_to_select)
        # Send data to Kafka in batches of 10 records with a 10-second sleep time between batches
        batch_size = 10
        for i in range(0, len(yelp_reviews_data_ops), batch_size):
            batch = yelp_reviews_data_ops[i:i + batch_size]
            iteration = (i // batch_size) + 1
            send_to_kafka(producer, topic, batch, iteration)
            print('Producer sleeps for 10 seconds:')
            time.sleep(10)

    except Exception as e:
        print(f"Error: {e}")
    finally:
        producer.close()
```

Calling Kafka Producer

```
[ ]  kafka_producer_call(drive_link,topic)
```

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503

```python
def kafka_consumer_call(topic):
    """
    Establishes a Kafka consumer, starts consuming messages in a separate thread, and handles interruption gracefully.

    Args:
        topic (str): Kafka topic from which messages will be consumed.
    """
    print('Consumer receving data:')

    # Create a Kafka consumer
    consumer = KafkaConsumer(topic, bootstrap_servers='localhost:9092', auto_offset_reset='earliest')

    # Start consuming messages in a separate thread
    consumer_thread = threading.Thread(target=consume_messages, args=(consumer,))
    consumer_thread.start()

    try:
        # Main thread sleeps for 30 seconds to allow message consumption
        time.sleep(30)
    except KeyboardInterrupt:
        print("Received keyboard interrupt. Stopping consumer gracefully.")
    finally:
        # Set a flag to stop the consumer thread gracefully
        consumer.close()
        consumer_thread.join()

    print("Exiting the main thread.")
```

Calling Kafka Consumer

```python
kafka_consumer_call(topic)
```

As per problem statement
‘**10 records that are written to Kafka, separated by a sleep time of 10 seconds until 10 0 records are written**’ has given so we have taken randomly 100 messages from the yelp dataset and sending batch of 10.

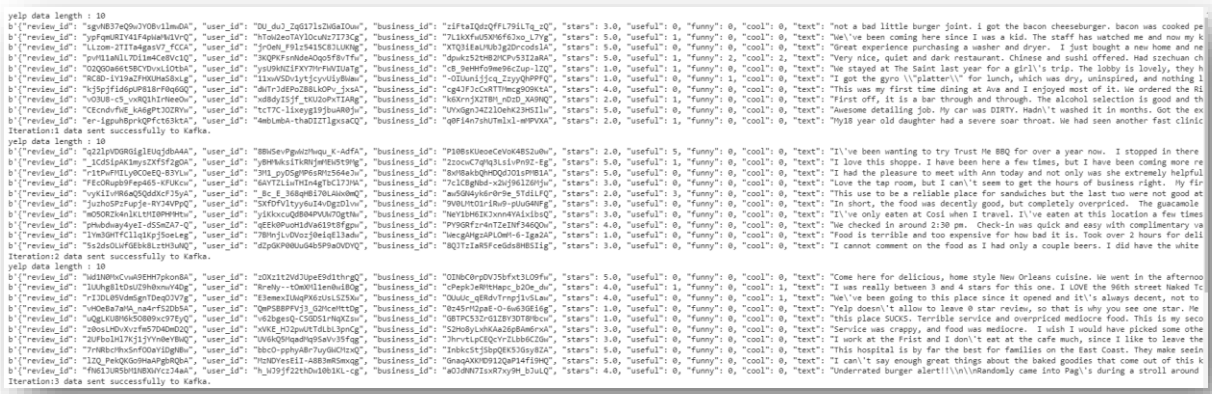
Below are the screenshots of the batch wise operation,

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503

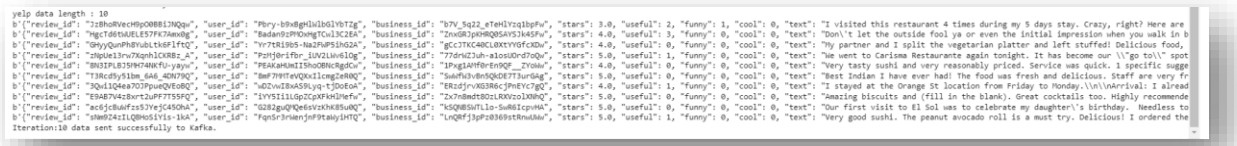


ID5003W:Industrial AI at Scale Laboratory

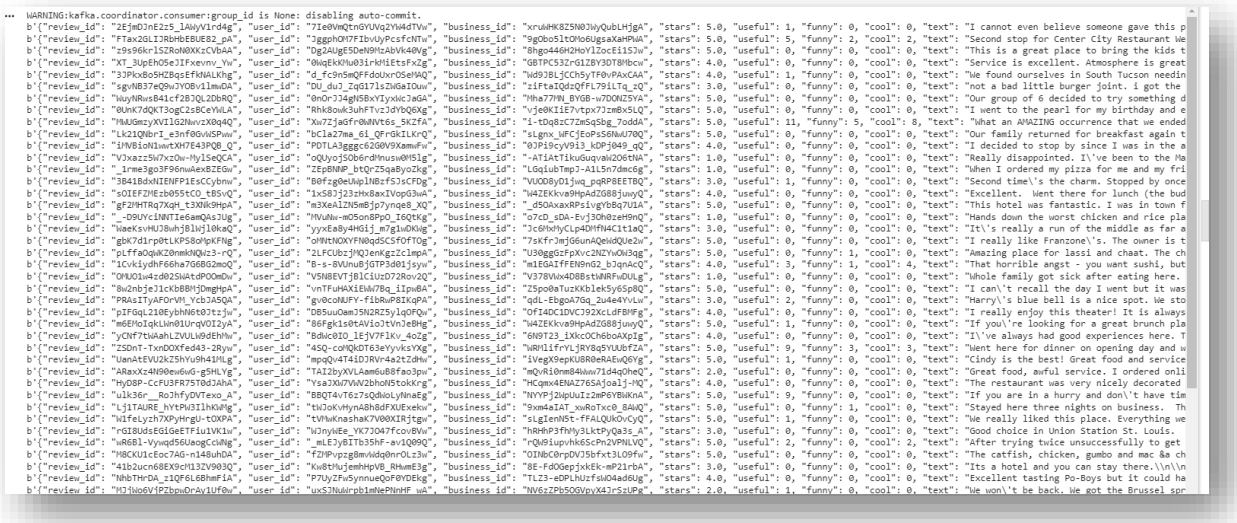
Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503



As per the problem statement after sending the details we are creating Kafka
“Kafka consumer will read in every 5 seconds from the producer”



Note: I have used random selection of 100 record from the whole data which leads different data selection at each run for sending to Kafka producer.

Parallelization of Kafka Producer and Consumer Call: I will be trying to implement the parallelization call of both producer and consumer. In first approach will be trying to implement the same in Google Colab and in next step will do the same using only google colab in a single notebook.

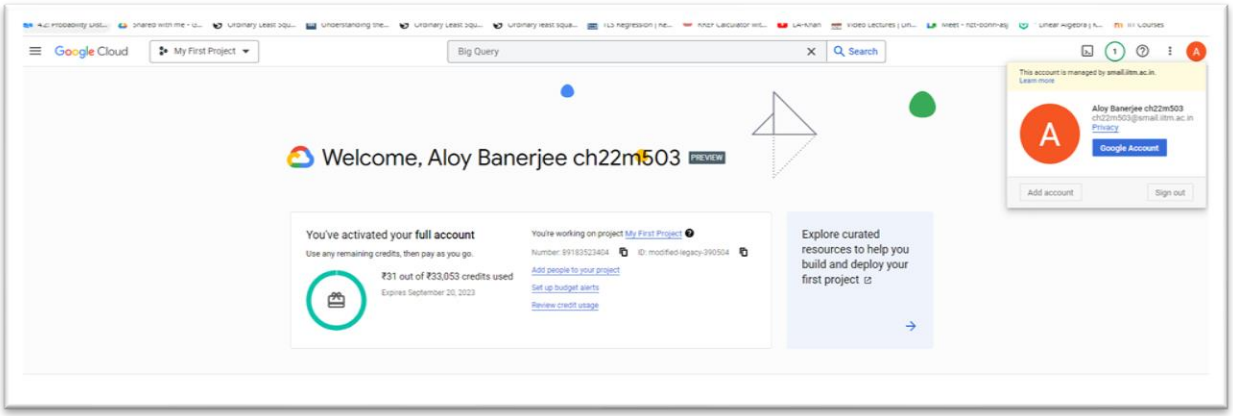
Approach 1: GCP implementation,

<u>Showing the credentials:</u>

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee
Roll Number: CH22M503



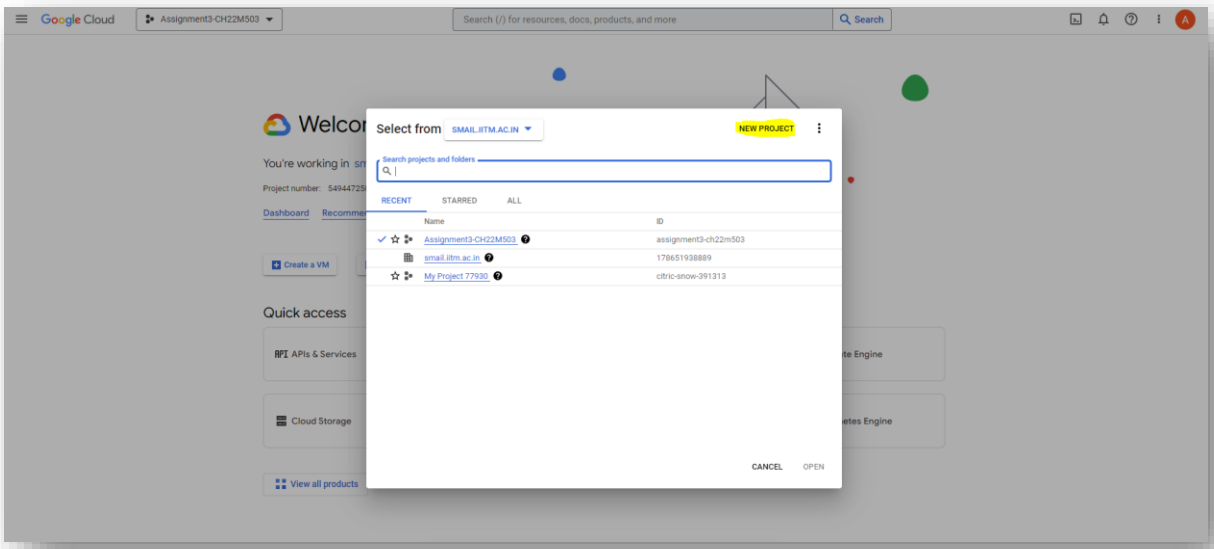
Now I am trying to create the sequential call of the Kafka producer and consumer method which allow the parallelization in the entire process using GCP. Below are the steps followed,

1. As a first step created two separate jupyter notebook named as below to have the code for kafka consumer and kafka producer separately which allows to run the producer and consumer sequentially.
    a. CH22M503_Kafka_Producer.ipynb
    b. CH22M503_Kafka_Consumer.ipynb
2. **Create project:** Click on the create project button it will take you to the project creation page.

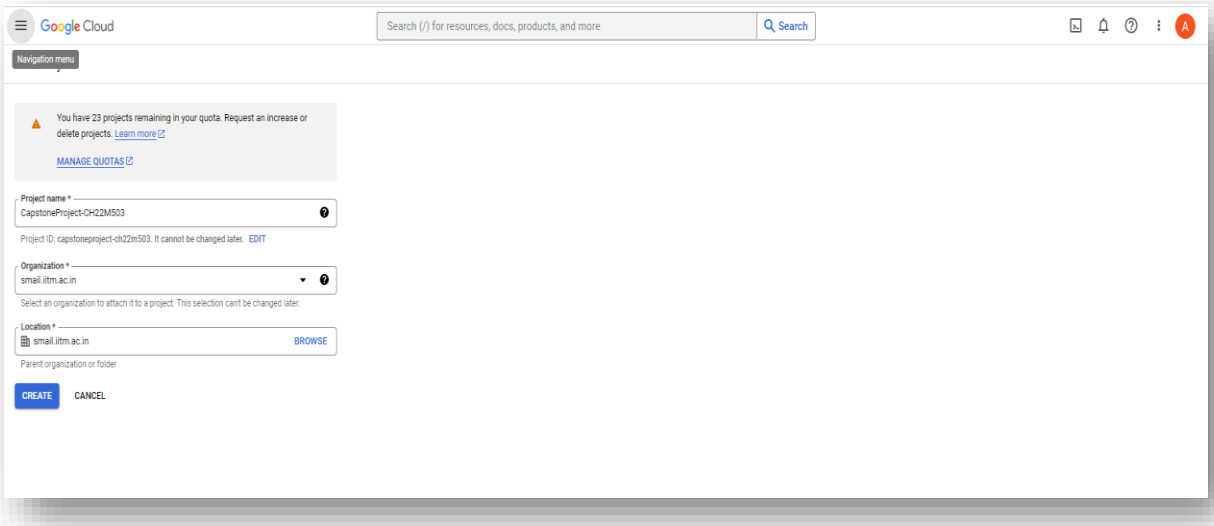
ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee
Roll Number: CH22M503



3. **Project creation page** - Fill out the Project name and click on the create button.

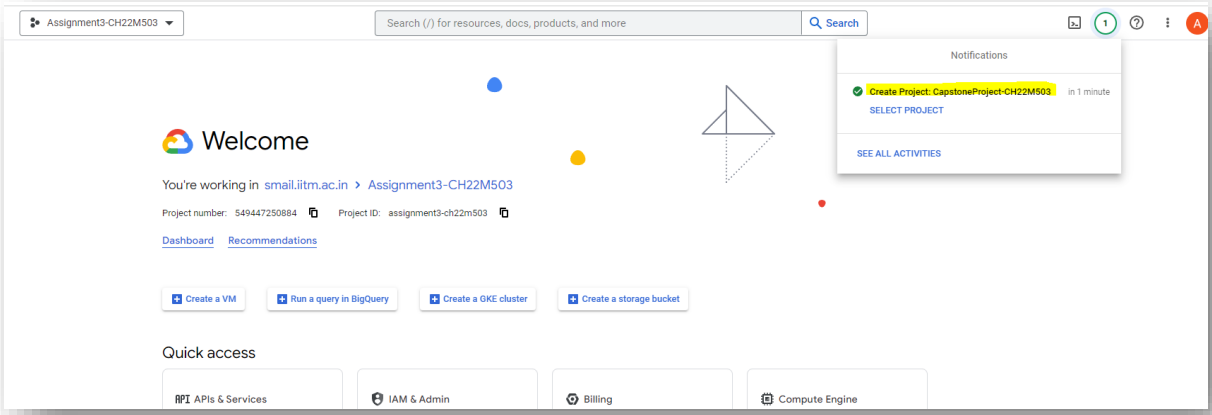


4. If the project gets created successfully then successful notification will appear at the top right corner in the notification section.

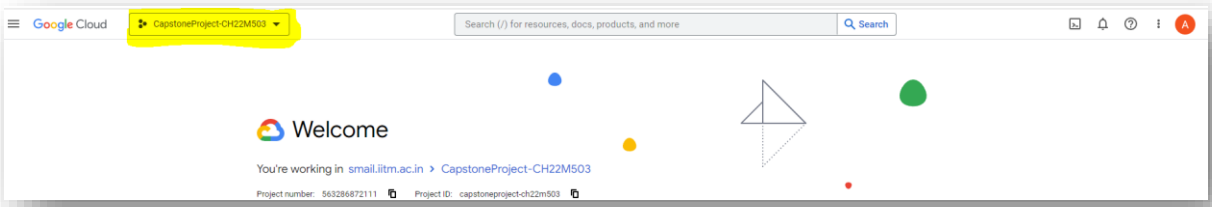
ID5003W:Industrial AI at Scale Laboratory

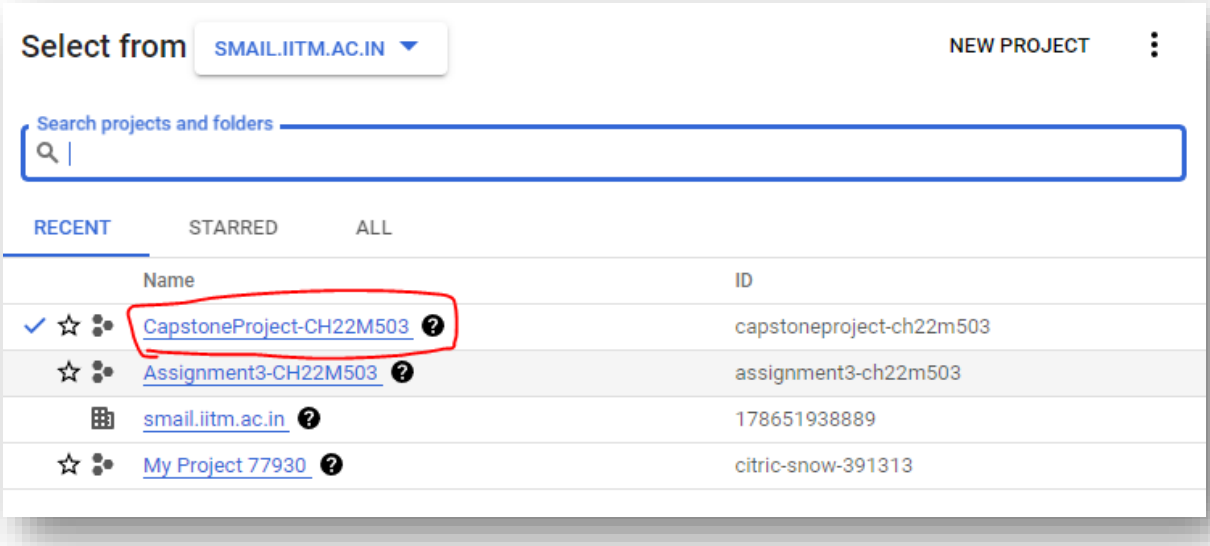
Capstone Project

Prepared By: Aloy Banerjee
Roll Number: CH22M503



5. Choose the project from the top left dropdown and make sure your newly created project appears,



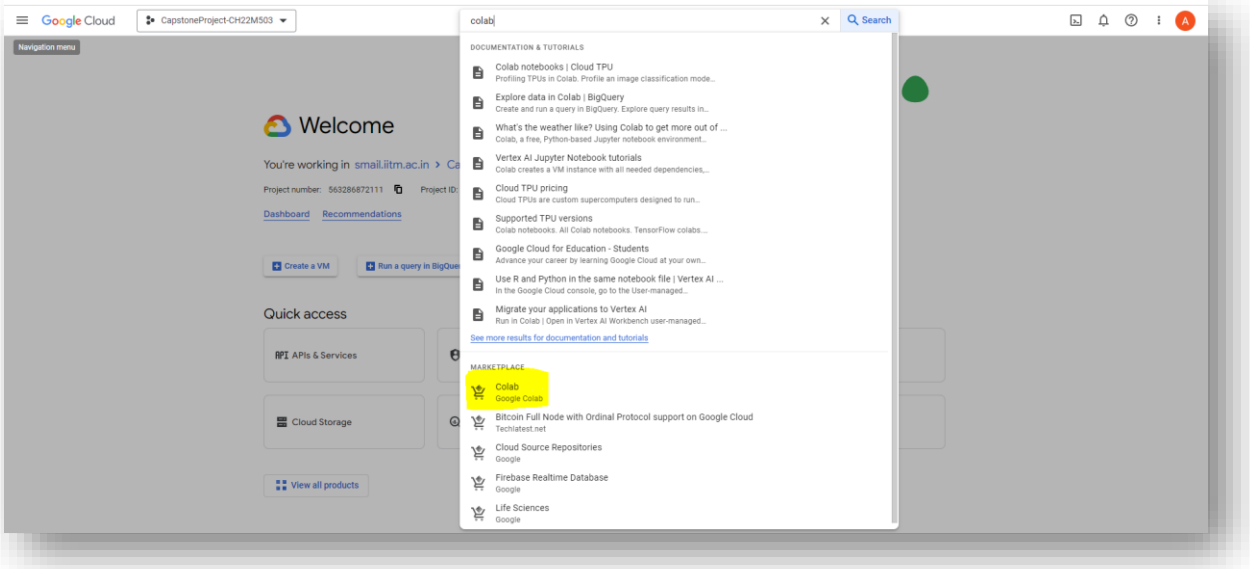


6. Create a Colab GCP VM -

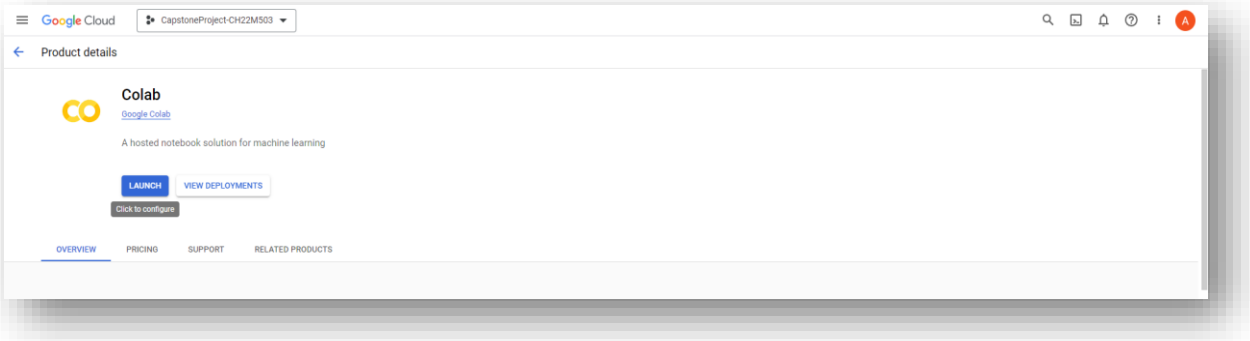
ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee
Roll Number: CH22M503



7. Upon clicking, a dialogue box will appear, presenting a list of required APIs for enabling. Enter each API's name into the search bar individually to activate them sequentially. Keep in mind that attempting to access the APIs directly from the list won't lead you to the correct destination.



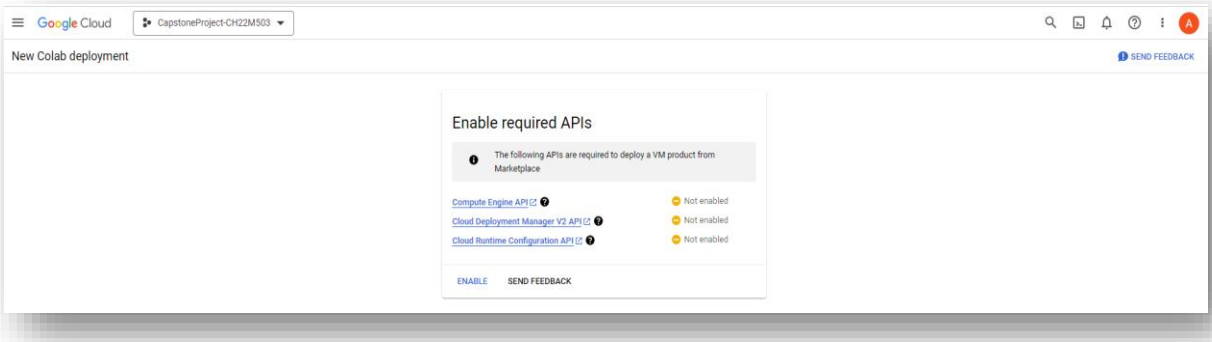
8. Click **'Launch'**

ID5003W:Industrial AI at Scale Laboratory

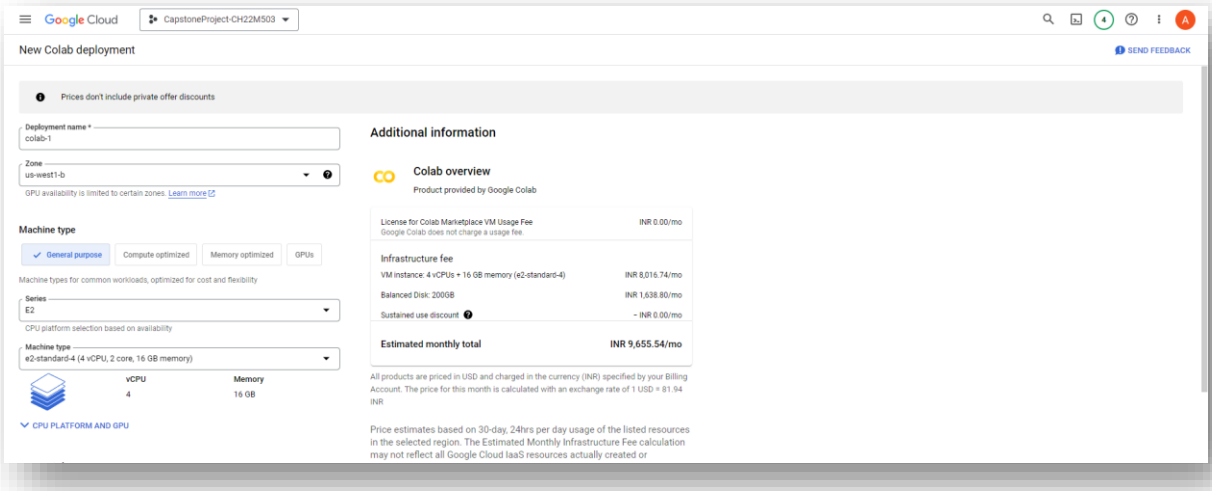
Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503



9. Once the required API's are enabled it will take you through '**New Colab Deployment**' page.

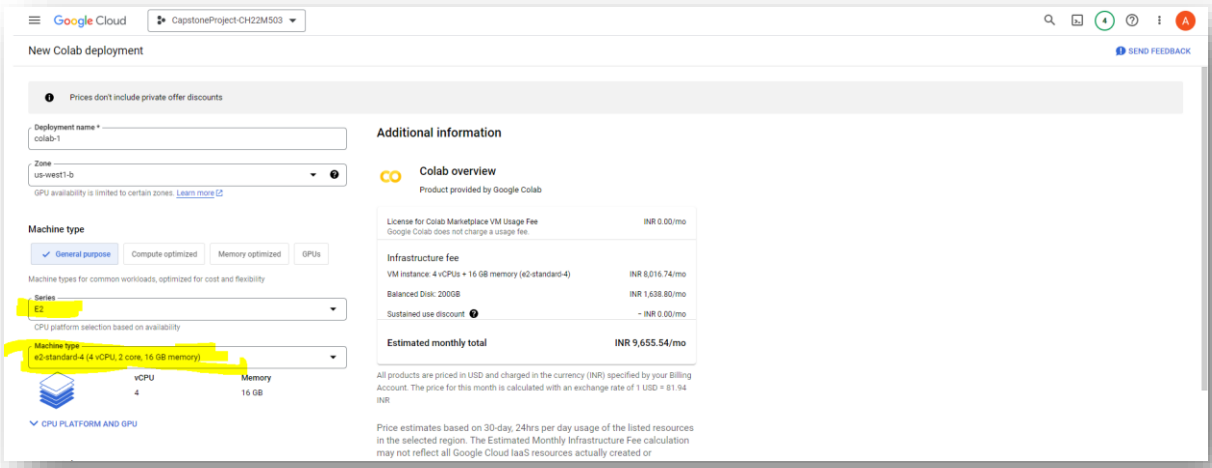


10. Generate a fresh Colab-accessible virtual machine, making certain to complete all the steps marked for attention. Regarding the GPU, kindly omit that section, as GPU usage is unnecessary

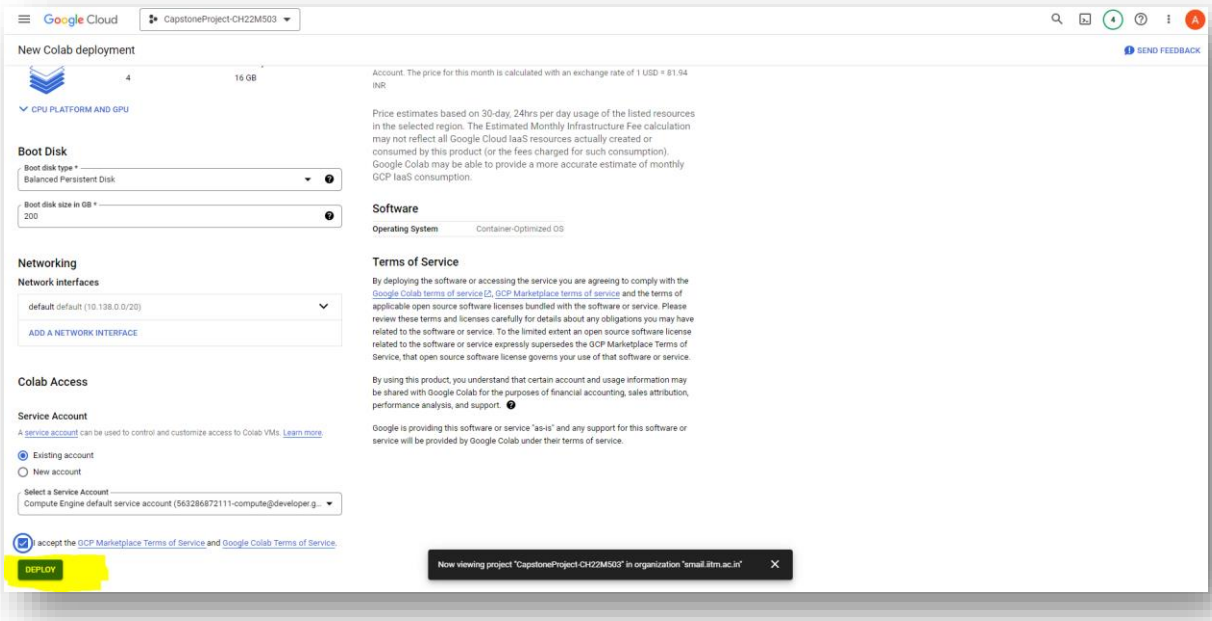
ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee
Roll Number: CH22M503



11. Click on ‘Deploy’ button at the bottom of the page.

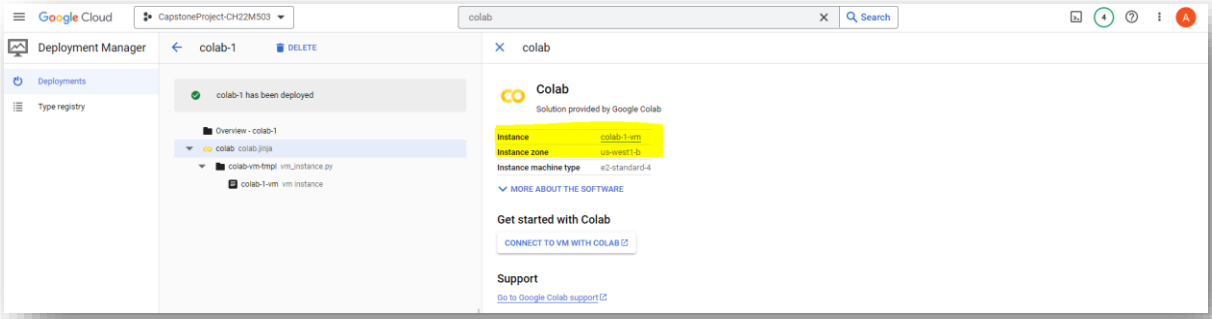


12. Patiently await the completion of your deployment. Jot down the "Instance Zone" and "Instance Name" displayed on the current screen.

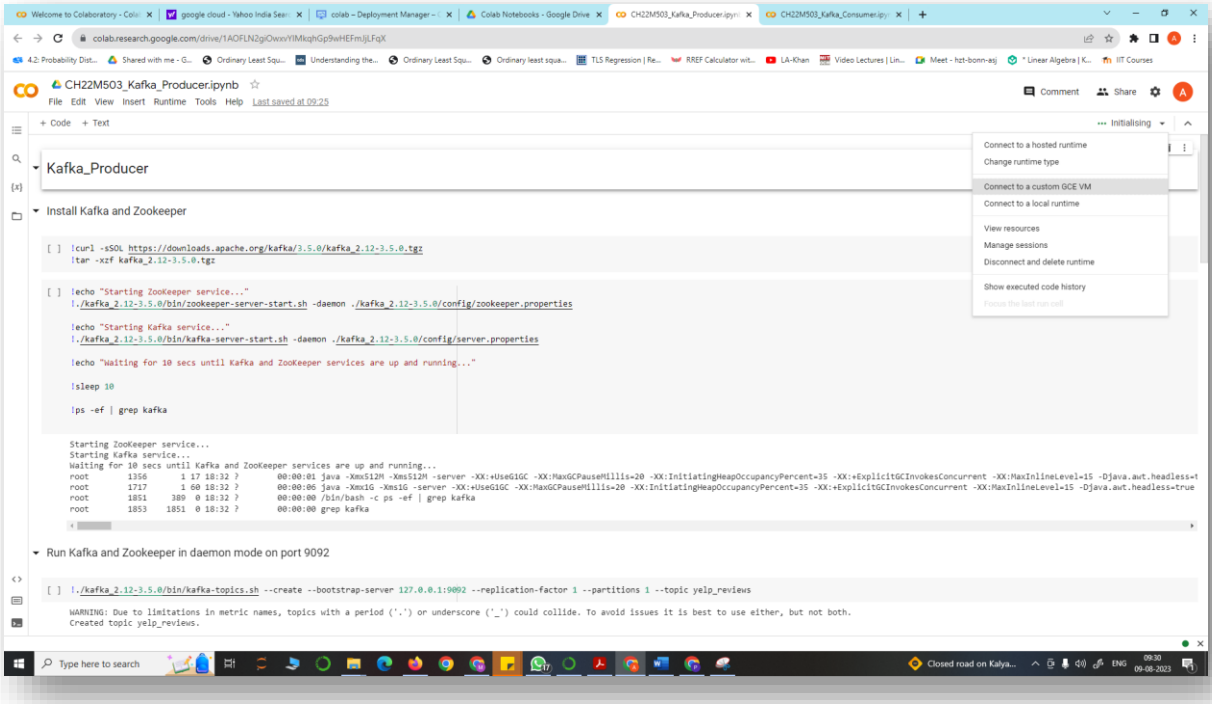
ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee
Roll Number: CH22M503



13. Open your Colab notebook for the Producer module and adhere to the provided instructions for establishing a connection between the notebook and the recently generated virtual machine. Click the "**Connect**" button and subsequently opt for the "**Connect to a custom GCE VM**" choice.

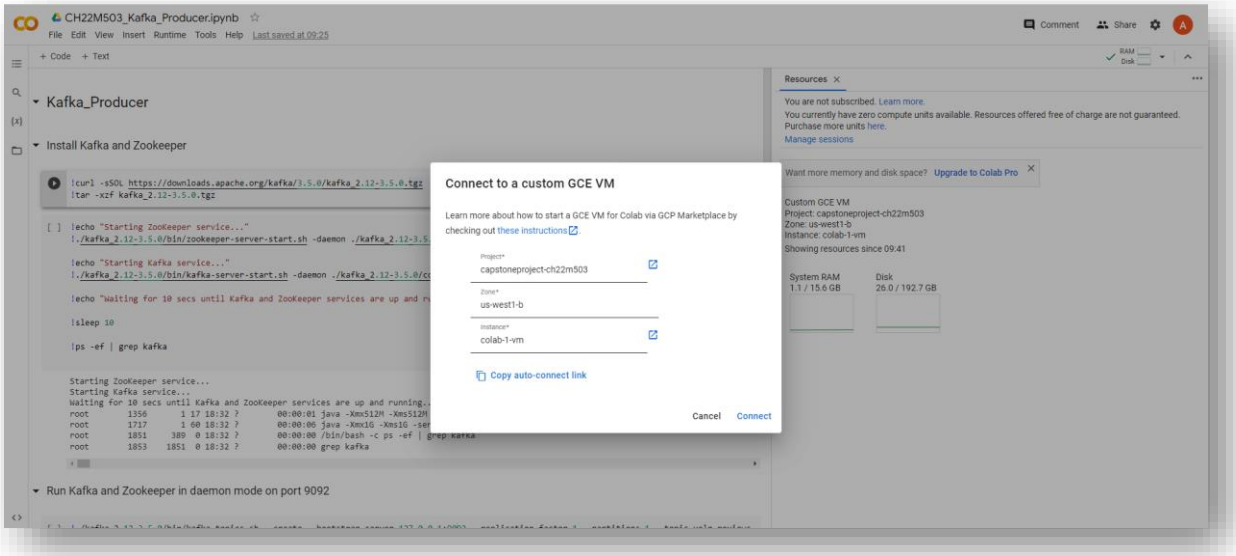


ID5003W:Industrial AI at Scale Laboratory

Capstone Project

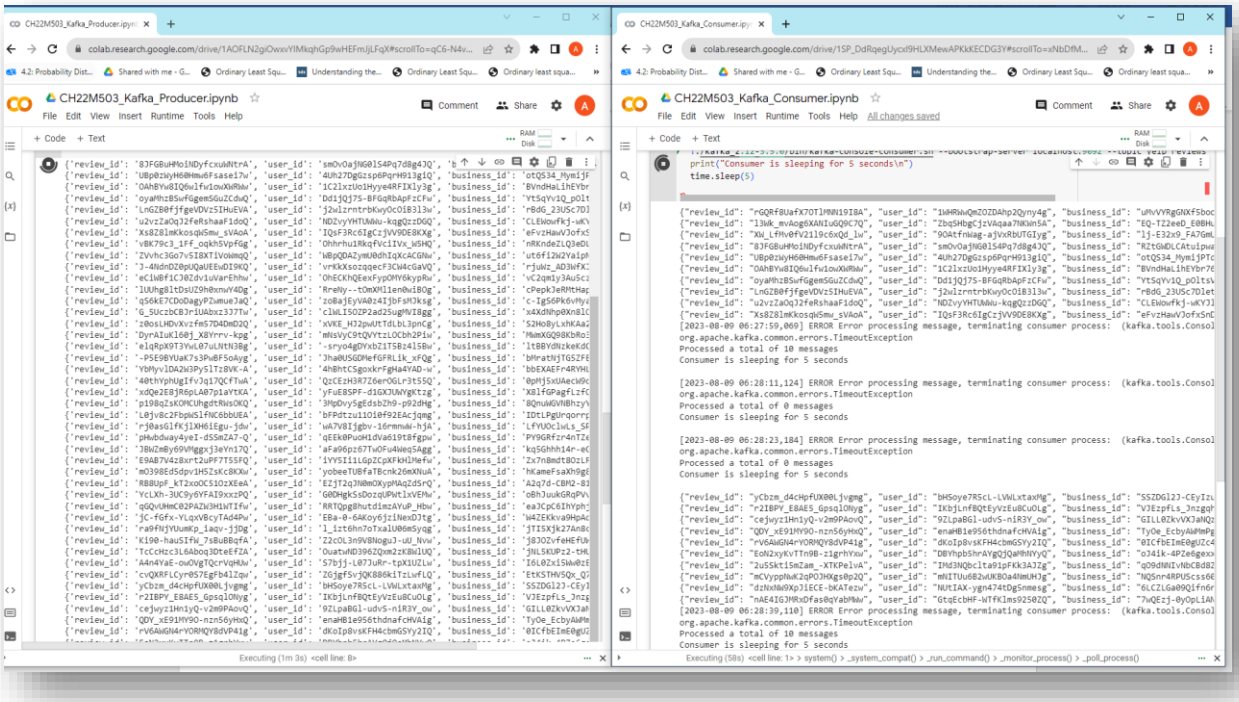
Prepared By: Aloy Banerjee

Roll Number: CH22M503



Note: Follow the same steps for Consumer file also.

14. Run the Producer and Consumer simultaneously and observe the results



ID5003W:Industrial AI at Scale Laboratory

Capstone Project

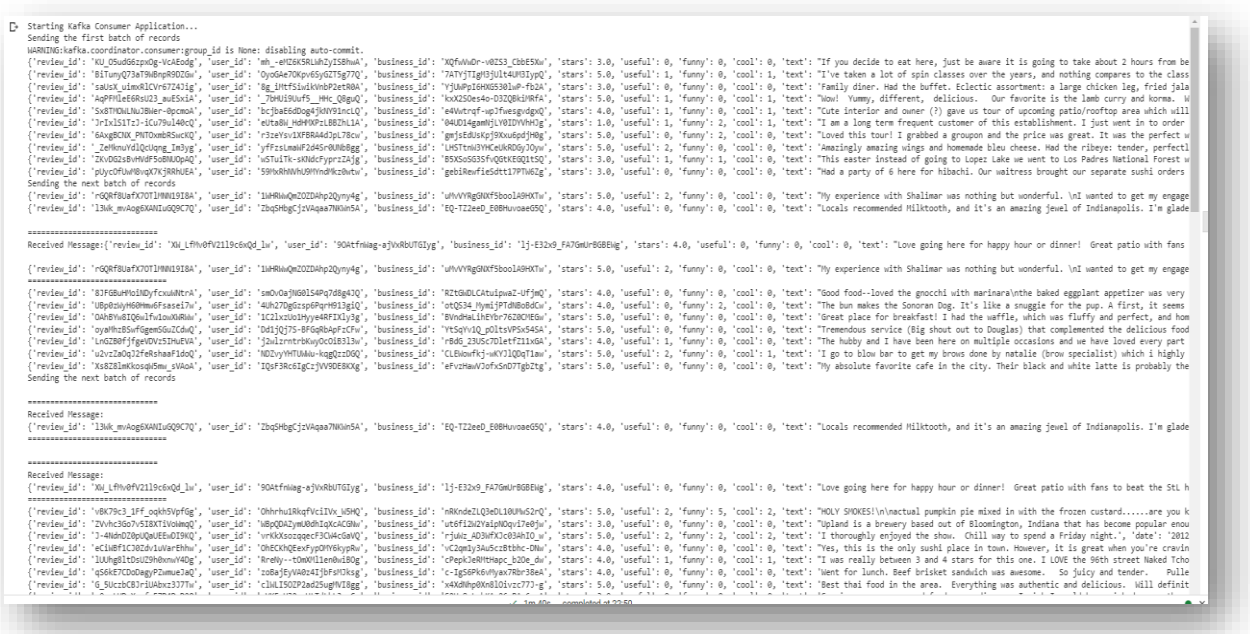
Prepared By: Aloy Banerjee
Roll Number: CH22M503

Approach 2: Google colab notebook implementation,

Step 1: Create two different function for implementing the parallalization of Kafka producer call and Kafka consumer call.

Step 2: Then using multiprocessing approach in python execute both the kafka producer and kafka consumer parallely as per the problem statement.

Output of the parallal call:

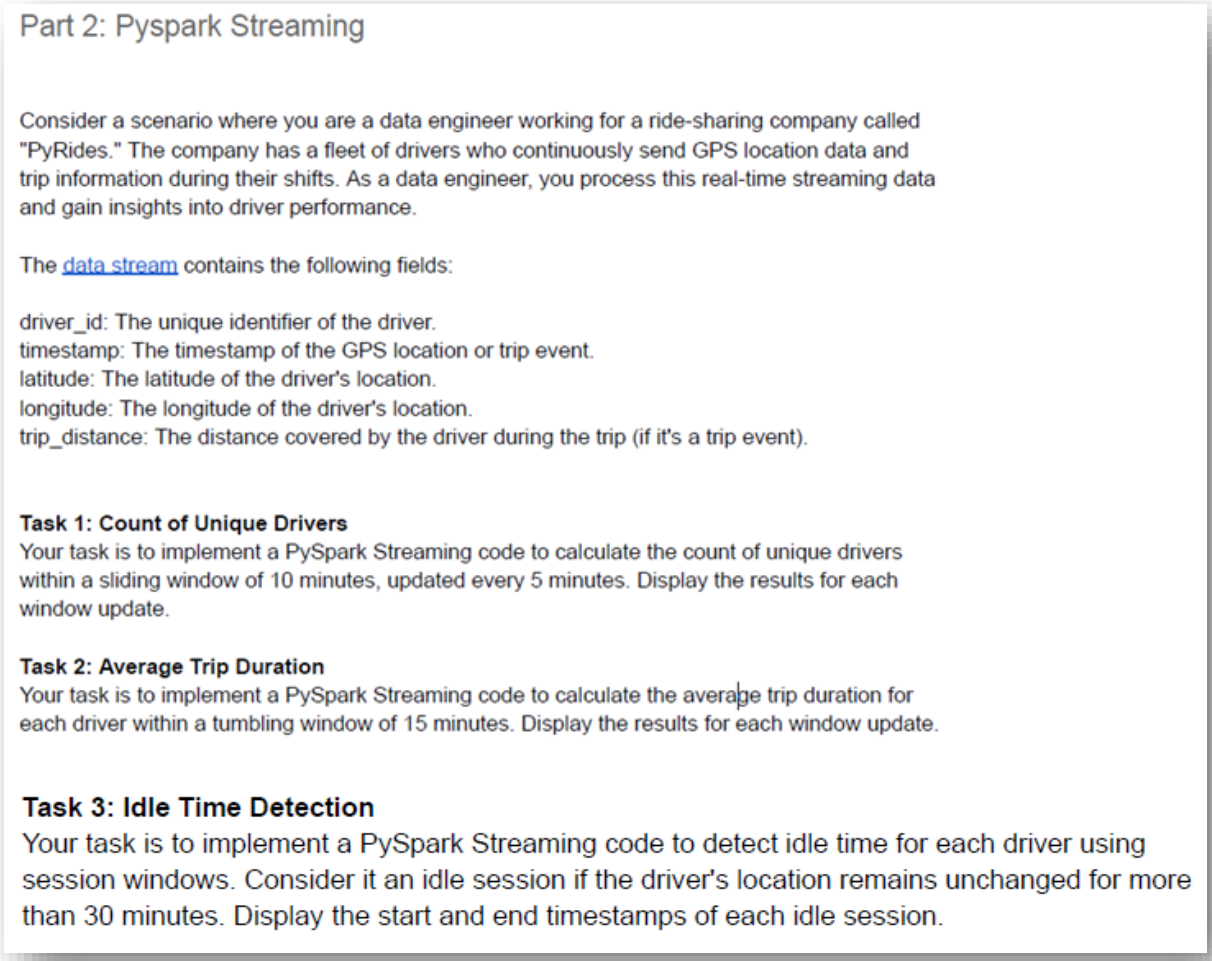


ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503



Part 2: Pyspark Streaming

Consider a scenario where you are a data engineer working for a ride-sharing company called "PyRides." The company has a fleet of drivers who continuously send GPS location data and trip information during their shifts. As a data engineer, you process this real-time streaming data and gain insights into driver performance.

The data stream contains the following fields:

driver_id: The unique identifier of the driver.
timestamp: The timestamp of the GPS location or trip event.
latitude: The latitude of the driver's location.
longitude: The longitude of the driver's location.
trip_distance: The distance covered by the driver during the trip (if it's a trip event).

Task 1: Count of Unique Drivers
Your task is to implement a PySpark Streaming code to calculate the count of unique drivers within a sliding window of 10 minutes, updated every 5 minutes. Display the results for each window update.

Task 2: Average Trip Duration
Your task is to implement a PySpark Streaming code to calculate the average trip duration for each driver within a tumbling window of 15 minutes. Display the results for each window update.

Task 3: Idle Time Detection
Your task is to implement a PySpark Streaming code to detect idle time for each driver using session windows. Consider it an idle session if the driver's location remains unchanged for more than 30 minutes. Display the start and end timestamps of each idle session.

PySpark Streaming is a component of the Apache Spark ecosystem that enables processing and analyzing real-time streaming data. It provides an abstraction for handling continuous data streams, allowing developers to work with streaming data using familiar Spark APIs.

- **Pros of PySpark Streaming:**
  - **Unified API**: PySpark Streaming integrates seamlessly with the Spark ecosystem, allowing users to leverage their existing knowledge of Spark's RDD (Resilient Distributed Dataset) transformations and actions for real-time data processing.
  - **High-level Abstractions**: PySpark Streaming provides high-level abstractions like DStreams (Discretized Streams), which abstract away the complexities

of managing streaming data and allow developers to focus on the business logic.

- **Fault Tolerance**: Like other Spark components, PySpark Streaming offers fault tolerance through lineage information. It can recover lost data due to node failures.
- **Scalability**: PySpark Streaming can easily scale to handle large volumes of streaming data by taking advantage of Spark's distributed computing capabilities.
- **Support for Windowing**: It supports various windowing operations (sliding and tumbling windows) that enable processing data within specific time intervals.
- **Broad Data Source Support**: PySpark Streaming can ingest data from various sources such as Kafka, Flume, HDFS, S3, and more, making it versatile for integrating with different data pipelines.
- **Integration with Batch Processing**: PySpark Streaming seamlessly integrates with batch processing in Spark, allowing users to combine real-time and batch processing for comprehensive data analysis.

❖ **Cons of PySpark Streaming**:

- **Micro-Batching Model**: PySpark Streaming operates using a micro-batching model, where data is processed in small batches. This introduces some latency as data is collected over intervals.
- **Latency**: Due to the micro-batching approach, the latency is higher compared to true real-time processing systems that operate on event-triggered updates.
- **Resource Management Overhead**: Managing a streaming application involves maintaining resources for both the streaming application itself and the Spark cluster, which can lead to increased operational complexity.
- **Complexity for Low-latency Use Cases**: For use cases requiring very low-latency processing, where the delay between data arrival and processing needs to be minimal, PySpark Streaming might not be the ideal choice.
- **Learning Curve**: Although PySpark Streaming abstracts away some of the complexities of real-time data processing, users still need to learn its concepts and APIs, which might require some initial effort.
- **Resource Consumption**: Since Spark is resource-intensive, setting up and maintaining a Spark cluster for streaming can require substantial resources and infrastructure.

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503

Explanation of Pyride data:

This is a JSON data structure that contains data related to a trip or multiple trips made by a driver. Each object (enclosed by {}) in the dataset corresponds to a single trip event. Let's look at the fields for each trip:

- "**driver_id**": This is a unique identifier for a driver. In this case, both trips were completed by driver "D001".
- "**timestamp**": This field represents the time when the trip event was logged. It appears to be in Unix time format, which is the number of seconds that have passed since 00:00:00 Thursday, 1 January 1970, Coordinated Universal Time (UTC), minus leap seconds.
- "**latitude**" and "**longitude**": These fields represent the geographic coordinates of the driver at the time the event was logged. They give the position of the driver on the globe.
- "**trip_distance**": This represents the distance of the trip in some unit, probably miles or kilometers.
- "**event_type**": This field represents the type of event. In this case, it is "Trip", which might suggest that the data logged is related to a trip the driver has taken and “GPS” indicating that the driver is idle or on no trip.

For performing all the task, we have made a logical segregation in the code for better maintainability and future reusability.

Step 1 : **Install Pyspark** : As a pre-requisite package we have installed pyspark before solving the problem statement.

```
[1] !pip install pyspark

    Requirement already satisfied: pyspark in /usr/local/lib/python3.10/dist-packages (3.4.1)
    Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
```

Step 2 : Importing Library:

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503

Importing Libraries

```
import os
import warnings
import threading
from google.colab import drive
from pyspark.sql import SparkSession
from pyspark.sql.functions import * # window, col, countDistinct, avg, lag, when, approx_count_distinct, sum, min, max,from_unixtime, unix_timestamp,expr
from pyspark.sql.types import StructType, StructField, StringType, TimestampType, DoubleType, LongType
from pyspark.sql.window import Window
from pyspark.streaming import StreamingContext
from pyspark import SparkConf, SparkContext
```

Import necessary modules:

- **os**: Provides a way to interact with the operating system, e.g., managing files and directories.
- **warnings:** Allows control over issuing warnings.
- **threading:** Enables multithreading support for running multiple threads simultaneously.
- **google.colab.drive**: Used for accessing and mounting Google Drive in a Colab environment.
- **pyspark.sql.SparkSession:** Used for creating and managing a Spark session.
- **pyspark.sql.functions.*:** Imports all functions from the pyspark.sql.functions module, providing a wide range of SQL functions for DataFrame manipulation.
- **pyspark.sql.types**: Provides classes for defining custom data types in Spark.
- **pyspark.sql.window.Window:** Enables the creation of window specifications for window **functions.**
- **pyspark.streaming.StreamingContext**: Represents the entry point for streaming functionality in Spark.
- **pyspark.SparkConf, pyspark.SparkContext:** Used for configuring and creating a Spark context.

Step 3: Common Variables

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee
Roll Number: CH22M503

Common Variable Declaration

```
[4] # Create a SparkSession
    spark_session = SparkSession.builder.master("local").appName("PyRidesDriverPerformance").config('spark.ui.port', '4050').getOrCreate()
    # File path to the JSON data
    file_path = '/content/drive/MyDrive/CapstoneProjectData/'
    # Define the sliding window duration and slide duration
    window_duration_part1, slide_duration_part1 = "10 minutes", "5 minutes"
    # Define the tumbling window of 15 minutes
    window_duration_part2 = "15 minutes"
    # Define the session window of 30 minutes
    session_gap_duration = "30 minutes"
    # Define a threshold for idle session detection (30 minutes in seconds)
    idle_threshold_seconds = 1800
    # Define the schema for the streaming data
    schema = StructType([
        StructField("driver_id", StringType(), nullable=False),
        StructField("timestamp", TimestampType(), nullable=False),
        StructField("latitude", DoubleType(), nullable=False),
        StructField("longitude", DoubleType(), nullable=False),
        StructField("trip_distance", DoubleType(), nullable=True),
        StructField("event_type", StringType(), nullable=True)  # Set nullable=True if trip_distance can be missing
    ])
```

- **Creating a SparkSession:**
  - A SparkSession is being created using the SparkSession.builder.
  - master("local"): This specifies that Spark should run in local mode.
  - appName("PyRidesDriverPerformance"): Assigns a name to the Spark application.
  - config('spark.ui.port', '4050'): Configures the port for the Spark UI.
  - getOrCreate(): Attempts to reuse an existing SparkSession or create a new one if none exists.
- **File Path and Data Directory:**
  - file_path: Specifies the directory path where the JSON data is located. This should be a directory on the Google Drive.
- **Time Durations:**
  - window_duration_part1 and slide_duration_part1: Define the duration of a sliding window and the slide duration for its movement. In this case, a sliding window of 10 minutes slides every 5 minutes.
  - window_duration_part2: Specifies a tumbling window duration of 15 minutes.

- session_gap_duration: Defines the session window duration, set at 30 minutes.

❖ **Idle Detection Threshold:**

- idle_threshold_seconds: Specifies the threshold for detecting idle sessions, set to 1800 seconds (30 minutes).

❖ **Schema Definition:**

- schema: Defines the schema for the streaming data using StructType and StructField. It includes fields like driver_id, timestamp, latitude, longitude, trip_distance, and event_type

Step 4: Loading Data

❖ As per the problem statement data should be read as a stream and for doing the same, we are reading the data as a stream.

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee
Roll Number: CH22M503

```
[6] # Retrieve data from the "ride_information_using_stream" temporary view using SQL query
    ride_information = spark_session.sql("select * from ride_information_using_stream")
```

```
# Display the data
ride_information.show()
```

```
+---------+-------------------+------------------+-------------------+------------------+----------+
|driver_id|          timestamp|          latitude|          longitude|     trip_distance|event_type|
+---------+-------------------+------------------+-------------------+------------------+----------+
|     null|               null|              null|               null|              null|      null|
|     D001|2023-07-30 05:55:48|37.538849579360246|-121.22748988885533| 3.423076573047031|      Trip|
|     D001|2023-07-30 07:10:08| 37.93476356947273| -121.0215678805398| 5.072880200441588|      Trip|
|     D001|2023-07-30 07:11:01|37.795197133875064|  -121.920222434928|              null|       GPS|
|     D001|2023-07-30 05:20:21| 37.46871545044007|-121.05888928225791|1.5685949947600415|      Trip|
|     D001|2023-07-30 05:27:29| 37.67930257604861| -121.7174686646489|              null|       GPS|
|     D001|2023-07-30 06:20:49| 37.57858135955766|-121.45726820764473|              null|       GPS|
|     D001|2023-07-30 07:06:38| 37.74438722614793|-121.17710081904086|7.7232027230627365|      Trip|
|     D001|2023-07-30 06:18:12|  37.0110278159199|-121.42103203264915|              null|       GPS|
|     D001|2023-07-30 05:34:54|37.536543146725734|-121.99413381326183| 5.553422145437799|      Trip|
|     D001|2023-07-30 05:43:44|37.416567538021916|-121.64854028369182| 4.557593809287299|      Trip|
|     D001|2023-07-30 06:35:15|37.939113113534425|-121.25154744526644|              null|       GPS|
|     D001|2023-07-30 06:24:44| 37.35144953430424| -121.6580016307517| 5.352056091877271|      Trip|
|     D001|2023-07-30 05:44:40| 37.93082254347954|   -121.84577258978|  4.52091123915711|      Trip|
|     D001|2023-07-30 05:23:03|37.824150511617596|-121.68988617388914| 8.046271597547339|      Trip|
|     D001|2023-07-30 07:02:44| 37.08195014659423| -121.4858342989973|   9.0334611014362|      Trip|
|     D001|2023-07-30 07:09:01| 37.54445263452247|-121.46904886667443|              null|       GPS|
|     D001|2023-07-30 06:11:36| 37.64415725449518|-121.79885410528092| 7.661613624439978|      Trip|
|     D001|2023-07-30 05:41:40|37.868529012745064| -121.5179241616071|2.5937936951862506|      Trip|
|     D001|2023-07-30 05:12:56| 37.00833917614987|-121.86966961698501|  8.16513422051636|      Trip|
+---------+-------------------+------------------+-------------------+------------------+----------+
only showing top 20 rows
```

Steps for loading the streaming data:

- Read the data from the given google drive path using read stream method using predefine schema and allowing the multiline readability.
- Once the data is captured as a stream data convert that into pyspart dataframe and start the writestream process with a specific query name.
- Once the process is completed use that query name in next cell to read and view the data.

As per the problem statement there are three tasks need to be performed,

a. **Count the Unique Driver**

b. **Calculate the Average Trip Duration**

c. **Driver's Idle Time Detection**

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee

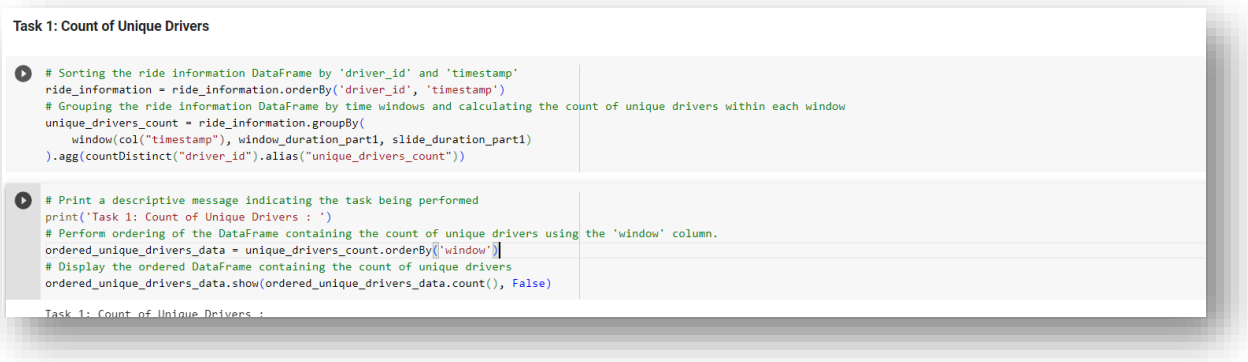
Roll Number: CH22M503

Task 1 :

Approach of the Task 1: Based on my comprehension, the initial step entails gathering information pertaining to all drivers operating within the stipulated timestamp range. Subsequently, it becomes imperative to identify the distinct driver entities among this cohort. My comprehensive evaluation, executed within an Excel framework, has also revealed the same result which I have received in programming.

Screenshots of Task 1 code:

Step 1: Sorting, Grouping and Counting the unique driver's id for a given window



Task 1: Count of Unique Drivers

```
# Sorting the ride information DataFrame by 'driver_id' and 'timestamp'
ride_information = ride_information.orderBy('driver_id', 'timestamp')
# Grouping the ride information DataFrame by time windows and calculating the count of unique drivers within each window
unique_drivers_count = ride_information.groupBy(
    window(col("timestamp"), window_duration_part1, slide_duration_part1)
).agg(countDistinct("driver_id").alias("unique_drivers_count"))
```

```
# Print a descriptive message indicating the task being performed
print('Task 1: Count of Unique Drivers : ')
# Perform ordering of the DataFrame containing the count of unique drivers using the 'window' column.
ordered_unique_drivers_data = unique_drivers_count.orderBy('window')
# Display the ordered DataFrame containing the count of unique drivers
ordered_unique_drivers_data.show(ordered_unique_drivers_data.count(), False)
```

```
Task 1: Count of Unique Drivers :
```

Explanation of Task 1 code:

- **Sorting the DataFrame:**
  - ride_information DataFrame is being sorted in ascending order based on two columns: 'driver_id' and 'timestamp'.
  - This step ensures that the data is organized by driver and timestamp for further processing.
- **Grouping and Counting Unique Drivers:**
  - ride_information DataFrame is grouped by time windows defined by the window_duration_part1 and slide_duration_part1 parameters.
  - Within each time window, the code calculates the count of distinct (unique) drivers using the countDistinct function. The result is aliased as "unique_drivers_count".
- **Printing a Descriptive Message:**

- A message indicating the current task being performed is printed to the console.
- In this case, the message is 'Task 1: Count of Unique Drivers :'.

❖ **Ordering the DataFrame:**

- The DataFrame unique_drivers_count is ordered based on the 'window' column. This is done to prepare the data for display in chronological order.

❖ **Displaying the Ordered DataFrame:**

- The ordered DataFrame ordered_unique_drivers_data is displayed using the show() function.
- The first argument of show() specifies the number of rows to display.
- The second argument, False, specifies that the column values should not be truncated for display.

❖ **Overall, the code performs the following tasks:**

- Sorts the ride information DataFrame.
- Groups the data into time windows and counts the number of unique drivers in each window.
- Prints a descriptive message.
- Orders the result by time window.
- Displays the ordered result, showing the count of unique drivers within each window.

Output of Task 1:

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503

Task 1: Count of Unique Drivers :

window	unique_drivers_count
{2023-07-30 05:05:00, 2023-07-30 05:15:00}	47
{2023-07-30 05:10:00, 2023-07-30 05:20:00}	50
{2023-07-30 05:15:00, 2023-07-30 05:25:00}	50
{2023-07-30 05:20:00, 2023-07-30 05:30:00}	50
{2023-07-30 05:25:00, 2023-07-30 05:35:00}	50
{2023-07-30 05:30:00, 2023-07-30 05:40:00}	50
{2023-07-30 05:35:00, 2023-07-30 05:45:00}	50
{2023-07-30 05:40:00, 2023-07-30 05:50:00}	50
{2023-07-30 05:45:00, 2023-07-30 05:55:00}	49
{2023-07-30 05:50:00, 2023-07-30 06:00:00}	50
{2023-07-30 05:55:00, 2023-07-30 06:05:00}	50
{2023-07-30 06:00:00, 2023-07-30 06:10:00}	50
{2023-07-30 06:05:00, 2023-07-30 06:15:00}	50
{2023-07-30 06:10:00, 2023-07-30 06:20:00}	50
{2023-07-30 06:15:00, 2023-07-30 06:25:00}	50
{2023-07-30 06:20:00, 2023-07-30 06:30:00}	50
{2023-07-30 06:25:00, 2023-07-30 06:35:00}	50
{2023-07-30 06:30:00, 2023-07-30 06:40:00}	50
{2023-07-30 06:35:00, 2023-07-30 06:45:00}	50
{2023-07-30 06:40:00, 2023-07-30 06:50:00}	50
{2023-07-30 06:45:00, 2023-07-30 06:55:00}	50
{2023-07-30 06:50:00, 2023-07-30 07:00:00}	50
{2023-07-30 06:55:00, 2023-07-30 07:05:00}	50
{2023-07-30 07:00:00, 2023-07-30 07:10:00}	50
{2023-07-30 07:05:00, 2023-07-30 07:15:00}	50
{2023-07-30 07:10:00, 2023-07-30 07:20:00}	40

Above table shows for each time duration how many unique drivers are available.

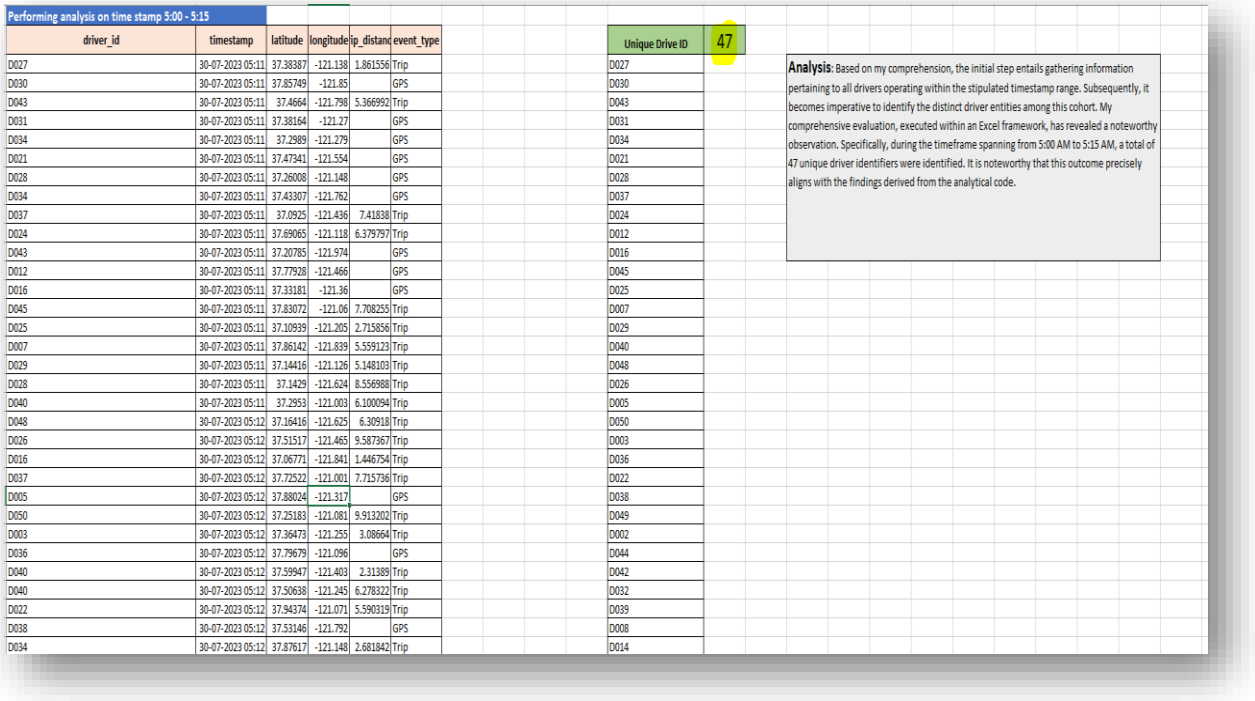
Same analysis has been done in the excel file for a single driver(**D001**) data from the py-ride dataset. Below is the screenshot for the same,

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503



Performing analysis on time stamp 5:00 - 5:15

driver_id	timestamp	latitude	longitude	ip_distanc	event_type
D027	30-07-2023 05:11	37.38387	-121.138	1.861556	Trip
D030	30-07-2023 05:11	37.85749	-121.85		GPS
D043	30-07-2023 05:11	37.4664	-121.798	5.366992	Trip
D031	30-07-2023 05:11	37.38164	-121.27		GPS
D034	30-07-2023 05:11	37.2989	-121.279		GPS
D021	30-07-2023 05:11	37.47341	-121.554		GPS
D028	30-07-2023 05:11	37.26008	-121.148		GPS
D034	30-07-2023 05:11	37.43307	-121.762		GPS
D037	30-07-2023 05:11	37.0925	-121.436	7.41838	Trip
D024	30-07-2023 05:11	37.69065	-121.118	6.379797	Trip
D043	30-07-2023 05:11	37.20785	-121.974		GPS
D012	30-07-2023 05:11	37.77928	-121.466		GPS
D016	30-07-2023 05:11	37.33181	-121.36		GPS
D045	30-07-2023 05:11	37.83072	-121.06	7.708255	Trip
D025	30-07-2023 05:11	37.10939	-121.205	2.715856	Trip
D007	30-07-2023 05:11	37.86142	-121.839	5.559123	Trip
D029	30-07-2023 05:11	37.14416	-121.126	5.148103	Trip
D028	30-07-2023 05:11	37.1429	-121.624	8.556988	Trip
D040	30-07-2023 05:11	37.2953	-121.003	6.100094	Trip
D048	30-07-2023 05:12	37.16416	-121.625	6.30918	Trip
D026	30-07-2023 05:12	37.51517	-121.465	9.587367	Trip
D016	30-07-2023 05:12	37.06771	-121.841	1.446754	Trip
D037	30-07-2023 05:12	37.72522	-121.001	7.715736	Trip
D005	30-07-2023 05:12	37.88024	-121.317		GPS
D050	30-07-2023 05:12	37.25183	-121.081	9.913202	Trip
D003	30-07-2023 05:12	37.36473	-121.255	3.08664	Trip
D036	30-07-2023 05:12	37.79679	-121.096		GPS
D040	30-07-2023 05:12	37.59947	-121.403	2.31389	Trip
D040	30-07-2023 05:12	37.50638	-121.245	6.278322	Trip
D022	30-07-2023 05:12	37.94374	-121.071	5.590319	Trip
D038	30-07-2023 05:12	37.53146	-121.792		GPS
D034	30-07-2023 05:12	37.87617	-121.148	2.681842	Trip

Unique Drive ID	47
D027	
D030	
D043	
D031	
D034	
D021	
D028	
D037	
D024	
D012	
D016	
D045	
D025	
D007	
D029	
D040	
D048	
D026	
D005	
D050	
D003	
D036	
D022	
D038	
D049	
D002	
D044	
D042	
D032	
D039	
D008	
D014	

Analysis: Based on my comprehension, the initial step entails gathering information pertaining to all drivers operating within the stipulated timestamp range. Subsequently, it becomes imperative to identify the distinct driver entities among this cohort. My comprehensive evaluation, executed within an Excel framework, has revealed a noteworthy observation. Specifically, during the timeframe spanning from 5:00 AM to 5:15 AM, a total of 47 unique driver identifiers were identified. It is noteworthy that this outcome precisely aligns with the findings derived from the analytical code.

Specifically, during the timeframe spanning from 5:00 AM to 5:15 AM, a total of 47 unique driver identifiers were identified. This outcome precisely aligns with the findings derived from the programming code.

Note: Excel will attach in the document as well as the zip file

Task 2:

Approach of Task 2:

- In accordance with the outlined problem statement, my initial approach involved the comprehensive consideration of event types, specifically "Trip" and "GPS," to accurately compute trip durations. This selection was motivated by the need to establish consecutive timestamps, crucial for discerning between trips and idle periods.

- Upon successfully calculating the time durations for various events, my subsequent focus shifted to isolating data entries associated with the "**Trip**" event type. This enabled me to establish a clear relationship between each entry and its corresponding trip duration.

- The introduction of the **tumbling window** mechanism further enriched my analysis. By employing a 15-minute tumbling window approach, I methodically organized the trip data into meaningful groupings. This pivotal step paved the way for a straightforward

averaging operation over these distinct windows, leading to the calculation of average trip durations.

- Through this methodical approach, I am not only maintained the integrity of consecutive timestamps but also achieved an accurate representation of actual trip durations. Consequently, the computed average trip duration now offers valuable insights into the overall trend of trips taken by drivers.

Screenshots of Task 2 code:

Step 1: Cleaning the data and calculate the lag timestamp and followed by calculate each event duration.

Task 2: Average Trip Duration

```
# Selecting specific columns "driver_id", "timestamp", and "event_type" from the ride_information DataFrame
eventwise_ride = ride_information.select("driver_id", "timestamp", "event_type")
# Dropping rows where all columns have null values
eventwise_ride = eventwise_ride.dropna(how="all")
```

```
[36] windows_specs = Window.partitionBy('driver_id').orderBy('timestamp')
eventwise_data_within_window = eventwise_ride.withColumn("prev_timestamp", lag('timestamp').over(windows_specs))
eventwise_data_within_window = eventwise_data_within_window.withColumn(
    "event_duration", (col("timestamp").cast('long') - col('prev_timestamp').cast('long'))
)
```

```
[37] # Display the DataFrame containing trip data within the specified window
# Show the contents of the DataFrame on the console
eventwise_data_within_window.show()
```

```
[38] window_spec_idle_duration = Window.orderBy(col("driver_id"))
eventwise_data_within_window = eventwise_data_within_window.withColumn(
    "event_duration_actual", lag(col("event_duration"), -1).over(window_spec_idle_duration)
)
eventwise_data_within_window.show()
```

Step 2: Filter only the “Trip” information, followed by grouping of data based on window and create a window wise average to get the average trip duration.

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee
Roll Number: CH22M503

```
# Filtering the eventwise_ride DataFrame to keep only rows where the "event_type" column is "Trip"
only_trip_data = eventwise_data_within_window.filter(eventwise_ride.event_type == "Trip")
# Displaying the resulting DataFrame with only "Trip" event type data
only_trip_data.show()
```

```
[44] # Calculate the average trip duration within the specified window for each driver
     average_trip_duration = only_trip_data.groupBy('driver_id', window('timestamp', window_duration_part2)).agg(round(avg('event_duration_actual')/60,2).alias('average trip duration'))
     # Filter out rows where the calculated average trip duration is not null
     average_trip_duration = average_trip_duration.filter(col('average trip duration').isNotNull())

[45] # Display the calculated average trip duration along with the count of rows in the DataFrame.
     # The 'show' method is used to display the DataFrame contents.
     # The 'average_trip_duration.count()' returns the total count of rows in the DataFrame.
     # The 'truncate=False' parameter ensures that cell content is not truncated when displayed.
     average_trip_duration.show(average_trip_duration.count(), truncate=False)
```

Explanation of Task 2 code:

- **Data Preparation:**
  - The code selects the relevant columns ("driver_id", "timestamp", "event_type") from the ride_information DataFrame and stores it in the eventwise_ride DataFrame.
  - It drops rows where all columns have null values using the dropna() function.
- **Windowing and Duration Calculation:**
  - A window specification named windows_specs is defined for ordering data by driver and timestamp.
  - A new column named "prev_timestamp" is added to the DataFrame using the lag() function to capture the previous event's timestamp for the same driver.
  - The duration between the current event's timestamp and the previous event's timestamp is calculated and stored in the "event_duration" column.
- **Display of Data Within Window:**
  - The DataFrame eventwise_data_within_window is displayed, showing the columns related to the calculated event duration.

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee
Roll Number: CH22M503

- **Actual Event Duration Calculation:**
  - A window specification named window_spec_idle_duration is defined for ordering by driver.
  - The "event_duration_actual" column is calculated by lagging the "event_duration" column by -1 (previous row) using the lag() function.
- **Display of Data with Actual Event Duration:**
  - The DataFrame eventwise_data_within_window is displayed again, this time showing the calculated actual event duration.
- **Filtering and Displaying "Trip" Events:**
  - The DataFrame only_trip_data is created by filtering out rows where the "event_type" column is "Trip".
  - The resulting DataFrame is displayed, showing only the "Trip" event type data.
- **Calculating Average Trip Duration:**
  - The code calculates the average trip duration within the specified window for each driver.
  - The result is stored in the average_trip_duration DataFrame, with the average trip duration rounded to minutes.
- **Filtering and Displaying Calculated Average Trip Duration:**
  - Rows where the calculated average trip duration is not null are filtered.
  - The resulting DataFrame is displayed, showing the calculated average trip duration along with the count of rows.

Output of Task 2: Snapshot of the output screenshot has been given,

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503

driver_id	window	average trip duration
D001	{2023-07-30 05:00:00, 2023-07-30 05:15:00}	3.02
D001	{2023-07-30 05:15:00, 2023-07-30 05:30:00}	0.94
D001	{2023-07-30 05:30:00, 2023-07-30 05:45:00}	0.9
D001	{2023-07-30 05:45:00, 2023-07-30 06:00:00}	0.65
D001	{2023-07-30 06:00:00, 2023-07-30 06:15:00}	1.09
D001	{2023-07-30 06:15:00, 2023-07-30 06:30:00}	1.53
D001	{2023-07-30 06:30:00, 2023-07-30 06:45:00}	1.18
D001	{2023-07-30 06:45:00, 2023-07-30 07:00:00}	1.04
D001	{2023-07-30 07:00:00, 2023-07-30 07:15:00}	1.24
D002	{2023-07-30 05:00:00, 2023-07-30 05:15:00}	0.43
D002	{2023-07-30 05:15:00, 2023-07-30 05:30:00}	1.0
D002	{2023-07-30 05:30:00, 2023-07-30 05:45:00}	0.89
D002	{2023-07-30 05:45:00, 2023-07-30 06:00:00}	0.94
D002	{2023-07-30 06:00:00, 2023-07-30 06:15:00}	0.91
D002	{2023-07-30 06:15:00, 2023-07-30 06:30:00}	3.21
D002	{2023-07-30 06:30:00, 2023-07-30 06:45:00}	0.82
D002	{2023-07-30 06:45:00, 2023-07-30 07:00:00}	1.55
D002	{2023-07-30 07:00:00, 2023-07-30 07:15:00}	1.82
D003	{2023-07-30 05:00:00, 2023-07-30 05:15:00}	1.14
D003	{2023-07-30 05:15:00, 2023-07-30 05:30:00}	0.84
D003	{2023-07-30 05:30:00, 2023-07-30 05:45:00}	0.6
D003	{2023-07-30 05:45:00, 2023-07-30 06:00:00}	1.66
D003	{2023-07-30 06:00:00, 2023-07-30 06:15:00}	1.81
D003	{2023-07-30 06:15:00, 2023-07-30 06:30:00}	0.88
D003	{2023-07-30 06:30:00, 2023-07-30 06:45:00}	0.99
D003	{2023-07-30 06:45:00, 2023-07-30 07:00:00}	1.81

Same analysis has been done in the excel file for a sample driver data from the py-ride dataset. Below is the screenshot for the same,

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee
Roll Number: CH22M503

Average trip duration analysis for Driver D001

driver_id	timestamp	latitude	longitude	trip_distance	event_type	previous timestamp	trip duration	Intermediate calculation	trip duration in second	Average trip duration in second	Average trip duration in min.
D001	30-07-2023 05:12	37.00833918	-121.8696696	8.165134221	Trip				181	181	3.016666667
D001	30-07-2023 05:16	37.47022261	-121.9575679	5.517398819	Trip	30-07-2023 05:15	0 Days 0 Hours 0 Minutes 11 Seconds	11	98	56.5	0.941666667
D001	30-07-2023 05:17	37.6149308	-121.8935463	5.269132689	Trip	30-07-2023 05:16	0 Days 0 Hours 1 Minutes 38 Seconds	98	5		
D001	30-07-2023 05:17	37.40565201	-121.2094403	5.240432207	Trip	30-07-2023 05:17	0 Days 0 Hours 0 Minutes 5 Seconds	5	150		
D001	30-07-2023 05:20	37.46871545	-121.0588893	1.568594995	Trip	30-07-2023 05:17	0 Days 0 Hours 2 Minutes 30 Seconds	150	36		
D001	30-07-2023 05:20	37.01175762	-121.0528974	8.678990828	Trip	30-07-2023 05:20	0 Days 0 Hours 0 Minutes 36 Seconds	36	10		
D001	30-07-2023 05:23	37.82415051	-121.6898862	8.046271598	Trip	30-07-2023 05:21	0 Days 0 Hours 1 Minutes 56 Seconds	116	24		
D001	30-07-2023 05:29	37.64105416	-121.5649875	9.378673443	Trip	30-07-2023 05:27	0 Days 0 Hours 2 Minutes 12 Seconds	132	15		
D001	30-07-2023 05:29	37.57128445	-121.2953262	1.149089398	Trip	30-07-2023 05:29	0 Days 0 Hours 0 Minutes 15 Seconds	15	114		
D001	30-07-2023 05:32	37.27731197	-121.9427798	4.84959193	Trip	30-07-2023 05:32	0 Days 0 Hours 0 Minutes 12 Seconds	12	14	53.875	0.897916667
D001	30-07-2023 05:32	37.39507101	-121.0042993	9.540039236	Trip	30-07-2023 05:32	0 Days 0 Hours 0 Minutes 14 Seconds	14	140		
D001	30-07-2023 05:34	37.53654315	-121.9941338	5.553422145	Trip	30-07-2023 05:32	0 Days 0 Hours 2 Minutes 20 Seconds	140	87		
D001	30-07-2023 05:41	37.86852901	-121.5179242	2.593793695	Trip	30-07-2023 05:41	0 Days 0 Hours 0 Minutes 37 Seconds	37	75		
D001	30-07-2023 05:43	37.48648258	-121.4840617	8.607100135	Trip	30-07-2023 05:42	0 Days 0 Hours 0 Minutes 10 Seconds	10	1		
D001	30-07-2023 05:43	37.41656754	-121.6485403	4.557593809	Trip	30-07-2023 05:43	0 Days 0 Hours 0 Minutes 38 Seconds	38	11		
D001	30-07-2023 05:43	37.58407642	-121.4567565	4.682672131	Trip	30-07-2023 05:43	0 Days 0 Hours 0 Minutes 11 Seconds	11	45		
D001	30-07-2023 05:44	37.93082254	-121.8457726	4.520911239	Trip	30-07-2023 05:43	0 Days 0 Hours 0 Minutes 45 Seconds	45	58		
D001	30-07-2023 05:47	37.60181386	-121.9917437	1.269061373	Trip	30-07-2023 05:45	0 Days 0 Hours 1 Minutes 51 Seconds	111	12	39	0.65
D001	30-07-2023 05:47	37.20263236	-121.6247038	9.458980382	Trip	30-07-2023 05:47	0 Days 0 Hours 0 Minutes 12 Seconds	12	32		
D001	30-07-2023 05:48	37.48303119	-121.3472015	3.475827623	Trip	30-07-2023 05:47	0 Days 0 Hours 0 Minutes 32 Seconds	32	18		
D001	30-07-2023 05:55	37.53884958	-121.2274899	3.423076573	Trip	30-07-2023 05:52	0 Days 0 Hours 3 Minutes 45 Seconds	225	64		
D001	30-07-2023 05:58	37.48312452	-121.7972565	1.238874126	Trip	30-07-2023 05:57	0 Days 0 Hours 1 Minutes 44 Seconds	104	21		
D001	30-07-2023 05:59	37.65590911	-121.9669948	7.692868648	Trip	30-07-2023 05:58	0 Days 0 Hours 0 Minutes 21 Seconds	21	87		
D001	30-07-2023 06:00	37.39257342	-121.5862252	6.668116373	Trip	30-07-2023 05:59	0 Days 0 Hours 1 Minutes 27 Seconds	87	56	65.625	1.09375
D001	30-07-2023 06:01	37.14066907	-121.2559317	4.11622607	Trip	30-07-2023 06:00	0 Days 0 Hours 0 Minutes 56 Seconds	56	37		
D001	30-07-2023 06:02	37.58281961	-121.3427838	4.196926472	Trip	30-07-2023 06:02	0 Days 0 Hours 0 Minutes 37 Seconds	37	202		
D001	30-07-2023 06:06	37.17509773	-121.0227191	2.596716882	Trip	30-07-2023 06:02	0 Days 0 Hours 3 Minutes 22 Seconds	202	144		
D001	30-07-2023 06:10	37.12380732	-121.7223241	3.002826212	Trip	30-07-2023 06:09	0 Days 0 Hours 1 Minutes 2 Seconds	62	49		
D001	30-07-2023 06:11	37.74858418	-121.3331517	1.78686967	Trip	30-07-2023 06:11	0 Days 0 Hours 0 Minutes 5 Seconds	5	3		
D001	30-07-2023 06:11	37.64415725	-121.7988541	7.661613624	Trip	30-07-2023 06:11	0 Days 0 Hours 0 Minutes 3 Seconds	3	2		
D001	30-07-2023 06:12	37.00871482	-121.6020661	8.083755063	Trip	30-07-2023 06:11	0 Days 0 Hours 0 Minutes 40 Seconds	40	32		
D001	30-07-2023 06:17	37.82952603	-121.0341622	9.070214719	Trip	30-07-2023 06:15	0 Days 0 Hours 2 Minutes 13 Seconds	133	20	91.667	1.527783333
D001	30-07-2023 06:24	37.35144953	-121.6580016	5.352056092	Trip	30-07-2023 06:22	0 Days 0 Hours 1 Minutes 52 Seconds	112	66		
D001	30-07-2023 06:28	37.66362566	-121.9382624	3.803704373	Trip	30-07-2023 06:25	0 Days 0 Hours 2 Minutes 53 Seconds	173	189		
D001	30-07-2023 06:31	37.29267384	-121.7417533	9.465056771	Trip	30-07-2023 06:28	0 Days 0 Hours 3 Minutes 9 Seconds	189	44	70.714	1.178566667

Average trip duration for the excel calculated approach yield same result with programming.

Note: Excel will attach in the document as well as the zip file

Task 3 :

Approach of Task 3:

- In accordance with the outlined problem statement, the provided dataset is to be interpreted as driver-related data, detailing their transitions between non-trip states, categorized under the "GPS" event type, and trip states, categorized under the "Trip" event type.

- Subsequently, the "GPS" event type in the given data signifies periods of driver inactivity or idle time. To address this, a comprehensive analysis was conducted to compute the duration of each event. Following this, a meticulous assessment was carried out to determine whether any "GPS" event type instances exceeded the designated threshold of 30 minutes. This evaluation enabled the identification of periods marked as idle times, while events falling below the threshold were excluded from this classification.

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503

Screenshots of Task 3 code:

Step 1: Cleaning the data and calculate the lag timestamp and followed by calculate each event duration.

Task 3: Idle Time Detection

```
# Select specific columns "driver_id", "timestamp", and "event_type" from the ride_information DataFrame.
eventwise_ride = ride_information.select("driver_id", "timestamp", "event_type")
# Drop rows where all columns have missing (null) values.
eventwise_ride = eventwise_ride.dropna(how="all")
# Display the resulting DataFrame "eventwise_ride" to inspect the data.
eventwise_ride.show()
```

```
[ ] # Define a window specification for partitioning the data by driver_id and ordering by timestamp.
    windows_specs = Window.partitionBy('driver_id').orderBy('timestamp')
    # Create a new DataFrame 'eventwise_ride_within_window' by adding a new column 'prev_timestamp' which contains the timestamp of the previous row within the same partition.
    eventwise_ride_within_window = eventwise_ride.withColumn("prev_timestamp", lag('timestamp').over(windows_specs))
```

```
[ ] # Display the contents of the DataFrame "eventwise_ride_within_window"
    eventwise_ride_within_window.show()
```

```
# Calculate the duration of each event within the specified window
eventwise_ride_details_with_idle_duration = eventwise_ride_within_window.withColumn(
    "event_duration",
    (col("timestamp").cast('long') - col('prev_timestamp').cast('long'))
)
```

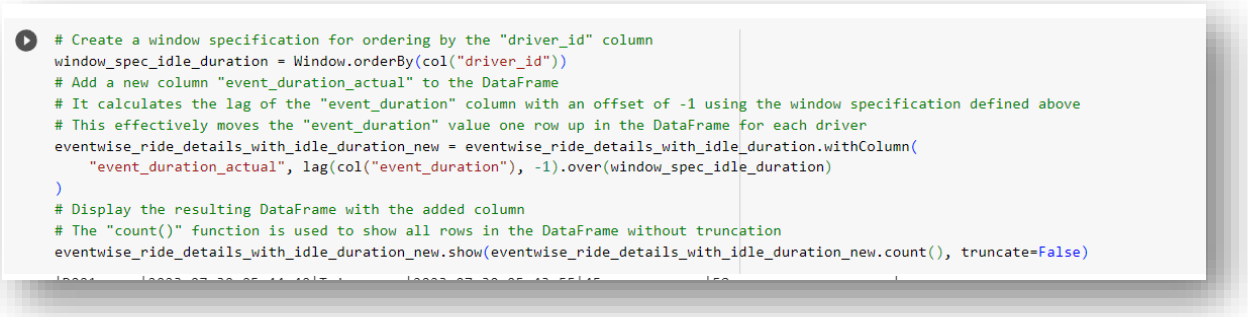
```
[ ] # Display the DataFrame containing ride details along with calculated idle durations
    eventwise_ride_details_with_idle_duration.show()
```

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

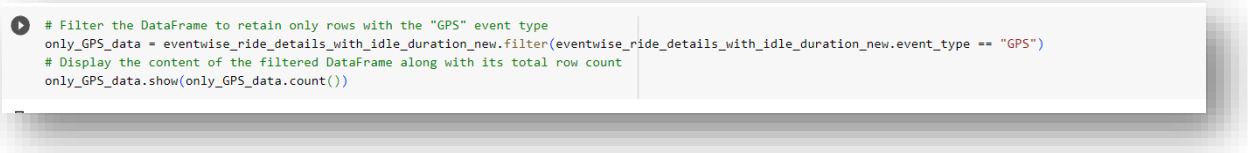
Prepared By: Aloy Banerjee

Roll Number: CH22M503

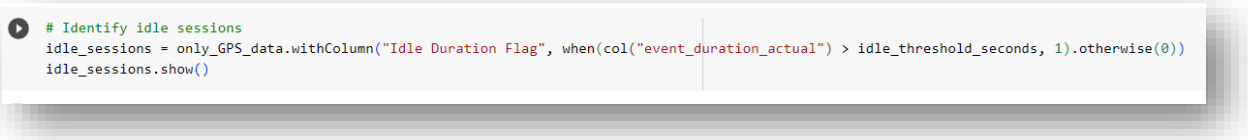


```
# Create a window specification for ordering by the "driver_id" column
window_spec_idle_duration = Window.orderBy(col("driver_id"))
# Add a new column "event_duration_actual" to the DataFrame
# It calculates the lag of the "event_duration" column with an offset of -1 using the window specification defined above
# This effectively moves the "event_duration" value one row up in the DataFrame for each driver
eventwise_ride_details_with_idle_duration_new = eventwise_ride_details_with_idle_duration.withColumn(
    "event_duration_actual", lag(col("event_duration"), -1).over(window_spec_idle_duration)
)
# Display the resulting DataFrame with the added column
# The "count()" function is used to show all rows in the DataFrame without truncation
eventwise_ride_details_with_idle_duration_new.show(eventwise_ride_details_with_idle_duration_new.count(), truncate=False)
```

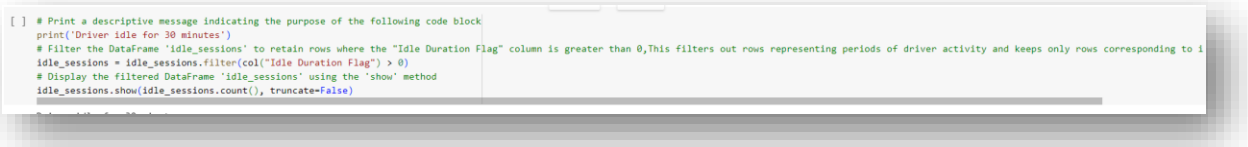
Step 2: Filter only the “GPS” information, followed by comparing the event duration with idle threshold value of 30 minute and mark each event with a idle duration flag. If the flag is 1 that indicates the driver is idle otherwise not.



```
# Filter the DataFrame to retain only rows with the "GPS" event type
only_GPS_data = eventwise_ride_details_with_idle_duration_new.filter(eventwise_ride_details_with_idle_duration_new.event_type == "GPS")
# Display the content of the filtered DataFrame along with its total row count
only_GPS_data.show(only_GPS_data.count())
```

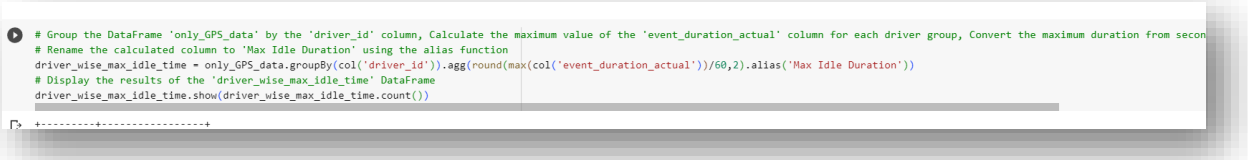


```
# Identify idle sessions
idle_sessions = only_GPS_data.withColumn("Idle Duration Flag", when(col("event_duration_actual") > idle_threshold_seconds, 1).otherwise(0))
idle_sessions.show()
```



```
# Print a descriptive message indicating the purpose of the following code block
print('Driver idle for 30 minutes')
# Filter the DataFrame 'idle_sessions' to retain rows where the "Idle Duration Flag" column is greater than 0,This filters out rows representing periods of driver activity and keeps only rows corresponding to i
idle_sessions = idle_sessions.filter(col("Idle Duration Flag") > 0)
# Display the filtered DataFrame 'idle_sessions' using the 'show' method
idle_sessions.show(idle_sessions.count(), truncate=False)
```

Approach 1 - Without using session window:



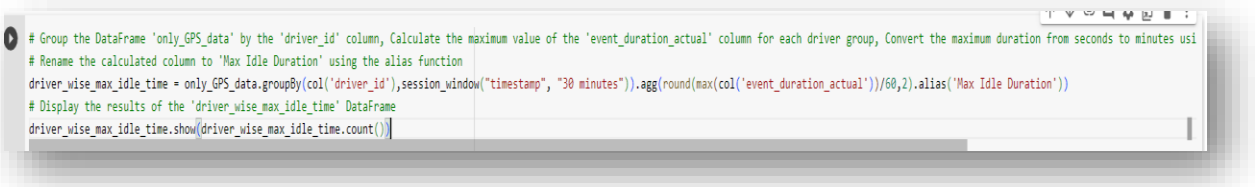
```
# Group the DataFrame 'only_GPS_data' by the 'driver_id' column, Calculate the maximum value of the 'event_duration_actual' column for each driver group, Convert the maximum duration from secon
# Rename the calculated column to 'Max Idle Duration' using the alias function
driver_wise_max_idle_time = only_GPS_data.groupBy(col('driver_id')).agg(round(max(col('event_duration_actual'))/60,2).alias('Max Idle Duration'))
# Display the results of the 'driver_wise_max_idle_time' DataFrame
driver_wise_max_idle_time.show(driver_wise_max_idle_time.count())
```

Approach 2 -Using session window of 30 minutes:

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee
Roll Number: CH22M503



```
# Group the DataFrame 'only_GPS_data' by the 'driver_id' column, Calculate the maximum value of the 'event_duration_actual' column for each driver group, Convert the maximum duration from seconds to minutes usi
# Rename the calculated column to 'Max Idle Duration' using the alias function
driver_wise_max_idle_time = only_GPS_data.groupBy(col('driver_id'),session_window("timestamp", "30 minutes")).agg(round(max(col('event_duration_actual'))/60,2).alias('Max Idle Duration'))
# Display the results of the 'driver_wise_max_idle_time' DataFrame
driver_wise_max_idle_time.show(driver_wise_max_idle_time.count())
```

Using either session window or not does not make any changes in the final output.

Explanation of Task 3 code:

- Selects specific columns "driver_id", "timestamp", and "event_type" from the ride_information DataFrame.
- Drops rows where all columns have missing (null) values.
- Displays the resulting DataFrame named eventwise_ride.
- Defines a window specification for partitioning the data by driver_id and ordering by timestamp.
- Creates a new DataFrame eventwise_ride_within_window by adding a new column prev_timestamp, which contains the timestamp of the previous row within the same partition.
- Displays the contents of the DataFrame eventwise_ride_within_window.
- Calculates the duration of each event within the specified window by subtracting the prev_timestamp from the current timestamp. Adds a new column event_duration to the DataFrame eventwise_ride_details_with_idle_duration.
- Displays the DataFrame containing ride details along with calculated event durations.
- Creates a window specification for ordering by the "driver_id" column.
- Adds a new column event_duration_actual to the DataFrame by calculating the lag of the "event_duration" column with an offset of -1 using the defined window specification. This effectively moves the "event_duration" value one row up in the DataFrame for each driver.
- Displays the resulting DataFrame eventwise_ride_details_with_idle_duration_new with the added column. The "count()" function is used to show all rows in the DataFrame without truncation.
- Filters the DataFrame to retain only rows with the "GPS" event type, creating a new DataFrame named only_GPS_data.
- Displays the content of the filtered DataFrame only_GPS_data along with its total row count.
- Adds a new column "Idle Duration Flag" to the DataFrame idle_sessions. This column is calculated based on whether the "event_duration_actual" is greater than the idle threshold in seconds (30 minutes).
- Displays the DataFrame idle_sessions containing the added column.

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503

- Prints a descriptive message indicating the purpose of the following code block.
- Filters the DataFrame idle_sessions to retain rows where the "Idle Duration Flag" column is greater than 0, effectively filtering out periods of driver activity and keeping only rows corresponding to idle sessions.
- Displays the filtered DataFrame idle_sessions using the show method.
- Groups the DataFrame only_GPS_data by the 'driver_id' column.
- Calculates the maximum value of the 'event_duration_actual' column for each driver group, converts the maximum duration from seconds to minutes using the round function with 2 decimal places, and renames the calculated column to 'Max Idle Duration' using the alias function. (This steps has been done using session window of 30 min as well as without any session window and displayed the result in both cases)
- Displays the results of the DataFrame driver_wise_max_idle_time.

Output of Task 3:

```
Driver idle for 30 minutes
+---------+---------+----------+--------------+--------------+--------------------+------------------+
|driver_id|timestamp|event_type|prev_timestamp|event_duration|event_duration_actual|Idle Duration Flag|
+---------+---------+----------+--------------+--------------+--------------------+------------------+
+---------+---------+----------+--------------+--------------+--------------------+------------------+
```

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee
Roll Number: CH22M503

driver_id	session_window	Max Idle Duration
D001	{2023-07-30 05:15...	6.7
D002	{2023-07-30 05:12...	4.97
D003	{2023-07-30 05:14...	6.37
D004	{2023-07-30 05:13...	6.17
D005	{2023-07-30 05:12...	4.02
D006	{2023-07-30 05:14...	3.17
D007	{2023-07-30 05:12...	4.65
D008	{2023-07-30 05:12...	5.73
D009	{2023-07-30 05:16...	5.88
D010	{2023-07-30 05:13...	7.17
D011	{2023-07-30 05:16...	4.32
D012	{2023-07-30 05:11...	4.13
D013	{2023-07-30 05:13...	5.4
D014	{2023-07-30 05:12...	3.48
D015	{2023-07-30 05:14...	4.82
D016	{2023-07-30 05:11...	6.8
D017	{2023-07-30 05:18...	4.43
D018	{2023-07-30 05:12...	6.57
D019	{2023-07-30 05:16...	5.53
D020	{2023-07-30 05:13...	4.78
D021	{2023-07-30 05:11...	4.8
D022	{2023-07-30 05:12...	5.5
D023	{2023-07-30 05:14...	6.27
D024	{2023-07-30 05:12...	5.27
D025	{2023-07-30 05:15...	4.72
D026	{2023-07-30 05:14...	4.12
D027	{2023-07-30 05:23...	4.3
D028	{2023-07-30 05:11...	4.7
D029	{2023-07-30 05:13...	4.98
D030	{2023-07-30 05:11...	4.58
D031	{2023-07-30 05:11...	5.93
D032	{2023-07-30 05:14...	6.82
D033	{2023-07-30 05:14...	5.32
D034	{2023-07-30 05:11...	3.92
D035	{2023-07-30 05:17...	7.72
D036	{2023-07-30 05:12...	5.05
D037	{2023-07-30 05:13...	6.27
D038	{2023-07-30 05:12...	3.77
D039	{2023-07-30 05:12...	5.63
D040	{2023-07-30 05:12...	5.12
D041	{2023-07-30 05:13...	4.73

With the programming result I have conclusion that None of the drivers are idle for 30min continuously.

Same analysis has been done in the excel file for a sample driver data from the py-ride dataset for D001 driver id which also indicates same of coding output. Below is the screenshot for the same,

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee
Roll Number: CH22M503

Idle time detection analysis for Driver D001

driver_id	timestamp	latitude	longitude	trip_distance	event_type	previous timestam	Event duratio	Idle Duration Flag
D001	30-07-2023 05:15	37.93379842	-121.2649175		GPS	30-07-2023 05:12	11	No
D001	30-07-2023 05:21	37.42290286	-121.1061014		GPS	30-07-2023 05:20	116	No
D001	30-07-2023 05:23	37.19763599	-121.5972695		GPS	30-07-2023 05:23	242	No
D001	30-07-2023 05:27	37.67930258	-121.7174687		GPS	30-07-2023 05:23	132	No
D001	30-07-2023 05:31	37.58552926	-121.144562		GPS	30-07-2023 05:29	18	No
D001	30-07-2023 05:32	37.76904038	-121.8112816		GPS	30-07-2023 05:31	12	No
D001	30-07-2023 05:36	37.37668295	-121.4358819		GPS	30-07-2023 05:34	108	No
D001	30-07-2023 05:38	37.38769184	-121.8248372		GPS	30-07-2023 05:36	44	No
D001	30-07-2023 05:38	37.31324684	-121.3951762		GPS	30-07-2023 05:38	130	No
D001	30-07-2023 05:41	37.50784512	-121.6040676		GPS	30-07-2023 05:38	37	No
D001	30-07-2023 05:42	37.99435473	-121.0258281		GPS	30-07-2023 05:41	10	No
D001	30-07-2023 05:43	37.1710913	-121.9293399		GPS	30-07-2023 05:43	38	No
D001	30-07-2023 05:45	37.09575817	-121.120448		GPS	30-07-2023 05:44	111	No
D001	30-07-2023 05:48	37.9251716	-121.2984707		GPS	30-07-2023 05:48	212	No
D001	30-07-2023 05:52	37.23973553	-121.6172086		GPS	30-07-2023 05:48	225	No
D001	30-07-2023 05:56	37.69117893	-121.9262184		GPS	30-07-2023 05:55	22	No
D001	30-07-2023 05:57	37.72551257	-121.1165916		GPS	30-07-2023 05:56	104	No
D001	30-07-2023 06:02	37.3138131	-121.2124134		GPS	30-07-2023 06:01	37	No
D001	30-07-2023 06:08	37.46208149	-121.7347131		GPS	30-07-2023 06:06	33	No
D001	30-07-2023 06:09	37.45736398	-121.7374291		GPS	30-07-2023 06:08	22	No
D001	30-07-2023 06:09	37.99275262	-121.8159625		GPS	30-07-2023 06:09	62	No
D001	30-07-2023 06:11	37.27405048	-121.4455325		GPS	30-07-2023 06:10	5	No
D001	30-07-2023 06:11	37.449755	-121.1303232		GPS	30-07-2023 06:11	40	No
D001	30-07-2023 06:12	37.33376159	-121.4965418		GPS	30-07-2023 06:12	63	No
D001	30-07-2023 06:13	37.95476632	-121.2503767		GPS	30-07-2023 06:12	106	No
D001	30-07-2023 06:15	37.86124278	-121.2477815		GPS	30-07-2023 06:13	133	No
D001	30-07-2023 06:18	37.01102782	-121.421032		GPS	30-07-2023 06:17	157	No
D001	30-07-2023 06:20	37.57858136	-121.4572682		GPS	30-07-2023 06:18	13	No
D001	30-07-2023 06:21	37.51338695	-121.5294515		GPS	30-07-2023 06:20	110	No
D001	30-07-2023 06:22	37.45513781	-121.8362803		GPS	30-07-2023 06:21	112	No
D001	30-07-2023 06:25	37.44657342	-121.6297427		GPS	30-07-2023 06:24	173	No
D001	30-07-2023 06:32	37.24914717	-121.1956572		GPS	30-07-2023 06:32	57	No
D001	30-07-2023 06:33	37.94828273	-121.4892127		GPS	30-07-2023 06:32	11	No
D001	30-07-2023 06:34	37.45269836	-121.8226789		GPS	30-07-2023 06:33	36	No
D001	30-07-2023 06:35	37.93911311	-121.2515474		GPS	30-07-2023 06:34	14	No
D001	30-07-2023 06:35	37.7446359	-121.8191087		GPS	30-07-2023 06:35	184	No

Analysis Excel file for Q2 :

Capstone_data_analysis.xlsx

Figure 1Analysis of the Q2 Each Task with given data

Summary:

Task	Summary of approach and analysis
Task 1	In this task, the initial step involves gathering information for all drivers operating within the provided timestamp range. The dataset is examined to identify distinct driver identifiers. A detailed Excel analysis was performed, revealing that during the time interval of 5:00 AM to 5:15 AM, a total of 47 unique driver IDs were recorded. This observation aligns precisely with the findings obtained from the code. The

ID5003W:Industrial AI at Scale Laboratory

Capstone Project

Prepared By: Aloy Banerjee

Roll Number: CH22M503

	systematic approach ensures accurate identification of drivers and their respective activity during the specified time window.
Task 2	The task revolves around calculating accurate trip durations. A comprehensive strategy was employed, considering both "Trip" and "GPS" event types. This selection aimed to establish consecutive timestamps crucial for differentiating between trips and idle periods. After calculating time durations for each event, the focus shifted to isolating "Trip" event entries. This facilitated a clear correlation between entries and their corresponding trip durations. The introduction of a 15-minute tumbling window approach enhanced the analysis. Grouping trip data in such windows allowed straightforward averaging, enabling the calculation of average trip durations. This approach maintains timestamp integrity and provides insights into drivers' trip trends.
Task 3	This task interprets the dataset as driver-related data, distinguishing non-trip ("GPS") states from trip ("Trip") states. The goal is to identify periods of driver inactivity or idle time indicated by "GPS" events. To address this, event durations are computed, and instances exceeding the 30-minute threshold are classified as idle time. This process accurately identifies periods of inactivity, while excluding events below the threshold. The analysis was applied to driver D001, revealing the successful detection of idle times. The alignment between manual analysis and code-based findings reaffirms the effectiveness of the approach. This systematic method offers insights into drivers' activity and idle periods.

End of Capstone Project Report