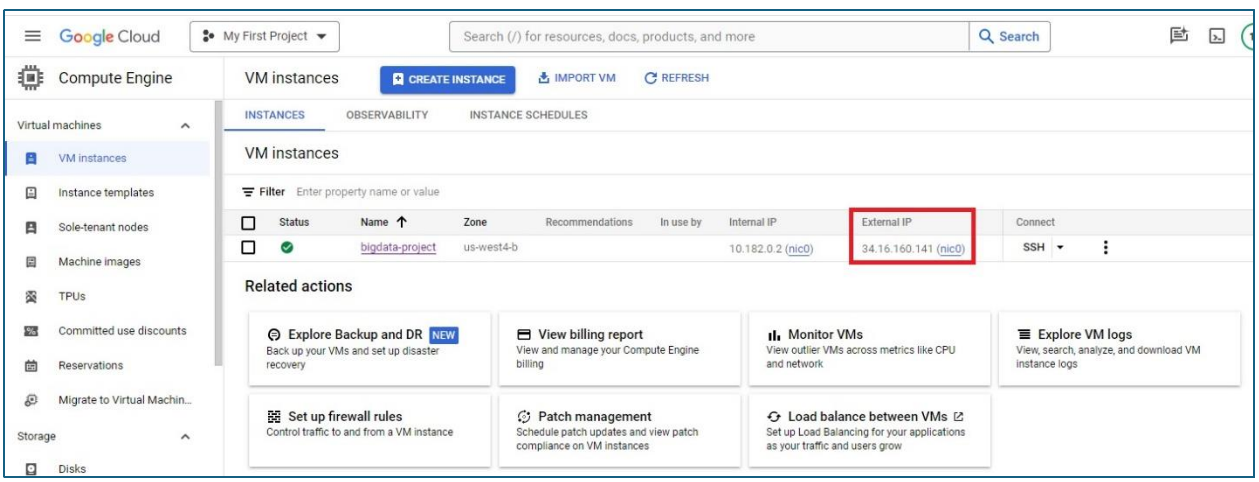
Building Machine Learning model on Train Arrival predictions using Big Data Technologies

Introduction

In this project, several big data technologies such as HDFS, Apache Hive, Apache Kafka, Apache Solr, and Apache Spark were used to build a Machine learning model to predict if the train arrival will be delayed or not. The source data for this project is taken from the API of the Chicago Transit Authority-CTA https://www.transitchicago.com/developers/traintracker/. The live data from the API is read and processed using big data and Machine Learning models are built on the data in Real time.

Infrastructure Setup:

All infrastructure for the project was provisioned on a Virtual Machine on GCP. An elastic IP was reserved for the project as highlighted in the screenshot.



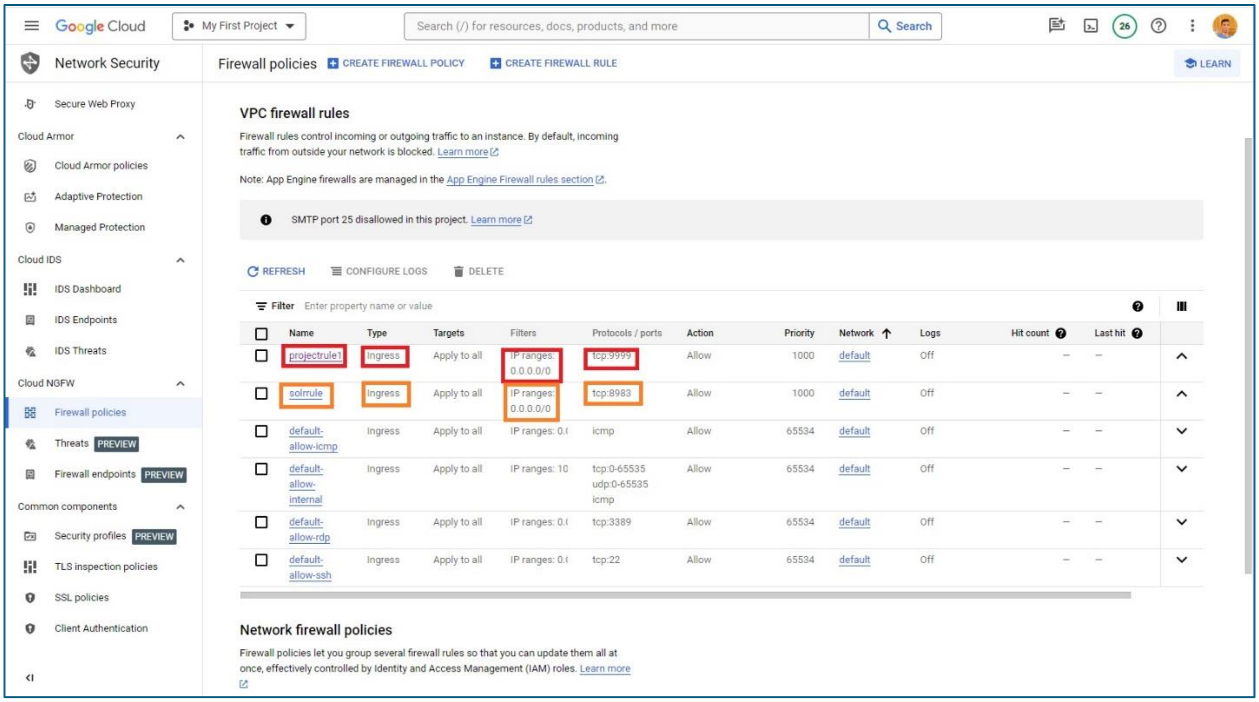
Hadoop version 3.3.6, Spark version 3.5.0, Solr Version 8.2.0, and Kafka version 2.6.1 were installed manually, and soft links were created in the /opt folder as shown in the screenshot. The JPS command shows the process running in the background.

```
guruprasadvk10@bigdata-project:~$ cd /opt
guruprasadvk10@bigdata-project:/opt$ ls -ltr
total 177680
-rwxr-xr-x  1 root           root               12694 Jul 18  2019 install_solr_service.sh
-rw-r--r--  1 root           root           181899182 Jul 19  2019 solr-8.2.0.tgz
drwxr-xr-x 13 guruprasadvk10 guruprasadvk10      4096 Sep  9 02:08 spark-3.5.0-bin-hadoop3
lrwxrwxrwx  1 root           root                  17 Feb 20 06:12 hadoop -> /opt/hadoop-3.3.6
drwxr-xr-x 12 guruprasadvk10 guruprasadvk10      4096 Feb 20 14:53 hadoop-3.3.6
drwxrwxr-x 10 guruprasadvk10 guruprasadvk10      4096 Feb 20 15:49 apache-hive-3.1.2-bin
lrwxrwxrwx  1 root           root                  26 Feb 20 15:51 hive -> /opt/apache-hive-3.1.2-bin
drwx--x--x  4 root           root                4096 Feb 20 16:00 containerd
lrwxrwxrwx  1 root           root                  28 Feb 20 19:11 spark3 -> /opt/spark-3.5.0-bin-hadoop3
lrwxrwxrwx  1 root           root                  16 Feb 20 21:43 kafka -> kafka_2.12-2.6.1
drwxr-xr-x  8 guruprasadvk10 guruprasadvk10      4096 Feb 21 05:32 kafka_2.12-2.6.1
drwxrwxr-x  5 guruprasadvk10 guruprasadvk10      4096 Feb 21 23:42 gen_logs
drwxrwxrwx  9 root           root                4096 Mar  2 17:50 solr-8.2.0
lrwxrwxrwx  1 root           root                  15 Mar  2 17:50 solr -> /opt/solr-8.2.0
guruprasadvk10@bigdata-project:/opt$ jps
29953 YarnCoarseGrainedExecutorBackend
29410 YarnCoarseGrainedExecutorBackend
2115 ResourceManager
1926 SecondaryNameNode
28839 ExecutorLauncher
91370 Jps
28077 SparkSubmit
1487 NameNode
27601 ConsoleConsumer
1651 DataNode
3193 QuorumPeerMain
2298 NodeManager
27227 ConnectStandalone
3550 Kafka
guruprasadvk10@bigdata-project:/opt$
```

The Hadoop and Kafka services were started and stopped using start_all.sh and stop_all.sh scripts respectively.

```
guruprasadvk10@bigdata-project:~$ cat start_all.sh
start-dfs.sh
start-yarn.sh
docker start cluster_util_db
zookeeper-server-start.sh -daemon /opt/kafka/config/zookeeper.properties
kafka-server-start.sh -daemon /opt/kafka/config/server.properties
jps
docker ps
guruprasadvk10@bigdata-project:~$
guruprasadvk10@bigdata-project:~$
guruprasadvk10@bigdata-project:~$ cat stop_all.sh
stop-yarn.sh
stop-dfs.sh
kafka-server-stop.sh
zookeeper-server-stop.sh
jps
guruprasadvk10@bigdata-project:~$
```

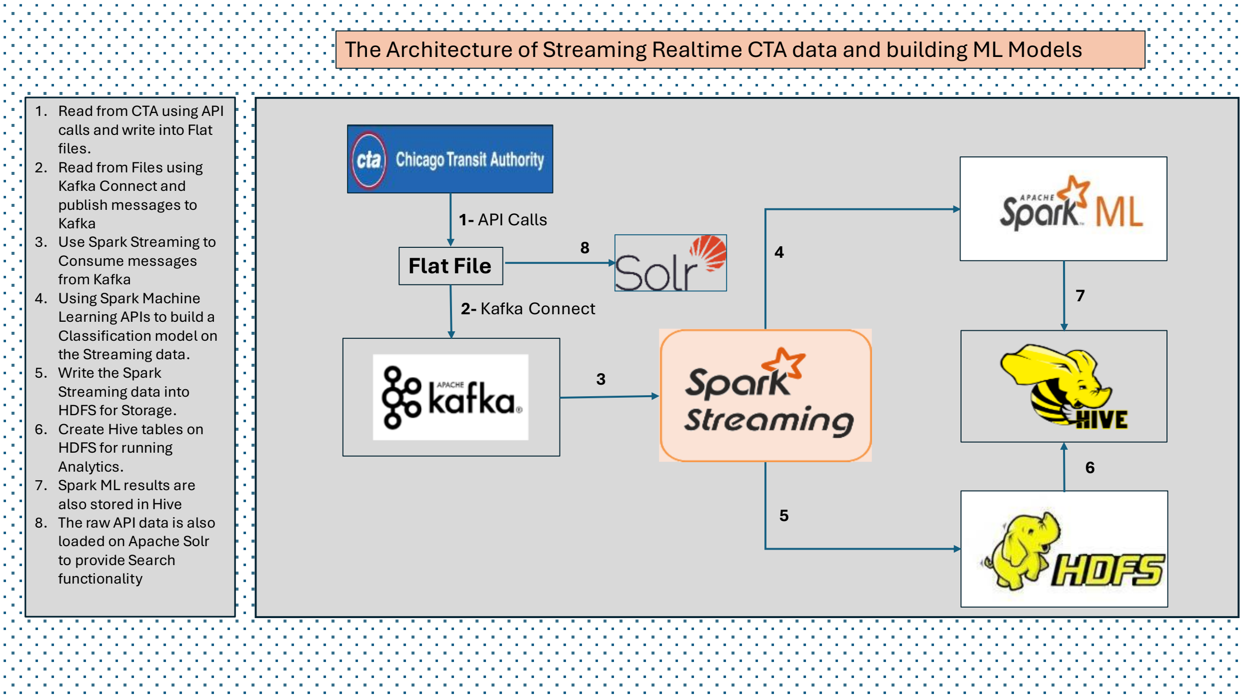
Also, the Jupyter Lab is provisioned on the port 9999 and Apache Solr on the port 8983. Hence 2 firewall rules were added on the GCP console to allow the Ingress traffic:



Project Overview

Live Streaming data was sourced from the CTA API http://lapi.transitchicago.com/api/1.0/ttpositions.aspx?key=4ba28f6b2b8843bf9cef1c0fcc05f874&rt=red&outputTy pe=JSON that takes the route name as an argument. The data was then processed using Python script and stored in a file which was used by Kafka Connect to publish to Kafka Topic. Also, Apache Solr is used to provide additional search functions on the data extracted from the API. The messages were read from Kafka using Spark Streaming APIs and the data was written into HDFS, on which a Hive table was built to query the data. Also, Spark Machine Learning Libraries were used to build a Classification Model on the live streaming data to determine if the train will be delayed or not.

As the project was built on a single node cluster on GCP, to keep the resource usage minimal without overwhelming the system, API calls were made every 10 seconds. Also, for each API call, the train information for one route was retrieved.



Module 1: Reading from the API

Script name: 1_Read_from_API.ipynb

The data was read from the CTA API

http://lapi.transitchicago.com/api/1.0/ttpositions.aspx?key=4ba28f6b2b8843bf9cef1c0fcc05f87

4&rt=red&outputTy pe=JSON using python script as shown below. The data was written into file

named **cta_api_dump.csv**

Reading from CTA API

```
[1]: # Loading the libraries
import pandas as pd
import numpy as np
import os
import re
import datetime
import requests
import json
import urllib3
import urllib.request,urllib.parse,urllib.error
from apscheduler.schedulers.background import BlockingScheduler
```

```
[2]: # Getting the current working directory
key_path=os.getcwd()
# Extracting the cta key from the json file created in the step above
with open("cta_key.json","r") as cta_key_file:
    json_key_cta=json.load(cta_key_file)
    # Reading the Key1 variable that contains the cta key
    cta_key=json_key_cta['key1']
```

```
[3]: def create_url():
    # Assigning the base URL to a variable
    cta_base_url="http://lapi.transitchicago.com/api/1.0/ttpositions.aspx?"
    # Creating a list of routes that willbe used in this project
    route_colors=["Red","Blue","Brn",'G',"Org","Pink"]
    # Randomly choosing a Route color list which will be used in the API call
    #route_color="Blue"
    route_color=np.random.choice(route_colors)
    print(f"The route used in the API is {route_color}")
    # Creating a dictionary of Parameters to be used in the API Call
    params2={'key':str(cta_key),'rt':route_color,"outputType":"JSON"}
    # Creating the final url by combining the base url and the Parameters that includes the API Key
    cta_api_url=str(cta_base_url)+urllib.parse.urlencode(params2)
    return cta_api_url
```

```python
[4]: def extract_cta_data(cta_json_data):
    """
    This function takes the json input and parses the data and returns a Dataframe of the extracted fields.
    """
    # Creating a list of fields that will be extracted from the API
    list_of_fields=["rn","destSt","destNm","trDr","nextStaId"
                    ,"nextStpId","nextStaNm","prdt","arrT","isApp","isDly","lat","lon"]
    # Checking of the error code from the API is 0 which indicates successful data retrieval
    if cta_json_data['ctatt']['errCd']=="0":
        # Extracting the Timestamp field
        cta_timestamp=cta_json_data['ctatt']['tmst']
        # Extracting the route information
        cta_routes=cta_json_data['ctatt']['route']
        # Extracting the route data
        cta_route_name=cta_routes[0]['@name']
        # Extracting the details of the train details
        cta_train=cta_routes[0]['train']
        # Getting the ctive number of trains
        cta_train_len=len(cta_train)
        # Creating a dummy dataframe with number of rows equal to number of active trains. The column
        # length is same as number of columns extracted from the API
        cta_df1=pd.DataFrame(np.random.rand(cta_train_len,len(list_of_fields)),columns=list_of_fields)
        # Adding new columns to the daraframe to contain the timestamp and Route name
        cta_df1["ROUTE_NAME"]=cta_route_name
        cta_df1["TIMESTAMP"]=cta_timestamp
        # Parsing each row and column in the dataframe and updating the values extracted from the API
        for idx1 in range(cta_train_len):
            # Looping through each field in the list of fields
            for idx2,field in enumerate(list_of_fields):
                #updating the Dataframe with the data extracted from API based on the Index positions
                cta_df1.iloc[idx1,idx2]=cta_train[idx1][field]
        # Formatting the dataframe by renaming columns
        cta_df1.rename(columns={"rn":"RUN_NUMBER",
                               "destSt":"DEST_STREET",
                               "destNm":"DEST_NAME",
                               "trDr":"TRAIN_ROUTE_NBR",
                               "nextStaId":"NEXT_STATION_ID",
                               "nextStpId":"NEXT_STOP_ID",
                               "nextStaNm":"NEXT_STATION_NAME",
                               "prdt":"PREDICTION_TS",
                               "arrT":"ARRIVAL_TS",
                               "isApp":"IS_APPROACHING",
                               "isDly":"IS_DELAYED",
                                "lat":"LATITUDE",
                                "lon":"LONGITUDE"
                               },inplace=True)
        # Converting the data to Upper case and stripping off the extra spaces.
        for col in cta_df1.columns:
            cta_df1[col]=cta_df1[col].apply(str.strip)
            cta_df1[col]=cta_df1[col].apply(str.upper)
```

```python
            cta_df1[col]=cta_df1[col].apply(str.upper)
        return cta_df1
    # If the errCd returned by the API is not 0, it indicates error
    else:
        print(f"Error Occurred: {cta_json_data['ctatt']['errNm']}")
        return None

[5]: def call_api():
    # Getting the API response
    cta_api_url=create_url()
    try:
        cta_url_response=urllib.request.urlopen(cta_api_url)
    # Handling the HTTPErrors if the movie details cannot be extracted or if page cannot be found
    except urllib.error.HTTPError as error1:
        print(f"Sorry could not retrieve the details of the movie {movie_name}")
    # Handling URLexceptions such as Incorrect URL or Internet connection issues
    except urllib.error.URLError as error2:
        print("Failed to reach the server")
        print(f"Reason: {error2.reason}" )
    # If no exceptions are found, data is extracted from the API response
    else:
        cta_url_data=cta_url_response.read()
        # The response is converted to json
        cta_json_data=json.loads(cta_url_data)
        cta_df1=extract_cta_data(cta_json_data)
        cta_df2=cta_df1[["ROUTE_NAME","RUN_NUMBER","DEST_STREET","DEST_NAME","NEXT_STATION_ID"
                        ,"NEXT_STATION_NAME","PREDICTION_TS","ARRIVAL_TS","IS_DELAYED",
                       "LATITUDE","LONGITUDE"]]
        #display(cta_df2)
        csv_dump_path=key_path+"/cta/cta_api_dump.csv"
        cta_df2.to_csv(csv_dump_path,mode="a",index=False,header=False)
        current_time = datetime.datetime.now()
        print(f"Successfully appended the output at {current_time}")

[8]: call_api()

The route used in the API is Pink
Successfully appended the output at 2024-02-23 02:36:23.161200

[ ]: import time
while True:
    # Code executed here
    time.sleep(10)
    call_api()
```

The python script that makes API calls during each execution was invoked every 10 seconds using the below module:

```
import time
while True:
    # Code executed here
    time.sleep(10)
    call_api()
"call_cta_api.py" 90L, 3732C
```

guruprasadvk10@bigdata-project:~/final_project/cta$ cat start_api.sh

#!/bin/bash

rm cta_api_dump.csv

echo "ROUTE_NAME,RUN_NUMBER,DEST_STREET,DEST_NAME,NEXT_STATION_ID,NEXT_STATION_NAME,PREDICTION_TS,ARRIVAL_TS,IS_DELAYED,LATITUDE,LONGITUDE" > cta_api_dump.csv

python3 call_cta_api.py

exit 0

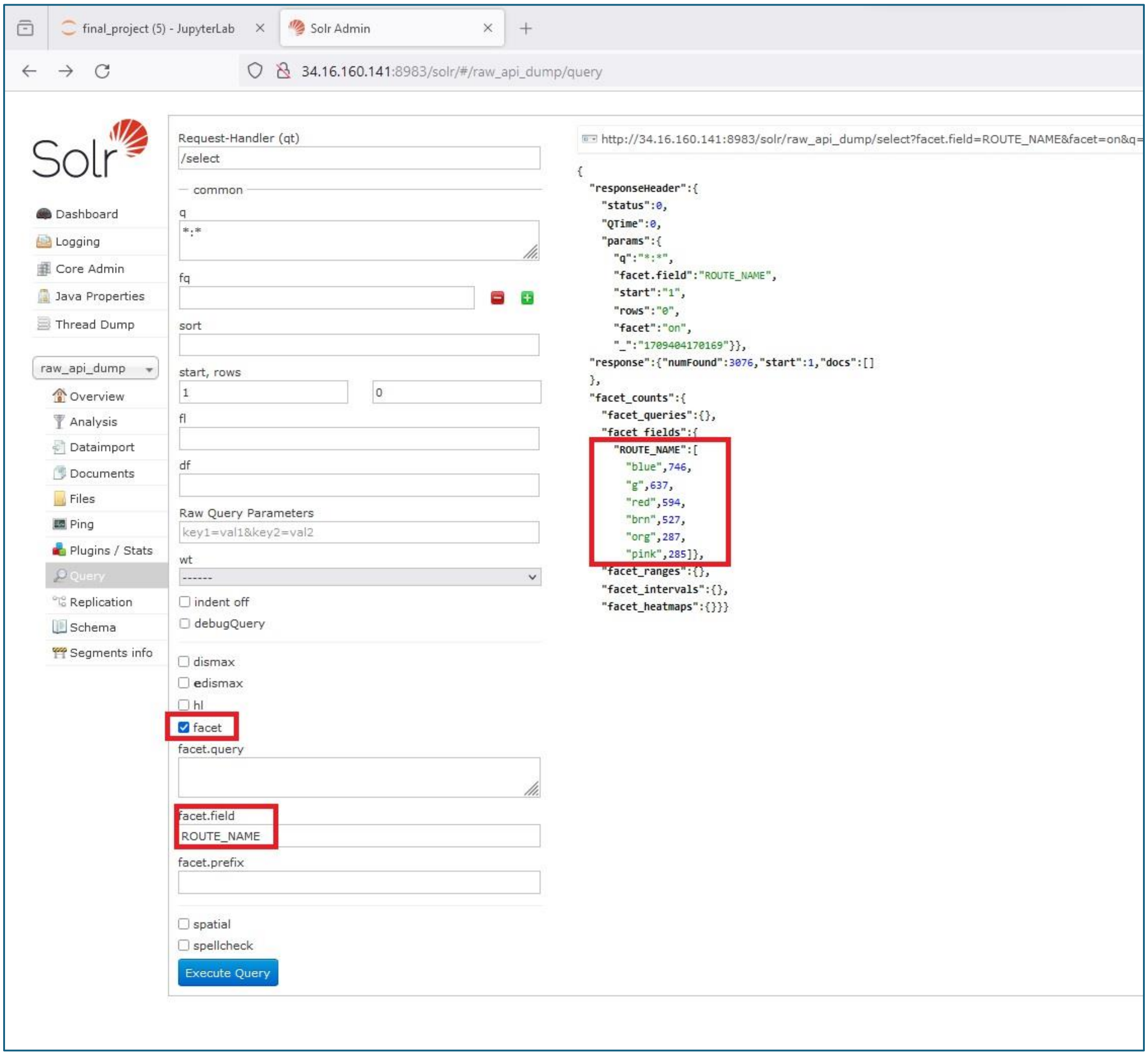
Search feature on Apache Solr:

Apache Solr version 8.2.0 was installed in the /opt directory and a collection named "raw_api_dump" was created to be able to query the API data.

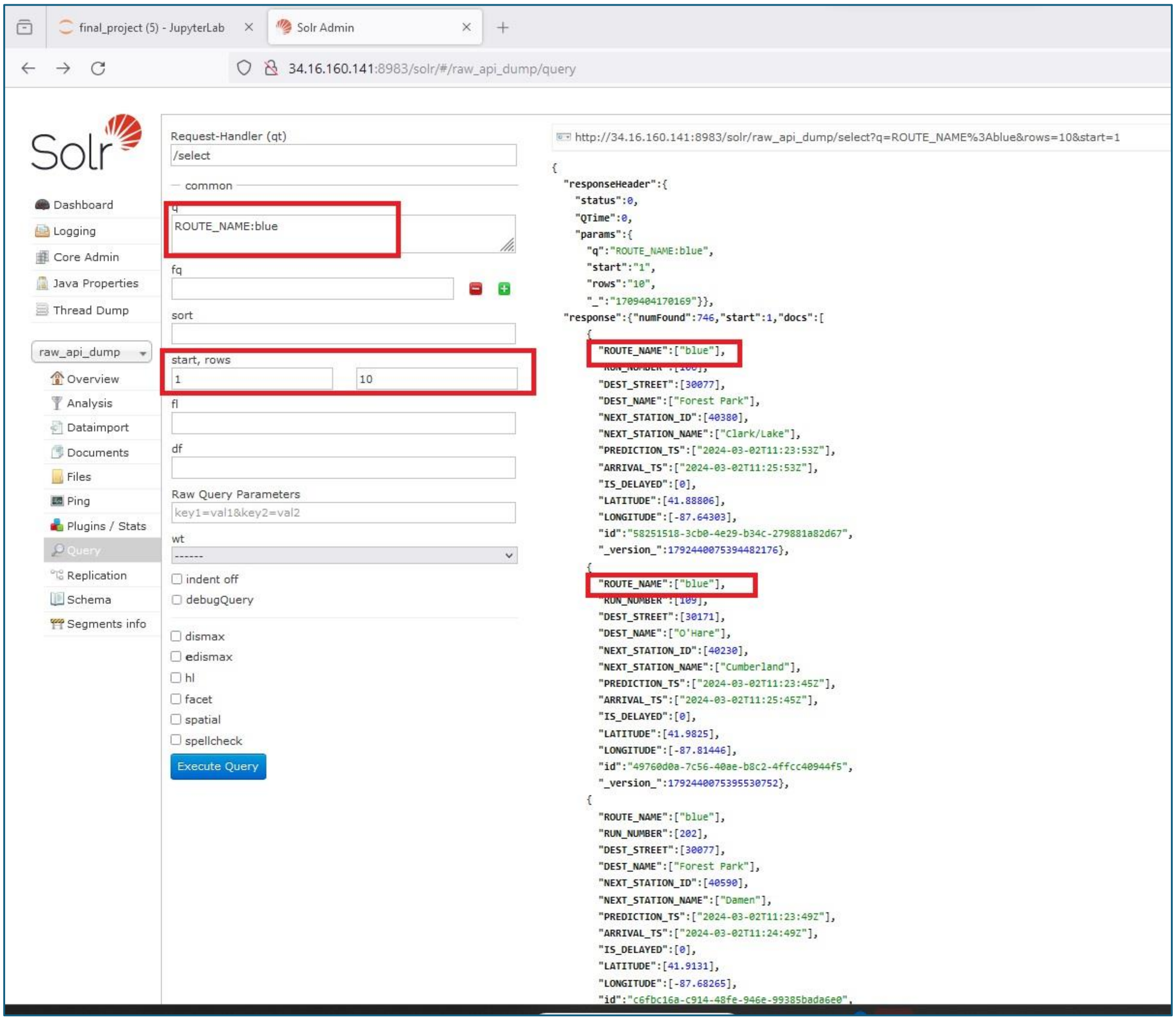
The collection "raw_api_dump " was created using the below commands:

- /opt/solr/bin/init.d/solr start → *To start the solr service*
- /opt/solr/bin/solr create -c raw_api_dump → *Creating collection*
- /opt/solr/bin/post -c raw_api_dump /home/guruprasadvk10/final_project/cta/cta_api_dump.csv → *Loading data to the collection.*

Screenshot of Collection using facet on the Route_color field



Screenshot of Collection using filter on Route_color= Blue



Module 2: Publishing data to Kafka using Kafka Connect:

Script name: 2_Read_from_API.ipynb

In this module, a new topic named ***cta_topic_kc*** was created and messages were published to the topic using Kafka connect that reads from the file ***cta_api_dump.csv.*** The Kafka_connect uses two configuration files for which screenshots are provided below:

I. *cta_file_source.properties*- This file contains the configuration for the source and the topic. The Source file type which is the “FileStreamSource” is included in the connector class. The “file” property includes the source file name and the topic name contains the name of the topic to which the messages are published.

II. *cta_file_standalone.properties:* This file contains the configuration of the kafka topic such as the offset file name, Port number, Bootstrap server details, etc.

```
guruprasadvk10@bigdata-project:~/final_project/kafka_connect/producer$ ls -ltr
total 12
-rw-rw-r-- 1 guruprasadvk10 guruprasadvk10 151 Feb 22 00:58 cta_file_source.properties
-rw-rw-r-- 1 guruprasadvk10 guruprasadvk10 229 Feb 25 16:29 retail.offsets
-rw-rw-r-- 1 guruprasadvk10 guruprasadvk10 385 Feb 25 17:14 cta_file_standalone.properties
guruprasadvk10@bigdata-project:~/final_project/kafka_connect/producer$ cat cta_file_source.properties
name=cta-data-file-source
connector.class=FileStreamSource
tasks.max=1
file=/home/guruprasadvk10/final_project/cta/cta_api_dump.csv
topic=cta_topic_kc
guruprasadvk10@bigdata-project:~/final_project/kafka_connect/producer$
guruprasadvk10@bigdata-project:~/final_project/kafka_connect/producer$
guruprasadvk10@bigdata-project:~/final_project/kafka_connect/producer$ cat cta_file_standalone.properties
bootstrap.servers=localhost:9092

key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=org.apache.kafka.connect.storage.StringConverter

key.converter.schemas.enable=true
value.converter.schemas.enable=true

offset.storage.file.filename=/home/guruprasadvk10/final_project/kafka_connect/producer/retail.offsets

offset.flush.interval.ms=30000

rest.port=18086
guruprasadvk10@bigdata-project:~/final_project/kafka_connect/producer$
guruprasadvk10@bigdata-project:~/final_project/kafka_connect/producer$
```

The below screenshot includes the code where the topic was created, and messages were published using kafka connect:

Using kafka connect to publish to the topic

```
[2]: # Deleting topic cta_topic_kc if it exists already
!kafka-topics.sh --delete \
  --zookeeper localhost:2181 \
  --topic cta_topic_kc
```

```
Topic cta_topic_kc is marked for deletion.
Note: This will have no impact if delete.topic.enable is not set to true.
```

```
[3]: # Creating topic cta_topic_kc with 1 Partition and replication factor as 1 as the code is run on a Single node cluster
!kafka-topics.sh --create \
  --zookeeper localhost:2181 \
  --replication-factor 1 \
  --partitions 1 \
  --topic cta_topic_kc
```

```
WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid issues it is best to use either, but not both.
Created topic cta_topic_kc.
```

```
[4]: # Listing the topics for validation
!kafka-topics.sh --list \
  --zookeeper localhost:2181
```

```
__consumer_offsets
cta_topic_kc
```

```
[ ]: # This is run from Console to publish data using Kafka Connect reading from the flat file
!/opt/kafka/bin/connect-standalone.sh \
    cta_file_standalone.properties \
    cta_file_source.properties
```

```
[*]: # Starting a Consumer to read messages from the topic cta_topic_kc
!/opt/kafka/bin/kafka-console-consumer.sh \
    --bootstrap-server localhost:9092 \
    --topic cta_topic_kc
```

```
brn,401,30249,Kimball,40530,Diversey,2024-02-25T11:29:57,2024-02-25T11:30:57,0,41.93273,-87.65313
brn,402,30249,Kimball,40730,Washington/Wells,2024-02-25T11:29:47,2024-02-25T11:31:47,0,41.8903,-87.63548
brn,403,30249,Loop,40530,Diversey,2024-02-25T11:29:57,2024-02-25T11:31:57,0,41.93975,-87.65338
brn,404,30249,Loop,41180,Kedzie,2024-02-25T11:29:56,2024-02-25T11:30:56,0,41.96606,-87.71187
brn,406,30249,Kimball,40460,Merchandise Mart,2024-02-25T11:29:55,2024-02-25T11:31:55,0,41.88574,-87.63089
pink,301,30114,Loop,40170,Ashland,2024-02-25T11:30:00,2024-02-25T11:31:00,0,41.88526,-87.66864
pink,302,30114,54th/Cermak,41040,Kedzie,2024-02-25T11:29:58,2024-02-25T11:30:58,0,41.85428,-87.69478
pink,304,30114,Loop,40150,Pulaski,2024-02-25T11:30:09,2024-02-25T11:31:09,0,41.85387,-87.72684
pink,306,30114,54th/Cermak,40160,LaSalle/Van Buren,2024-02-25T11:30:11,2024-02-25T11:31:11,0,41.87685,-87.6327
pink,307,30114,54th/Cermak,40420,Cicero,2024-02-25T11:30:15,2024-02-25T11:31:15,0,41.85272,-87.73648
```

Module 3: Consume Data from Kafka Using Spark Streaming and Exploring Hive:

Script name: 3_Spark_Streaming_and_Hive.ipynb

In this module, Spark session object was created as shown in the screenshot and messages were read from Kafka topic “cta_topic_kc” using readstream mode. The schema and properties of the read stream object were also explored.

```
[1]: # Importing SparkSession
     from pyspark.sql import SparkSession

[ ]: # Creating a SparkSession Object
     spark = SparkSession. \
         builder. \
         config('spark.jars.packages', 'org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.1'). \
         config('spark.ui.port', '0'). \
         config('spark.sql.warehouse.dir', f'/user/warehouse'). \
         enableHiveSupport(). \
         appName('Python - Kafka and Spark Integration for Spark Streaming for CTA Project'). \
         master('yarn'). \
         getOrCreate()

[3]: # Configuring the Bootstrap servers
     kafka_bootstrap_servers = 'localhost:9092'

[4]: # Creating an object for ReadStream
     df_cta = spark. \
       readStream. \
       format('kafka'). \
       option('kafka.bootstrap.servers', kafka_bootstrap_servers). \
       option('subscribe', 'cta_topic_kc'). \
       load()

[5]: # Validating if the Stream is active
     df_cta.isStreaming

[5]: True

[6]: # Printing the schema of the Stream dataframe
     df_cta.printSchema()

     root
      |-- key: binary (nullable = true)
      |-- value: binary (nullable = true)
      |-- topic: string (nullable = true)
      |-- partition: integer (nullable = true)
      |-- offset: long (nullable = true)
      |-- timestamp: timestamp (nullable = true)
      |-- timestampType: integer (nullable = true)

[ ]: # Using console mode to create a Write Stream object to write to the stream every 30 seconds.
     df_cta.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)"). \
         writeStream. \
         outputMode("update"). \
         format("console"). \
         option('truncate', 'false'). \
         trigger(processingTime='30 seconds'). \
         start()
```

Then a write stream objects using “Console” mode and “memory” mode were created. In the console mode, the data was written to the stream every 30 seconds. In the Memory mode, a query object named “df_cta_sql” was created to run queries using sql like syntax.

```
[ ]: # Using console mode to create a Write Stream object to write to the stream every 30 seconds.
     df_cta.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)"). \
         writeStream. \
         outputMode("update"). \
         format("console"). \
         option('truncate', 'false'). \
         trigger(processingTime='30 seconds'). \
         start()
```

```
[ ]: # Using Format mode to create a Write Stream object to write to the stream. Here query name object df_cta_sql is also created
     df_cta.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)"). \
         writeStream. \
         format("memory"). \
         queryName("df_cta_sql"). \
         start()
```

```
[8]: # Selecting count from the query name object
     spark.sql('SELECT count(1) FROM df_cta_sql').show()
```

```
+--------+
|count(1)|
+--------+
|      53|
+--------+
```

```
[9]: # Selecting data from the query name object
     spark.sql('SELECT * FROM df_cta_sql').show(truncate=False)
```

```
+----+------------------------------------------------------------------------------------------------------------------+
|key |value                                                                                                             |
+----+------------------------------------------------------------------------------------------------------------------+
|NULL|blue,102,30171,O'Hare,40750,Harlem (O'Hare Branch),2024-02-25T11:47:33,2024-02-25T11:49:33,0,41.98232,-87.8089    |
|NULL|blue,103,30171,O'Hare,40790,Monroe,2024-02-25T11:47:16,2024-02-25T11:48:16,0,41.87818,-87.6293                    |
|NULL|blue,104,30171,O'Hare,40010,Austin,2024-02-25T11:47:36,2024-02-25T11:49:36,0,41.87211,-87.7916                    |
|NULL|blue,107,30077,Forest Park,40370,Washington,2024-02-25T11:47:30,2024-02-25T11:48:30,0,41.88394,-87.62945          |
|NULL|blue,110,30077,Forest Park,40550,Irving Park,2024-02-25T11:47:29,2024-02-25T11:48:29,0,41.95655,-87.73564        |
|NULL|blue,202,30077,Forest Park,40920,Pulaski,2024-02-25T11:47:23,2024-02-25T11:50:23,0,41.87406,-87.71155            |
|NULL|blue,203,30077,Forest Park,41410,Chicago,2024-02-25T11:47:09,2024-02-25T11:49:09,0,41.90336,-87.6665             |
|NULL|blue,204,30077,Forest Park,40820,Rosemont,2024-02-25T11:47:35,2024-02-25T11:48:35,0,41.98337,-87.86345           |
|NULL|blue,207,30171,O'Hare,40670,Western (O'Hare Branch),2024-02-25T11:47:28,2024-02-25T11:48:28,0,41.90974,-87.67744|
|NULL|blue,209,30077,Forest Park,40010,Austin,2024-02-25T11:47:29,2024-02-25T11:48:29,0,41.87036,-87.76809            |
|NULL|blue,210,30171,O'Hare,40920,Pulaski,2024-02-25T11:47:21,2024-02-25T11:48:21,0,41.87365,-87.734                   |
|NULL|brn,401,30249,Kimball,41180,Kedzie,2024-02-25T11:47:57,2024-02-25T11:48:57,0,41.96614,-87.70297                  |
|NULL|brn,402,30249,Kimball,40800,Sedgwick,2024-02-25T11:47:58,2024-02-25T11:48:58,0,41.91041,-87.63886                |
|NULL|brn,403,30249,Kimball,40040,Quincy,2024-02-25T11:47:57,2024-02-25T11:48:57,0,41.88005,-87.63378                  |
|NULL|brn,404,30249,Loop,40530,Diversey,2024-02-25T11:47:53,2024-02-25T11:48:53,0,41.93759,-87.65332                   |
|NULL|brn,405,30249,Loop,40090,Damen,2024-02-25T11:47:24,2024-02-25T11:48:24,0,41.96625,-87.6885                       |
|NULL|brn,406,30249,Kimball,41320,Belmont,2024-02-25T11:47:47,2024-02-25T11:48:47,0,41.93817,-87.65335                 |
|NULL|g,001,30139,Cottage Grove,41400,Roosevelt,2024-02-25T11:48:09,2024-02-25T11:50:09,0,41.87296,-87.62678          |
|NULL|g,005,30004,Harlem/Lake,41400,Roosevelt,2024-02-25T11:48:13,2024-02-25T11:50:13,0,41.86004,-87.62647            |
|NULL|g,007,30057,Ashland/63rd,40170,Ashland,2024-02-25T11:47:17,2024-02-25T11:49:17,0,41.88498,-87.67667             |
+----+------------------------------------------------------------------------------------------------------------------+
only showing top 20 rows
```

New columns for route color, year, month, and dates were added to the query object of the write stream.

```
[10]: # Importing sql functions for processing
      from pyspark.sql.functions import lit, date_format, to_date, split, substring,unix_timestamp, from_unixtime
```

```
[11]: # Casting the columns as String and Adding new columns to extract Year, month and date information
      df_cta.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)"). \
          withColumn('route_color', split('value', ',')[0]). \
          withColumn('transit_date', to_date(split('value', ',')[6], "yyyy-MM-dd'T'HH:mm:ss")). \
          withColumn('year', date_format('transit_date', 'yyyy')). \
          withColumn('month', date_format('transit_date', 'MM')). \
          withColumn('dayofmonth', date_format('transit_date', 'dd')). \
          writeStream. \
          format("memory"). \
          queryName("df_cta_sql1"). \
          start()
```

```
24/02/25 17:50:51 WARN ResolveWriteToStream: Temporary checkpoint location created which is deleted normally when the query didn't fail: /tmp/temporary-ed956557-c837-4d6f-aa50-d412509bd46e.
If it's required to delete it under any circumstances, please set spark.sql.streaming.forceDeleteTempCheckpointLocation to true. Important to know deleting temp checkpoint folder is best eff
ort.
24/02/25 17:50:51 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Datasets and will be disabled.
```

```
[11]: <pyspark.sql.streaming.query.StreamingQuery at 0x7f6aa4260cd0>
```

```
[12]: # Selecting sample rows
      spark.sql('SELECT * FROM df_cta_sql1').show(truncate=False)
```

```
+----+-----------------------------------------------------------------------------------------------------------+-----------+------------+----+-----+----------+
|key |value                                                                                                      |route_color|transit_date|year|month|dayofmonth|
+----+-----------------------------------------------------------------------------------------------------------+-----------+------------+----+-----+----------+
|NULL|pink,301,30114,54th/Cermak,40170,Ashland,2024-02-25T11:48:49,2024-02-25T11:50:49,0,41.88558,-87.65217      |pink       |2024-02-25  |2024|02   |25        |
|NULL|pink,303,30114,Loop,40210,Damen,2024-02-25T11:48:58,2024-02-25T11:49:58,0,41.8544,-87.68513                |pink       |2024-02-25  |2024|02   |25        |
|NULL|pink,304,30114,54th/Cermak,40380,Clark/Lake,2024-02-25T11:48:50,2024-02-25T11:49:50,0,41.88571,-87.63742|pink       |2024-02-25  |2024|02   |25        |
|NULL|pink,306,30114,54th/Cermak,40740,Western,2024-02-25T11:48:09,2024-02-25T11:49:09,0,41.85451,-87.67597      |pink       |2024-02-25  |2024|02   |25        |
+----+-----------------------------------------------------------------------------------------------------------+-----------+------------+----+-----+----------+
```

New folders named data and checkpoint were created within “final_project_hdfs” folder in HDFS and spark streaming data was written to it in csv format as shown in the screenshot. The data was also partitioned using Year, month, and date.

Writing Spark Streaming data to HDFS

```
[15]: # Writing to HDFS and partitioning the data using Year, Month and Day columns. Data is written to HDFS every 30 seconds from the Spark stream.
df_cta.selectExpr("CAST(value AS STRING)"). \
    withColumn('route_color', split('value', ',')[0]). \
    withColumn('transit_date', to_date(split('value', ',')[6], "yyyy-MM-dd'T'HH:mm:ss")). \
    withColumn('year', date_format('transit_date', 'yyyy')). \
    withColumn('month', date_format('transit_date', 'MM')). \
    withColumn('dayofmonth', date_format('transit_date', 'dd')). \
    writeStream. \
    partitionBy('year', 'month', 'dayofmonth'). \
    format('csv'). \
    option("checkpointLocation", '/final_project_hdfs/checkpoint'). \
    option('path', '/final_project_hdfs/data'). \
    option('header',True). \
    option('sep','|'). \
    trigger(processingTime='30 seconds'). \
    start()

24/02/25 17:51:12 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Datasets and will be disabled.
[15]: <pyspark.sql.streaming.query.StreamingQuery at 0x7f6a7c20c160>

[16]: # Validating the HDFS directories
!hdfs dfs -ls /final_project_hdfs

Found 2 items
drwxr-xr-x   - guruprasadvk10 supergroup          0 2024-02-25 17:51 /final_project_hdfs/checkpoint
drwxr-xr-x   - guruprasadvk10 supergroup          0 2024-02-25 17:51 /final_project_hdfs/data

[18]: !hdfs dfs -ls -R /final_project_hdfs/data/year=2024

drwxr-xr-x   - guruprasadvk10 supergroup          0 2024-02-25 17:51 /final_project_hdfs/data/year=2024/month=02
drwxr-xr-x   - guruprasadvk10 supergroup          0 2024-02-25 17:51 /final_project_hdfs/data/year=2024/month=02/dayofmonth=25
-rw-r--r--   1 guruprasadvk10 supergroup       1619 2024-02-25 17:51 /final_project_hdfs/data/year=2024/month=02/dayofmonth=25/part-00000-b34ca1c2-84dc-417c-bedf-6cfda0519f8e.c000.csv

[19]: !hdfs dfs -ls /final_project_hdfs/checkpoint

Found 4 items
drwxr-xr-x   - guruprasadvk10 supergroup          0 2024-02-25 17:51 /final_project_hdfs/checkpoint/commits
-rw-r--r--   1 guruprasadvk10 supergroup         45 2024-02-25 17:51 /final_project_hdfs/checkpoint/metadata
drwxr-xr-x   - guruprasadvk10 supergroup          0 2024-02-25 17:51 /final_project_hdfs/checkpoint/offsets
drwxr-xr-x   - guruprasadvk10 supergroup          0 2024-02-25 17:51 /final_project_hdfs/checkpoint/sources

[20]: !hdfs dfs -cat /final_project_hdfs/checkpoint/sources/0/0

v1
{"cta_topic_kc":{"0":846}}

[21]: !hdfs dfs -ls /final_project_hdfs/checkpoint/offsets

Found 2 items
-rw-r--r--   1 guruprasadvk10 supergroup        667 2024-02-25 17:51 /final_project_hdfs/checkpoint/offsets/0
-rw-r--r--   1 guruprasadvk10 supergroup        667 2024-02-25 17:51 /final_project_hdfs/checkpoint/offsets/1
```

Spark Dataframes were created on the HDFS data and schema and data were validated as shown below:

```
[23]: # Validating the data written in HDFS by creating spark dataframe
df_from_hdfs1=spark.read.csv("/final_project_hdfs/data/",sep="|",header=True)

[24]: # Checking schema
df_from_hdfs1.printSchema()

root
 |-- value: string (nullable = true)
 |-- route_color: string (nullable = true)
 |-- transit_date: string (nullable = true)
 |-- year: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- dayofmonth: integer (nullable = true)

[25]: # CHecking sample rows
df_from_hdfs1.show(truncate=False)

+-----------------------------------------------------------------------------------------------------+-----------+------------+----+-----+----------+
|value                                                                                                |route_color|transit_date|year|month|dayofmonth|
+-----------------------------------------------------------------------------------------------------+-----------+------------+----+-----+----------+
|org,701,30182,Loop,40960,Pulaski,2024-02-25T11:49:15,2024-02-25T11:50:15,0,41.79497,-87.73636        |org        |2024-02-25  |2024|2    |25        |
|org,703,30182,Midway,41060,Ashland,2024-02-25T11:49:16,2024-02-25T11:51:16,0,41.84689,-87.64815      |org        |2024-02-25  |2024|2    |25        |
|org,704,30182,Midway,40040,Quincy,2024-02-25T11:49:17,2024-02-25T11:50:17,0,41.87698,-87.63366       |org        |2024-02-25  |2024|2    |25        |
|org,706,30182,Loop,41130,Halsted,2024-02-25T11:49:13,2024-02-25T11:50:13,0,41.8413,-87.66123         |org        |2024-02-25  |2024|2    |25        |
|org,707,30182,Midway,40310,Western,2024-02-25T11:49:12,2024-02-25T11:50:12,0,41.8152,-87.67988       |org        |2024-02-25  |2024|2    |25        |
|red,801,30089,95th/Dan Ryan,41220,Fullerton,2024-02-25T11:49:29,2024-02-25T11:50:29,0,41.93019,-87.65302|red        |2024-02-25  |2024|2    |25        |
|red,802,30089,95th/Dan Ryan,40080,Sheridan,2024-02-25T11:49:38,2024-02-25T11:51:38,0,41.95377,-87.65493 |red        |2024-02-25  |2024|2    |25        |
|red,810,30089,95th/Dan Ryan,40240,79th,2024-02-25T11:49:37,2024-02-25T11:50:37,0,41.75381,-87.62537     |red        |2024-02-25  |2024|2    |25        |
|red,811,30089,95th/Dan Ryan,41300,Loyola,2024-02-25T11:48:46,2024-02-25T11:51:46,0,42.00836,-87.66591   |red        |2024-02-25  |2024|2    |25        |
|red,901,30173,Howard,41230,47th,2024-02-25T11:49:25,2024-02-25T11:50:25,0,41.80854,-87.63177          |red        |2024-02-25  |2024|2    |25        |
|red,904,30089,95th/Dan Ryan,41170,Garfield,2024-02-25T11:49:25,2024-02-25T11:50:25,0,41.80808,-87.6318  |red        |2024-02-25  |2024|2    |25        |
|red,906,30089,95th/Dan Ryan,41660,Lake,2024-02-25T11:49:35,2024-02-25T11:50:35,0,41.88709,-87.62789     |red        |2024-02-25  |2024|2    |25        |
|red,907,30173,Howard,41200,Argyle,2024-02-25T11:48:54,2024-02-25T11:51:54,0,41.96427,-87.65759        |red        |2024-02-25  |2024|2    |25        |
|red,908,30173,Howard,40630,Clark/Division,2024-02-25T11:49:28,2024-02-25T11:51:28,0,41.89667,-87.62818 |red        |2024-02-25  |2024|2    |25        |
+-----------------------------------------------------------------------------------------------------+-----------+------------+----+-----+----------+
```

Hive tables were created on the HDFS data in the path /final_project_hdfs/data/year=2024/month=02/dayofmonth=25/. As shown in the screenshot, a new table cta_data was created and its schema was defined before loading HDFS data into it.

Creating tables in HIVE and querying the tables

[26]:
```
# Checking list of databases in Hive
spark.sql("show databases")
```

```
24/02/25 17:52:12 WARN HiveConf: HiveConf of name hive.stats.jdbc.timeout does not exist
24/02/25 17:52:12 WARN HiveConf: HiveConf of name hive.stats.retries.wait does not exist
```

[26]:

namespace
cta_db
default
retail_db

[28]:
```
# Using the cta_db created for this project
spark.sql("use cta_db")
```

```
24/02/25 17:52:23 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
```

[28]:

[42]:
```
# Drop the table cta_data is it exists already
spark.sql("DROP TABLE IF EXISTS cta_data;")
```

[42]:

•[43]:
```
# Creating Hive table structure
spark.sql("CREATE TABLE cta_data (    value STRING,    route_color  STRING,    transit_date STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|';")
```

```
24/02/25 18:44:28 WARN HiveMetaStore: Location: hdfs://localhost:9000/user/hive/warehouse/cta_db.db/cta_data specified for non-external table:cta_data
```

[43]:

[44]:
```
# Loading data into Hive table using HDFS data
spark.sql("LOAD DATA INPATH '/final_project_hdfs/data/year=2024/month=02/dayofmonth=25/' INTO TABLE cta_data;")
#spark.sql("LOAD DATA INPATH '/final_project_hdfs/data/' INTO TABLE cta_data;")
```

[44]:

SQL Queries were run on the Hive table cta_data to check the counts and sample data with filters.

•[44]:
```
# Loading data into Hive table using HDFS data
spark.sql("LOAD DATA INPATH '/final_project_hdfs/data/year=2024/month=02/dayofmonth=25/' INTO TABLE cta_data;")
```

[44]:

[45]:
```
# Selecting sample rows from cta_data tables
spark.sql("SELECT * FROM cta_data LIMIT 5;")
```

[45]:

value	route_color	transit_date
value	route_color	transit_date
g,001,30139,Cotta...	g	2024-02-25
g,005,30004,Harle...	g	2024-02-25
g,007,30057,Ashla...	g	2024-02-25
g,008,30057,Ashla...	g	2024-02-25

[46]:
```
# Selecting count of rows from cta_data tables
spark.sql("SELECT count(1) FROM cta_data;")
```

[46]:

count(1)
2350

[47]:
```
spark.sql("SELECT * FROM cta_data where route_color='g';")
```

[47]:

value	route_color	transit_date
g,001,30139,Cotta...	g	2024-02-25
g,005,30004,Harle...	g	2024-02-25
g,007,30057,Ashla...	g	2024-02-25
g,008,30057,Ashla...	g	2024-02-25
g,601,30004,Harle...	g	2024-02-25
g,603,30004,Harle...	g	2024-02-25
g,605,30139,Cotta...	g	2024-02-25
g,608,30004,Harle...	g	2024-02-25
g,001,30139,Cotta...	g	2024-02-25
g,002,30057,Ashla...	g	2024-02-25
g,005,30004,Harle...	g	2024-02-25
g,007,30057,Ashla...	g	2024-02-25

Module 4: Building a Machine Learning Model:

Script name: 4_Building_ML_models.ipynb

In this module, Machine Learning Classification model was built using Logistic Regression algorithm. The streaming data contains a field called “is_delayed” which indicates if a train is delayed or not. The classification model was built using the features such as 'route_color', 'run_number', 'dest_name', 'next_station_id', 'next_station_name' , year','month','day','hour' and 'minute' data to predict if the train will be delayed or not. The model will create a binary output- 0 indicating no delay and 1 indicating a delay.

Same as the previous module, a Read stream object was created and the data was then written to HDFS path /final_project_hdfs/ml/data for historical analysis. Also, Hive tables were built on the data.

Writing ML streaming data to HDFS

```
[57]: #!hdfs dfs -rm -R -skipTrash /final_project_hdfs/ml
```

```
[58]: # Writing the data into HDFS in a new path /final_project_hdfs/ml/data
df_cta_readstream.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)"). \
    withColumn('route_color', split('value', ',')[0]). \
    withColumn('run_number', split('value', ',')[1]). \
    withColumn('dest_name', split('value', ',')[3]). \
    withColumn('next_station_id', split('value', ',')[4]). \
    withColumn('next_station_name', split('value', ',')[5]). \
    withColumn('lat', split('value', ',')[9]). \
    withColumn('lon', split('value', ',')[10]). \
    withColumn('transit_date', to_date(split('value', ',')[6], "yyyy-MM-dd'T'HH:mm:ss")). \
    withColumn('year', date_format('transit_date', 'yyyy')). \
    withColumn('month', date_format('transit_date', 'MM')). \
    withColumn('dayofmonth', date_format('transit_date', 'dd')). \
    withColumn('hour', date_format('transit_date', 'HH')). \
    withColumn('minute', date_format('transit_date', 'mm')). \
    withColumn('is_delayed', split('value', ',')[8]). \
    writeStream. \
    format('csv'). \
    option("checkpointLocation", '/final_project_hdfs/ml/checkpoint'). \
    option('path', '/final_project_hdfs/ml/data'). \
    option('header',True). \
    option('sep','|'). \
    trigger(processingTime='10 seconds'). \
    start()

24/03/02 19:20:53 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Datasets and will be disabled.
24/03/02 19:20:53 WARN StreamingQueryManager: Stopping existing streaming query [id=2d9d4a30-cfe0-4117-920c-b929b0101318, runId=bc173ec8-b4f4-41dc-8ee3-809d3884c9c5], as a new run is being s
tarted.

[58]: <pyspark.sql.streaming.query.StreamingQuery at 0x7f62845438b0>
```

```
[54]: !hdfs dfs -ls /final_project_hdfs/ml/data

Found 416 items
drwxr-xr-x   - guruprasadvk10 supergroup          0 2024-03-02 18:39 /final_project_hdfs/ml/data/_spark_metadata
-rw-r--r--   1 guruprasadvk10 supergroup       1987 2024-03-02 17:36 /final_project_hdfs/ml/data/part-00000-0037e94a-92e3-4818-aad9-d2ea1a2611c6-c000.csv
-rw-r--r--   1 guruprasadvk10 supergroup       1209 2024-03-02 18:36 /final_project_hdfs/ml/data/part-00000-00b52e82-8d89-42a4-bcac-6ac4d2c2effa-c000.csv
-rw-r--r--   1 guruprasadvk10 supergroup       2031 2024-03-02 17:50 /final_project_hdfs/ml/data/part-00000-00c8e9b3-e262-4d1f-9fec-343ad32e854a-c000.csv
-rw-r--r--   1 guruprasadvk10 supergroup       1225 2024-03-02 18:12 /final_project_hdfs/ml/data/part-00000-01ffbd6c-82d8-459e-b9ad-db994f84299f-c000.csv
-rw-r--r--   1 guruprasadvk10 supergroup       2383 2024-03-02 18:17 /final_project_hdfs/ml/data/part-00000-026e023a-4e8f-482d-998f-1630950d0a02-c000.csv
-rw-r--r--   1 guruprasadvk10 supergroup       2162 2024-03-02 18:16 /final_project_hdfs/ml/data/part-00000-04d00dfe-e6f0-41d5-a215-478eba2192eb-c000.csv
-rw-r--r--   1 guruprasadvk10 supergroup       2160 2024-03-02 18:30 /final_project_hdfs/ml/data/part-00000-04f70095-cf28-43fe-b8b5-8d5fc84032d1-c000.csv
-rw-r--r--   1 guruprasadvk10 supergroup       2136 2024-03-02 18:26 /final_project_hdfs/ml/data/part-00000-05f123ea-c165-4040-b719-49c108a4445a-c000.csv
-rw-r--r--   1 guruprasadvk10 supergroup       2226 2024-03-02 18:36 /final_project_hdfs/ml/data/part-00000-06253993-374f-4ed8-a043-3e19432354a3-c000.csv
-rw-r--r--   1 guruprasadvk10 supergroup       1995 2024-03-02 17:56 /final_project_hdfs/ml/data/part-00000-06505f64-e79e-41e7-aad3-c39a234b5b9e-c000.csv
-rw-r--r--   1 guruprasadvk10 supergroup       1026 2024-03-02 17:43 /final_project_hdfs/ml/data/part-00000-07399473-34d7-4adf-b0d9-294da1ab9c12-c000.csv
-rw-r--r--   1 guruprasadvk10 supergroup       2162 2024-03-02 17:47 /final_project_hdfs/ml/data/part-00000-0790a502-9533-44d4-9bdb-ac5110f10352-c000.csv
-rw-r--r--   1 guruprasadvk10 supergroup       1416 2024-03-02 18:08 /final_project_hdfs/ml/data/part-00000-0859e785-6de9-47cd-9436-1346deab46e8-c000.csv
-rw-r--r--   1 guruprasadvk10 supergroup       1408 2024-03-02 18:34 /final_project_hdfs/ml/data/part-00000-08c8e84a-6c9a-44ff-846a-393c956963a4-c000.csv
```

Hive Table named cta_ml_data built on the Data Stream that was stored in the HDFS.

Creating Hive tables on the ML streaming data stored in the HDFS

[59]:
```
# Selecting the Hive DB
spark.sql("use cta_db")
# Drop table if it already exists
spark.sql("DROP TABLE IF EXISTS cta_ml_data;")
```

[59]:

[60]:
```
# Create HIve table cta_ml_data
spark.sql("CREATE TABLE cta_ml_data (   value STRING,    route_color  STRING,  run_number STRING,  dest_name STRING, next_station_id STRING, next_station_name STRING, \
         lat STRING, lon STRING , transit_date STRING,  year STRING, month STRING, dayofmonth STRING, hour STRING , minute STRING, is_delayed STRING ) ROW FORMAT DELIMITED FIELDS TERMINATED
# Load data into Hive from HDFS path
spark.sql("LOAD DATA INPATH '/final_project_hdfs/ml/data' INTO TABLE cta_ml_data;")
```

```
24/03/02 19:21:05 WARN HiveMetaStore: Location: hdfs://localhost:9000/user/hive/warehouse/cta_db.db/cta_ml_data specified for non-external table:cta_ml_data
```

[60]:

[61]:
```
# Running sample queries on hive table
spark.sql("SELECT * FROM cta_ml_data  order by run_number LIMIT 5;")
```

[61]:

value	route_color	run_number	dest_name	next_station_id	next_station_name	lat	lon	transit_date	year	month	dayofmonth	hour	minute	is_delayed
	blue,110,30077,Fo...	blue	110	Forest Park	40590	Damen	41.91427	-87.68446	2024-03-02	2024	03	02	00	00
	blue,112,30171,O'...	blue	112	O'Hare	40010	Austin	41.87109	-87.77971	2024-03-02	2024	03	02	00	00
	blue,107,30077,Fo...	blue	107	Forest Park	40550	Irving Park	41.96149	-87.74362	2024-03-02	2024	03	02	00	00
	blue,203,30077,Fo...	blue	203	Forest Park	40370	Washington	41.88471	-87.62946	2024-03-02	2024	03	02	00	00
	blue,113,30077,Fo...	blue	113	Forest Park	41340	LaSalle	41.87818	-87.6293	2024-03-02	2024	03	02	00	00

[62]:
```
spark.sql("SELECT count(1) FROM cta_ml_data;")
```

[62]:

count(1)
6089

Then the formatting of the data such as String Indexing, One hot encoding to convert the categorical variables into numbers and Vector creation were done in the next step.

Building Machine Learning Model

[64]:
```
# Exporting required libraries for building the model
from pyspark.ml.feature import OneHotEncoder
from pyspark.ml.feature import MinMaxScaler
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import OneHotEncoder
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline
```

[65]:
```
# Creating list of numeric and Categorical columns
catCols=['route_color', 'run_number', 'dest_name', 'next_station_id', 'next_station_name']
numCols=['year','month','day','hour','minute']
print(catCols)
print(numCols)
```

```
['route_color', 'run_number', 'dest_name', 'next_station_id', 'next_station_name']
['year', 'month', 'day', 'hour', 'minute']
```

[66]:
```
# Creating a String Indexer object on the categorical columns
string_indexer=[
    StringIndexer(inputCol=x, outputCol=x+"_StringIndexer",handleInvalid="skip")
    for x in catCols
]
string_indexer
```

[66]:
```
[StringIndexer_c26df6e97feb,
 StringIndexer_9d39cf756252,
 StringIndexer_06f5bdd3275e,
 StringIndexer_42c199204d62,
 StringIndexer_33293fef0160]
```

[67]:
```
# Creating a One Hot Encoding object on the categorical columns
one_hot_encoder=[
    OneHotEncoder(inputCols=[f"{x}_StringIndexer" for x in catCols],
                  outputCols=[f"{x}_OneHotEncoder" for x in catCols],
                 )
]
one_hot_encoder
```

[67]:
```
[OneHotEncoder_38144cd3c1ed]
```

[68]:
```
# Creating an assembler object
assemblerInput=[x for x in numCols]
assemblerInput += [f"{x}_OneHotEncoder" for x in catCols]
assemblerInput
```

```
[69]: # Creating a Vector object on the assembler object as ML can accept the features as a Vector
vector_assembler=VectorAssembler(
    inputCols=assemblerInput,outputCol="vectorAssembler_features"
)
```

```
[70]: # Creating a list called stages that contains the stages in the pipeline
stages=[]
stages += string_indexer
stages += one_hot_encoder
stages += [vector_assembler]
stages
```

```
[70]: [StringIndexer_c26df6e97feb,
 StringIndexer_9d39cf756252,
 StringIndexer_06f5bdd3275e,
 StringIndexer_42c199204d62,
 StringIndexer_33293fef0160,
 OneHotEncoder_38144cd3c1ed,
 VectorAssembler_08e6c845c54f]
```

```
[71]: # Creating a pipeline from the stages list
from pyspark.ml import Pipeline
pipeline= Pipeline().setStages(stages)
```

```
[72]: # Creating a temporary Dataframe extracting the data from df_cta_sql_ml
temp_df=spark.sql("select CAST(value AS STRING), \
                  split(value, ',')[0] as route_color, \
                  split(value, ',')[1] as run_number, \
                  split(value, ',')[3] as dest_name, \
                  split(value, ',')[4] as next_station_id, \
                  split(value, ',')[5] as next_station_name, \
                  split(value, ',')[8] as is_delayed, \
                  to_date(split(split(value, ',')[6],'T')[0],'yyyy-MM-dd') as transit_date, \
                  year(transit_date) as year, \
                  month(transit_date) as month, \
                  day(transit_date) as day, \
                  to_date(split(split(value, ',')[6],'T')[1],'HH:mm:ss') as transit_time, \
                  hour(transit_time) as hour, \
                  minute(transit_time) as minute \
                  from df_cta_sql_ml"
         )
# Selecting the required columns
ml_df_orig=temp_df.select("route_color","run_number","dest_name","next_station_id","next_station_name","year","month","day","hour","minute","is_delayed")
# Renaming the column is_delayed as output
ml_df = ml_df_orig.withColumnRenamed("is_delayed","output")
```

```
[73]: ml_df.count()
```

```
[73]: 5576
```

The Pipeline object was applied on the ml_df to convert the categorical columns into Numeric features as vectors that the ML algorithm can process.

```
[76]: # Fitting  and transforming the pipeline object on the ml_df
ml_df_fit=pipeline.fit(ml_df)
ml_df_trans=ml_df_fit.transform(ml_df)
# Selecting the required columns to display
ml_df_trans.select('route_color', 'run_number', 'dest_name', 'next_station_id', 'next_station_name','year','month','day','hour','minute','vectorAssembler_features').show(5,truncate=False)
```

```
+-----------+----------+------------+---------------+-----------------+----+-----+---+----+------+-------------------------------------------------------------------+
|route_color|run_number|dest_name   |next_station_id|next_station_name|year|month|day|hour|minute|vectorAssembler_features                                           |
+-----------+----------+------------+---------------+-----------------+----+-----+---+----+------+-------------------------------------------------------------------+
|g          |001       |Ashland/63rd|41160          |Clinton          |2024|3    |2  |0   |0     |(333,[0,1,2,7,34,103,110,241],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|
|g          |004       |Harlem/Lake |41350          |Oak Park         |2024|3    |2  |0   |0     |(333,[0,1,2,7,66,99,226,266],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0]) |
|g          |005       |Harlem/Lake |41400          |Roosevelt        |2024|3    |2  |0   |0     |(333,[0,1,2,7,40,99,104,234],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0]) |
|g          |006       |Harlem/Lake |40030          |Pulaski          |2024|3    |2  |0   |0     |(333,[0,1,2,7,88,99,147,236],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0]) |
|g          |007       |Ashland/63rd|41080          |47th             |2024|3    |2  |0   |0     |(333,[0,1,2,7,50,103,141,255],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|
+-----------+----------+------------+---------------+-----------------+----+-----+---+----+------+-------------------------------------------------------------------+
only showing top 5 rows
```

```
[77]: # creating an objet called Data that contains the features and output
from pyspark.sql.functions import col
data=ml_df_trans.select(
    col("vectorAssembler_features").alias("features"),
    col("output").alias("label")
)
```

```
[78]: # Casting the datatype of output column as Integer
data_df=data.selectExpr("features","cast(label as int) label")
data_df.show(5,truncate=False)
```

```
+-------------------------------------------------------------------+-----+
|features                                                           |label|
+-------------------------------------------------------------------+-----+
|(333,[0,1,2,7,34,103,110,241],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |
|(333,[0,1,2,7,66,99,226,266],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0]) |0    |
|(333,[0,1,2,7,40,99,104,234],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0]) |0    |
|(333,[0,1,2,7,88,99,147,236],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0]) |0    |
|(333,[0,1,2,7,50,103,141,255],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |
+-------------------------------------------------------------------+-----+
only showing top 5 rows
```

```
[79]: # Printing the schema of the data frame on which the ML model will be built
data_df.printSchema()
```

```
root
 |-- features: vector (nullable = true)
 |-- label: integer (nullable = true)
```

The Dataset was then split into train and test sets. The Train dataset was trained using the Logistic regression model and was used to predict the outcome using the test set. The column "is_delayed" was used as the output column for the model.

Building a Logistic Regression Model

```
[80]: # Split the dataset into Test and Train sets based on 30:70 ratio
      test_df, train_df = data_df.randomSplit([0.3, 0.7])

[81]: # Creating a Logistic regression object
      lr=LogisticRegression(maxIter=10, regParam= 0.01)

[82]: # Training the Logistic regression Model using the tran dataset
      model=lr.fit(train_df)
      # Creating an object for the Model summary
      trainingSummary = model.summary
      # Printing the Model accuracy
      trainingSummary.accuracy

[82]: 0.9918388166284111

[83]: # Creating an object to store the Model predictions
      predictions = model.transform(test_df)
      # Printing sample rows from the predictions
      #predictions.select("prediction","label","features").show(10)
      predictions.show(truncate=False)
```

```
+------------------------------------------------------------+-----+------------------------------------------+-----------------------------------------+----------+
|features                                                    |label|rawPrediction                             |probability                              |prediction|
+------------------------------------------------------------+-----+------------------------------------------+-----------------------------------------+----------+
|(333,[0,1,2,5,14,95,113,246],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[6.6549711900460915,-6.6549711900460915]|[0.998714050556637,0.0012859494433630214] |0.0       |
|(333,[0,1,2,5,14,95,119,262],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[6.647070912240843,-6.647070912240843]  |[0.9987038641820852,0.0012961358179147675]|0.0       |
|(333,[0,1,2,5,14,95,119,262],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[6.647070912240843,-6.647070912240843]  |[0.9987038641820852,0.0012961358179147675]|0.0       |
|(333,[0,1,2,5,14,95,119,262],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[6.647070912240843,-6.647070912240843]  |[0.9987038641820852,0.0012961358179147675]|0.0       |
|(333,[0,1,2,5,14,95,119,262],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[6.647070912240843,-6.647070912240843]  |[0.9987038641820852,0.0012961358179147675]|0.0       |
|(333,[0,1,2,5,14,95,119,262],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[6.647070912240843,-6.647070912240843]  |[0.9987038641820852,0.0012961358179147675]|0.0       |
|(333,[0,1,2,5,14,95,119,262],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[6.647070912240843,-6.647070912240843]  |[0.9987038641820852,0.0012961358179147675]|0.0       |
|(333,[0,1,2,5,14,95,122,242],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[6.260989814017458,-6.260989814017458]  |[0.998094283734097,0.00190571626590296 8] |0.0       |
|(333,[0,1,2,5,14,95,132,236],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[6.509372155971109,-6.509372155971109]  |[0.9985128005069654,0.0014871994930345522]|0.0       |
|(333,[0,1,2,5,14,95,135,273],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[2.7208323214250836,-2.7208323214250836]|[0.9382447772954524,0.061755222704547585] |0.0       |
|(333,[0,1,2,5,14,95,148,266],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[6.368376252858188,-6.368376252858188]  |[0.9982879943920949,0.00171200560790507 6]|0.0       |
|(333,[0,1,2,5,14,95,148,266],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[6.368376252858188,-6.368376252858188]  |[0.9982879943920949,0.00171200560790507 6]|0.0       |
|(333,[0,1,2,5,14,95,148,266],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[6.368376252858188,-6.368376252858188]  |[0.9982879943920949,0.00171200560790507 6]|0.0       |
|(333,[0,1,2,5,14,95,159,289],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[6.708378449165139,-6.708378449165139]  |[0.9987808462965453,0.0012191537034547117]|0.0       |
|(333,[0,1,2,5,14,95,164,291],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[6.80900647711315,-6.80900647711315]    |[0.9988974280118282,0.0011025719881717633]|0.0       |
|(333,[0,1,2,5,14,95,168,296],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[6.6069336836768136,-6.6069336836768136]|[0.9986508543520537,0.0013491456479463038]|0.0       |
|(333,[0,1,2,5,20,97,105,238],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[0.9753255912981977,-0.9753255912981977]|[0.7261797274911406,0.2738202725088594 3] |0.0       |
|(333,[0,1,2,5,20,97,142,277],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[5.040688755664033,-5.040688755664033]  |[0.993572291693009,0.0064277083069910 2]  |0.0       |
|(333,[0,1,2,5,20,97,142,277],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[5.040688755664033,-5.040688755664033]  |[0.993572291693009,0.0064277083069910 2]  |0.0       |
|(333,[0,1,2,5,20,97,142,277],[2024.0,3.0,2.0,1.0,1.0,1.0,1.0,1.0])|0    |[5.040688755664033,-5.040688755664033]  |[0.993572291693009,0.0064277083069910 2]  |0.0       |
+------------------------------------------------------------+-----+------------------------------------------+-----------------------------------------+----------+
only showing top 20 rows
```

Based on the model results, model accuracy, predictions , Area Under ROC were all calculated. The Prediction results were also stored in the dataframe and displayed. The prediction data was also stored in the HDFS in Parquet format, on which Hive table called "prediction results" was calculated.

```
[86]: predictions.count()

[86]: 1669
```

Storing Model results in HDFS and querying using Hive

```
[87]: # Saving the dataframe data in HDFS in parquet format
      !hdfs dfs -rm -R -skipTrash /model_results
      predictions.write.format("parquet").save("/model_results/")

      Deleted /model_results

[88]: # CHecking the contents of HDFS path
      !hdfs dfs -ls /model_results

      Found 3 items
      -rw-r--r--   1 guruprasadvk10 supergroup          0 2024-03-02 19:22 /model_results/_SUCCESS
      -rw-r--r--   1 guruprasadvk10 supergroup      50083 2024-03-02 19:22 /model_results/part-00000-9b06e66e-5e3c-4a3a-8e51-dd7b77835a7c-c000.snappy.parquet
      -rw-r--r--   1 guruprasadvk10 supergroup      50265 2024-03-02 19:22 /model_results/part-00001-9b06e66e-5e3c-4a3a-8e51-dd7b77835a7c-c000.snappy.parquet

[89]: # Dropping the hive table if it already exists
      spark.sql("use cta_db")
      spark.sql("DROP TABLE IF EXISTS prediction_results;")

[89]:

[90]: spark.sql("CREATE EXTERNAL TABLE prediction_results( features STRING) STORED AS PARQUET LOCATION '/model_results/';")
      # Selecting count of rows
      spark.sql("SELECT count(1) FROM prediction_results;")

[90]: count(1)
      1669
```

Module 5: Exploring Spark Streaming further

Script name: 5_Exploring_Spark_Streaming.ipynb

In this module different modes of spark streaming such as append, complete and output modes were explored. This is an alternate approach for Spark streaming without using Kafka, where the data from the API was streamed to a port on the host to simulate a web server. Spark streaming API was used to read the messages from that port in the “socket” format.

```
[1]: # Importing Spark session object
     from pyspark.sql import SparkSession
```

```
[2]: # Creating a spark session object
     spark = SparkSession. \
         builder. \
         enableHiveSupport(). \
         appName('Demo'). \
         master('yarn'). \
         getOrCreate()
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/02/25 21:36:17 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

```
[3]: # Setting the number of shuffle partition as 2 by overriding the default
     spark.conf.set('spark.sql.shuffle.partitions', '2')
```

Spark Streaming reading messages from port 9100

```
[ ]: # This command is run from the console where the content from the file cta_api_dump.csv is streamed to a web server on the port 9100
     !sh tail_api.sh|nc -lk `hostname -f` 9100
```

```
[4]: # Importing the required libraries such as socket
     import socket
     hostname = socket.gethostname()
     hostname

[4]: 'bigdata-project'
```

```
[5]: # Creating a read stream by reading the messages from port 9101
     api_messages = spark. \
         readStream. \
         format("socket"). \
         option("host", hostname). \
         option("port", 9101). \
         load()
```

```
24/02/25 21:36:40 WARN TextSocketSourceProvider: The socket source should not be used for production applications! It does not support recovery.
```

```
[6]: api_messages.isStreaming

[6]: True
```

```
[7]: api_messages.printSchema()

     root
      |-- value: string (nullable = true)
```

Using Append mode(default mode) to create a write stream object to print the messages to the console. As shown in the screenshot, the messages from each batch are displayed.

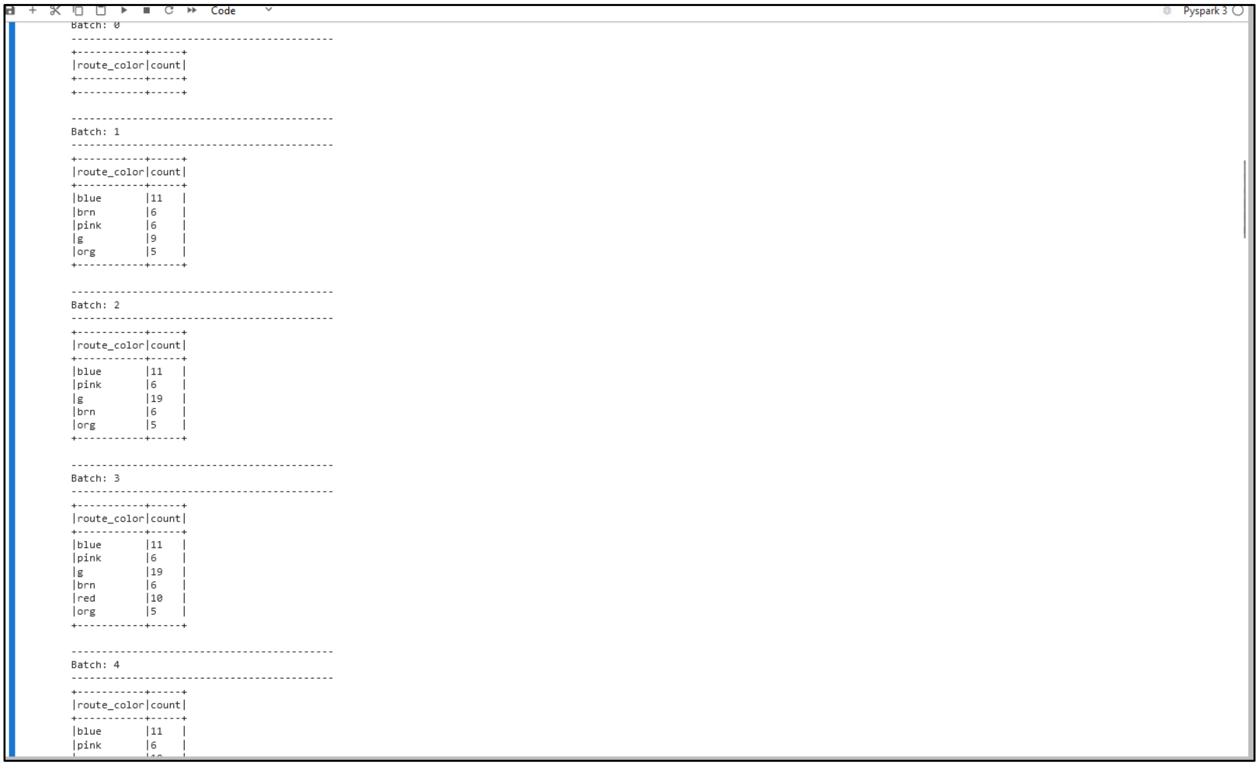
```
[8]: # Starting a write stream object by reading from api_messages every 10 seconds using append mode
api_messages. \
    writeStream. \
    outputMode("append"). \
    format("console"). \
    option('truncate', 'false'). \
    trigger(processingTime='10 seconds'). \
    start()
```

```
24/02/25 21:36:50 WARN ResolveWriteToStream: Temporary checkpoint location created which is deleted normally when the query didn't fail: /tmp/temporary-73d1fd96-93b9-4f33-9a0b-19bbd8c9906b. If it's required to delete it under any circumstances, please set spark.sql.streaming.forceDeleteTempCheckpointLocation to true. Important to know deleting temp checkpoint folder is best effort.
24/02/25 21:36:50 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Datasets and will be disabled.
[8]: <pyspark.sql.streaming.query.StreamingQuery at 0x7efff40b42b0>

-------------------------------------------
Batch: 0
-------------------------------------------
+-----+
|value|
+-----+
+-----+

-------------------------------------------
Batch: 1
-------------------------------------------
+------------------------------------------------------------------------------------------------------------+
|value                                                                                                       |
+------------------------------------------------------------------------------------------------------------+
|red,810,30089,95th/Dan Ryan,41090,Monroe,2024-02-25T15:32:51,2024-02-25T15:33:51,0,41.88481,-87.62781        |
|red,813,30089,95th/Dan Ryan,40880,Thorndale,2024-02-25T15:32:51,2024-02-25T15:34:51,0,41.99366,-87.6592      |
|red,905,30089,95th/Dan Ryan,40450,95th/Dan Ryan,2024-02-25T15:33:06,2024-02-25T15:35:06,0,41.73537,-87.62475 |
|red,907,30173,Howard,40900,Howard,2024-02-25T15:32:47,2024-02-25T15:34:47,0,42.01588,-87.66909              |
|red,909,30173,Howard,41490,Harrison,2024-02-25T15:33:17,2024-02-25T15:35:17,0,41.8674,-87.6274              |
|blue,108,30171,O'Hare,40670,Western (O'Hare Branch),2024-02-25T15:33:48,2024-02-25T15:34:48,0,41.91186,-87.68072|
|blue,112,30171,O'Hare,40010,Austin,2024-02-25T15:33:30,2024-02-25T15:35:30,0,41.87211,-87.7916              |
|blue,208,30077,Forest Park,41280,Jefferson Park,2024-02-25T15:33:50,2024-02-25T15:34:50,0,41.97063,-87.76089 |
|blue,211,30077,Forest Park,40350,UIC-Halsted,2024-02-25T15:33:45,2024-02-25T15:34:45,0,41.87552,-87.64535    |
|blue,215,30171,O'Hare,40750,Harlem (O'Hare Branch),2024-02-25T15:33:38,2024-02-25T15:35:38,0,41.978,-87.77609 |
|org,705,30182,Midway,40310,Western,2024-02-25T15:33:47,2024-02-25T15:37:47,0,41.82949,-87.6807              |
|org,708,30182,Loop,40120,35th/Archer,2024-02-25T15:33:47,2024-02-25T15:35:47,0,41.81729,-87.68086           |
|org,711,30182,Midway,41400,Roosevelt,2024-02-25T15:33:43,2024-02-25T15:35:43,0,41.87296,-87.62678           |
|red,810,30089,95th/Dan Ryan,40560,Jackson,2024-02-25T15:34:05,2024-02-25T15:35:05,0,41.88198,-87.62772       |
|red,814,30173,Howard,41420,Addison,2024-02-25T15:34:02,2024-02-25T15:35:02,0,41.94525,-87.65353             |
|red,906,30089,95th/Dan Ryan,41420,Addison,2024-02-25T15:33:22,2024-02-25T15:35:22,0,41.9534,-87.65369        |
|red,908,30173,Howard,41230,47th,2024-02-25T15:34:10,2024-02-25T15:35:10,0,41.79977,-87.63129                |
|red,910,30089,95th/Dan Ryan,40910,63rd,2024-02-25T15:33:59,2024-02-25T15:35:59,0,41.79542,-87.63117         |
|brn,407,30249,Kimball,40800,Sedgwick,2024-02-25T15:34:09,2024-02-25T15:35:09,0,41.91041,-87.6393            |
|brn,409,30249,Kimball,41210,Wellington,2024-02-25T15:34:09,2024-02-25T15:35:09,0,41.93276,-87.65313         |
+------------------------------------------------------------------------------------------------------------+
only showing top 20 rows

-------------------------------------------
```

In the Output mode, the data was grouped based on the route_color for each batch and the grouped results were displayed. The aggregate operation is not supported in the append mode.

```
[6]: # Importing required libraries
from pyspark.sql.functions import split, count, lit
# Creating route_count dataframe that groups the data based on the route color and prints the results
route_count = api_messages. \
    select(split('value', ',')[0].alias('route_color')). \
    groupBy('route_color'). \
    agg(count(lit(1)).alias('count'))
```

```
[7]: # Using "complete" mode to print the resul
route_count. \
    writeStream. \
    outputMode("complete"). \
    format("console"). \
    option('truncate', 'false'). \
    trigger(processingTime='10 seconds'). \
    start()
```



```
Batch: 0
-------------------------------------------
+-----------+-----+
|route_color|count|
+-----------+-----+
+-----------+-----+

-------------------------------------------
Batch: 1
-------------------------------------------
+-----------+-----+
|route_color|count|
+-----------+-----+
|blue       |11   |
|brn        |6    |
|pink       |6    |
|g          |9    |
|org        |5    |
+-----------+-----+

-------------------------------------------
Batch: 2
-------------------------------------------
+-----------+-----+
|route_color|count|
+-----------+-----+
|blue       |11   |
|pink       |6    |
|g          |19   |
|brn        |6    |
|org        |5    |
+-----------+-----+

-------------------------------------------
Batch: 3
-------------------------------------------
+-----------+-----+
|route_color|count|
+-----------+-----+
|blue       |11   |
|pink       |6    |
|g          |19   |
|brn        |6    |
|red        |10   |
|org        |5    |
+-----------+-----+

-------------------------------------------
Batch: 4
-------------------------------------------
+-----------+-----+
|route_color|count|
+-----------+-----+
|blue       |11   |
|pink       |6    |
```

Update Mode: In the update mode, only the total count of the colors local to a batch were displayed in the output.

```
[9]: # Using "update" mode to print the results to the console
route_count. \
    writeStream. \
    outputMode("update"). \
    format("console"). \
    option('truncate', 'false'). \
    trigger(processingTime='10 seconds'). \
    start()

24/02/25 21:52:23 WARN ResolveWriteToStream: Temporary checkpoint location created which is deleted normally when the query didn't fail: /tmp/temporary-239a90de-ad4f-4ce5-9187-00c2153ca13d. If it's required to delete it under any circumstances, please set spark.sql.streaming.forceDeleteTempCheckpointLocation to true. Important to know deleting temp checkpoint folder is best effort.
24/02/25 21:52:23 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not supported in streaming DataFrames/Datasets and will be disabled.

[9]: <pyspark.sql.streaming.query.StreamingQuery at 0x7f9c20c91b50>

-------------------------------------------
Batch: 0
-------------------------------------------
+-----------+-----+
|route_color|count|
+-----------+-----+
+-----------+-----+

-------------------------------------------
Batch: 1
-------------------------------------------
+-----------+-----+
|route_color|count|
+-----------+-----+
|blue       |10   |
|pink       |11   |
|org        |5    |
+-----------+-----+

-------------------------------------------
Batch: 2
-------------------------------------------
+-----------+-----+
|route_color|count|
+-----------+-----+
|g          |8    |
+-----------+-----+

-------------------------------------------
Batch: 3
-------------------------------------------
+-----------+-----+
|route_color|count|
+-----------+-----+
|org        |10   |
+-----------+-----+
```

Recommendations and Conclusion:

The project can be further enhanced by incorporating Apache Nifi for ingesting the data from the API, HBase to store the Model results and Apache Solr to add search function to the HDFS data. The machine learning model built in this project is overfit as the model accuracy was almost 99%. Hence the model can be rebuilt with more data samples so it can truly help to predict the delay factor in the unseen data.