به نام خدا

در این پروژه یک دیتاست که هر رکورد آن یک event است داریم. قرار است با استفاده از یک برنامه‌ی اسپارک داده‌ها را بصورت stream دریافت و پردازش کنیم. و باید توسط دو ستون date و value ناهنجاری event را تشخیص بدهیم و به کمک ستون label که نشان‌دهنده‌ی ناهنجار بودن یا نبودن است دقت پردازش خود را سنجیده و داخل ترمینال و csv بنویسیم.

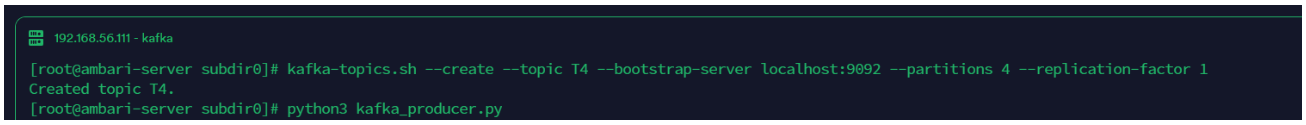
این پروژه مراحل مختلفی دارد که بصورت بخش‌های زیر پیش می‌رویم:

نوشتن داده در تاپیک کافکا برای استفاده اسپارک

- برنامه‌ای می‌نویسیم که داده‌ها را بصورت batch های ۱۰۰ تایی به تاپیک ارسال کند. کد این بخش بصورت زیر است:

```python
import pandas as pd
import time
import json
from confluent_kafka import Producer

df = pd.read_csv('timeseries.csv')
df = df.astype(str)
batch_size = 100
kafka_bootstrap_servers = 'localhost:9092'
kafka_topic = 'T4'

producer_config = {'bootstrap.servers': kafka_bootstrap_servers}
producer = Producer(producer_config)

total_rows = len(df)

# Loop through the dataset and send 100 rows at a time
for i in range(0, total_rows, batch_size):
batch = df.iloc[i:i + batch_size]
selected_columns = batch[['id', 'date', 'value', 'label']]
json_data = selected_columns.to_json(orient='records')

try:
producer.produce(kafka_topic, key=str(batch['id'].iloc[0]),
value=json_data.encode('utf-8'))
except Exception as e:
print(f"Error sending batch: {e}")

time.sleep(1)

producer.flush()
```

با دستور زیر تاپیک مورد نظر را می‌سازیم. و برنامه را اجرا می‌کنیم:



```
192.168.56.111 - kafka
[root@ambari-server subdir0]# kafka-topics.sh --create --topic T4 --bootstrap-server localhost:9092 --partitions 4 --replication-factor 1
Created topic T4.
[root@ambari-server subdir0]# python3 kafka_producer.py
```

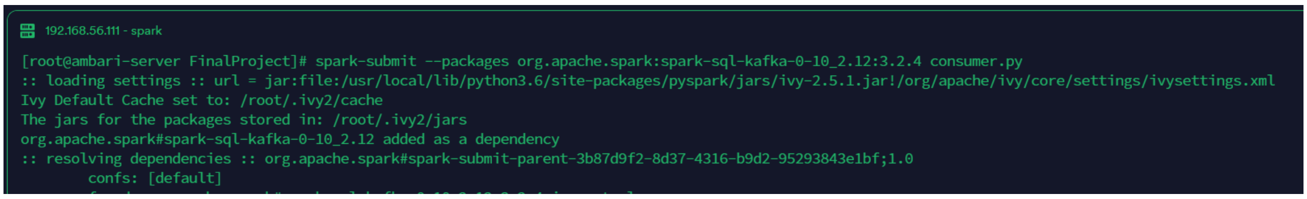
استفاده از اسپارک برای خواندن و پردازش داده‌ها و تشخیص ناهنجاری

- حالا که کدی زدیم که داده‌ها را به تاپیک کافکا می‌فرستد، کدی پیاده‌سازی می‌کنیم که با استفاده از اسپارک داده‌ها را بصورت batch بخواند و هر ۱۰۰ داده که می‌گیرد را ابتدا طبق schema مورد نظر نمایش می‌دهیم. کد این بخش بصورت زیر است:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, from_json, explode, split, expr
from pyspark.sql.types import StructType, StructField, StringType

# Define your Spark session
spark = SparkSession.builder \
    .appName("SparkConsumeKafka") \
    .config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-
10_2.12:3.2.0") \
    .getOrCreate()

df = spark \
    .readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "T4") \
    .load()


my_schema = StructType([
    StructField("id", StringType(), True),
    StructField("date", StringType(), True),
    StructField("value", StringType(), True),
    StructField("label", StringType(), True)
])

# Select the value column as a string
df = df.select(col("value").cast("string"), "timestamp")

# Remove the first and last characters of the string
df = df.withColumn("value", expr("substring(value, 2, length(value) - 2)"))

# Split the 'value' column into separate JSON objects
df = df.select(
    explode(split(col("value"), "(?<=\\}),(?=\\{)")).alias("values"),
"timestamp")
```

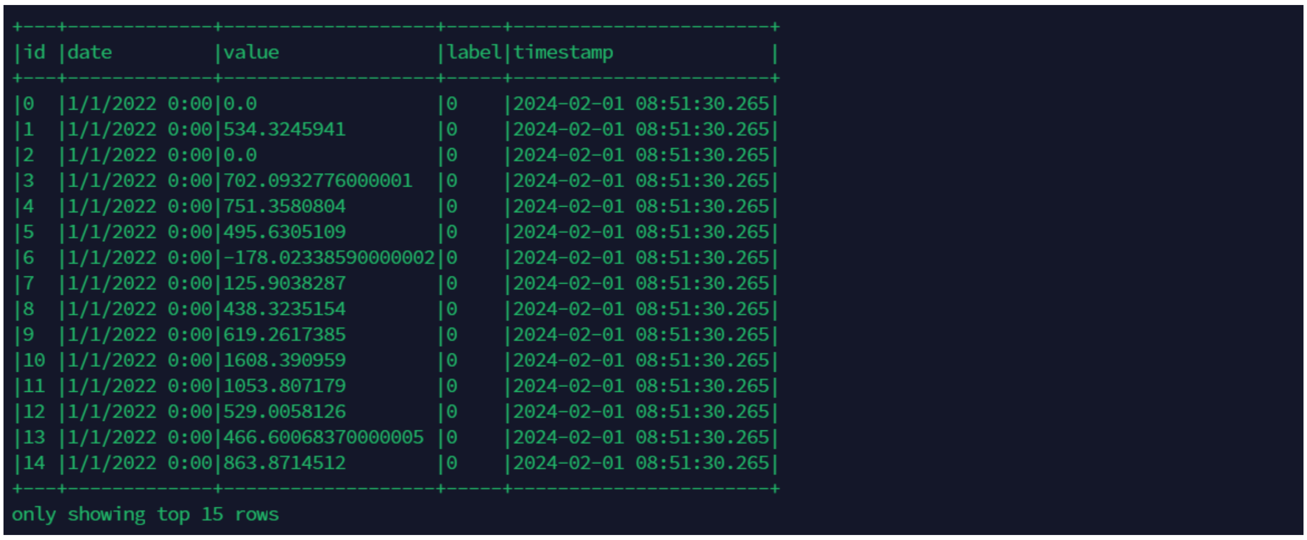
```python
# Deserialize the JSON data using the defined schema
df = df.select(
    from_json(col("values"), my_schema).alias("datas"),
    "timestamp",
)

# Extract columns from the 'datas' struct
df = df.select(
    "datas.*",
    "timestamp"
)

def show_batch(df, epoch_id):
    df.show(100, truncate=False)

# Write to the console with formatted options
query = df.writeStream.foreachBatch(show_batch).start()

# Await termination
query.awaitTermination()

# Stop the Spark session
spark.stop()
```

کدمان را با دستور زیر اجرا می‌کنیم و نتایج بدست آمده در ۱۲۰ جدول ۱۰۰ تایی (چون ۱۲۰۰۰ داده داریم) نمایش داده می‌شوند، اما برای سهولت در نمایش ۱۵ ردیف اول را نمایش می‌دهیم:

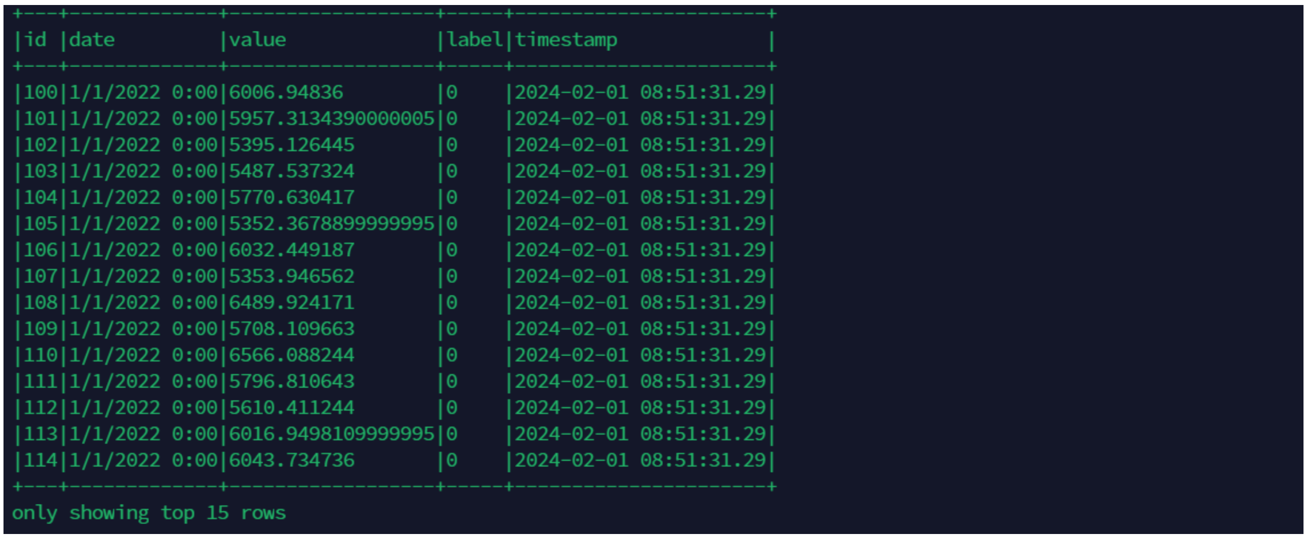


```
[root@ambari-server FinalProject]# spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.2.4 consumer.py
:: loading settings :: url = jar:file:/usr/local/lib/python3.6/site-packages/pyspark/jars/ivy-2.5.1.jar!/org/apache/ivy/core/settings/ivysettings.xml
Ivy Default Cache set to: /root/.ivy2/cache
The jars for the packages stored in: /root/.ivy2/jars
org.apache.spark#spark-sql-kafka-0-10_2.12 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-3b87d9f2-8d37-4316-b9d2-95293843e1bf;1.0
        confs: [default]
```

و نتایج بصورت نمونه:

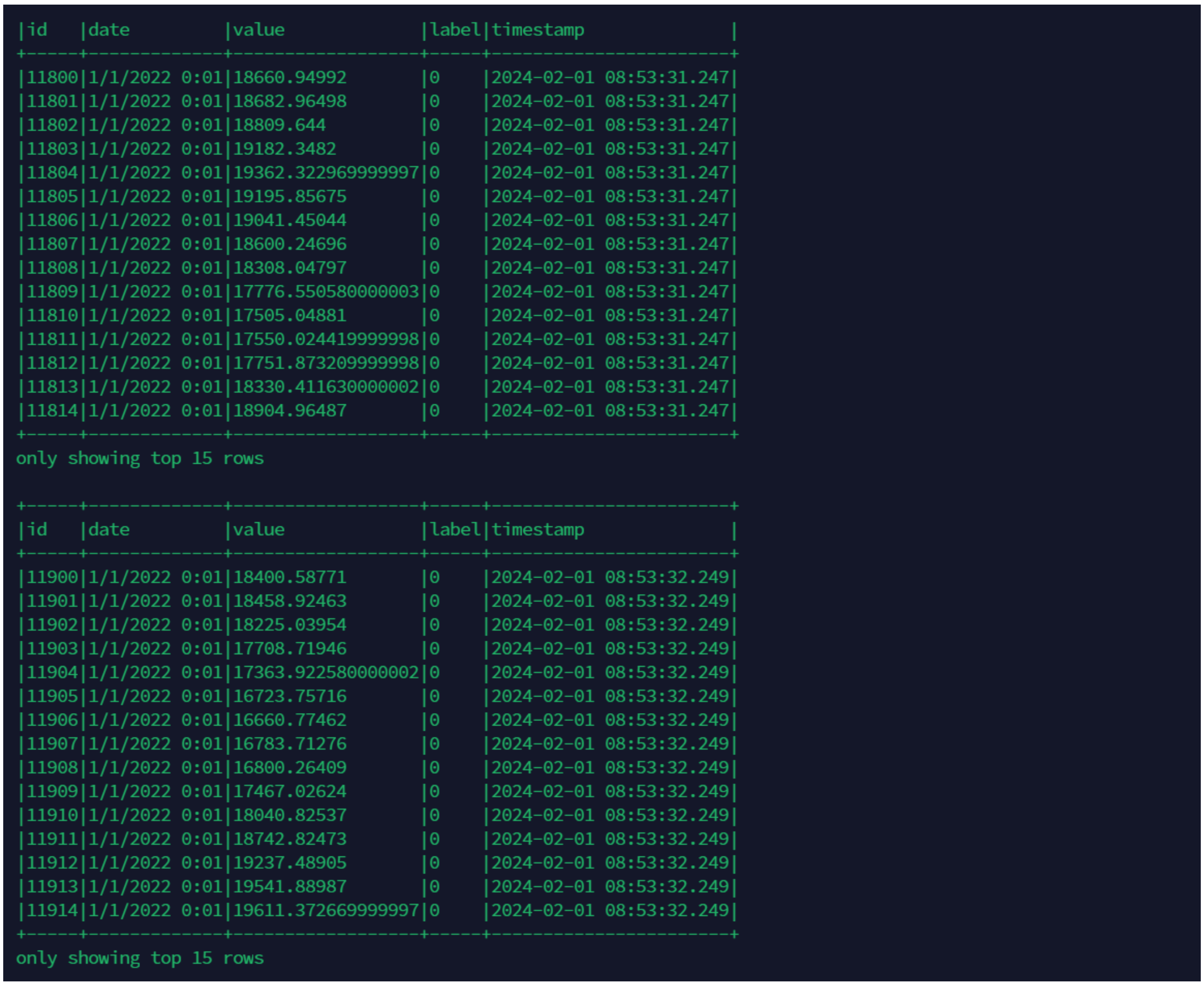


```
+---+-------------+-------------------+-----+-----------------------+
|id |date         |value              |label|timestamp              |
+---+-------------+-------------------+-----+-----------------------+
|0  |1/1/2022 0:00|0.0                |0    |2024-02-01 08:51:30.265|
|1  |1/1/2022 0:00|534.3245941        |0    |2024-02-01 08:51:30.265|
|2  |1/1/2022 0:00|0.0                |0    |2024-02-01 08:51:30.265|
|3  |1/1/2022 0:00|702.0932776000001  |0    |2024-02-01 08:51:30.265|
|4  |1/1/2022 0:00|751.3580804        |0    |2024-02-01 08:51:30.265|
|5  |1/1/2022 0:00|495.6305109        |0    |2024-02-01 08:51:30.265|
|6  |1/1/2022 0:00|-178.02338590000002|0    |2024-02-01 08:51:30.265|
|7  |1/1/2022 0:00|125.9038287        |0    |2024-02-01 08:51:30.265|
|8  |1/1/2022 0:00|438.3235154        |0    |2024-02-01 08:51:30.265|
|9  |1/1/2022 0:00|619.2617385        |0    |2024-02-01 08:51:30.265|
|10 |1/1/2022 0:00|1608.390959        |0    |2024-02-01 08:51:30.265|
|11 |1/1/2022 0:00|1053.807179        |0    |2024-02-01 08:51:30.265|
|12 |1/1/2022 0:00|529.0058126        |0    |2024-02-01 08:51:30.265|
|13 |1/1/2022 0:00|466.60068370000005 |0    |2024-02-01 08:51:30.265|
|14 |1/1/2022 0:00|863.8714512        |0    |2024-02-01 08:51:30.265|
+---+-------------+-------------------+-----+-----------------------+
only showing top 15 rows
```



```
+---+-------------+------------------+-----+----------------------+
|id |date         |value             |label|timestamp             |
+---+-------------+------------------+-----+----------------------+
|100|1/1/2022 0:00|6006.94836        |0    |2024-02-01 08:51:31.29|
|101|1/1/2022 0:00|5957.3134390000005|0    |2024-02-01 08:51:31.29|
|102|1/1/2022 0:00|5395.126445       |0    |2024-02-01 08:51:31.29|
|103|1/1/2022 0:00|5487.537324       |0    |2024-02-01 08:51:31.29|
|104|1/1/2022 0:00|5770.630417       |0    |2024-02-01 08:51:31.29|
|105|1/1/2022 0:00|5352.3678899999995|0    |2024-02-01 08:51:31.29|
|106|1/1/2022 0:00|6032.449187       |0    |2024-02-01 08:51:31.29|
|107|1/1/2022 0:00|5353.946562       |0    |2024-02-01 08:51:31.29|
|108|1/1/2022 0:00|6489.924171       |0    |2024-02-01 08:51:31.29|
|109|1/1/2022 0:00|5708.109663       |0    |2024-02-01 08:51:31.29|
|110|1/1/2022 0:00|6566.088244       |0    |2024-02-01 08:51:31.29|
|111|1/1/2022 0:00|5796.810643       |0    |2024-02-01 08:51:31.29|
|112|1/1/2022 0:00|5610.411244       |0    |2024-02-01 08:51:31.29|
|113|1/1/2022 0:00|6016.9498109999995|0    |2024-02-01 08:51:31.29|
|114|1/1/2022 0:00|6043.734736       |0    |2024-02-01 08:51:31.29|
+---+-------------+------------------+-----+----------------------+
only showing top 15 rows
```

and …



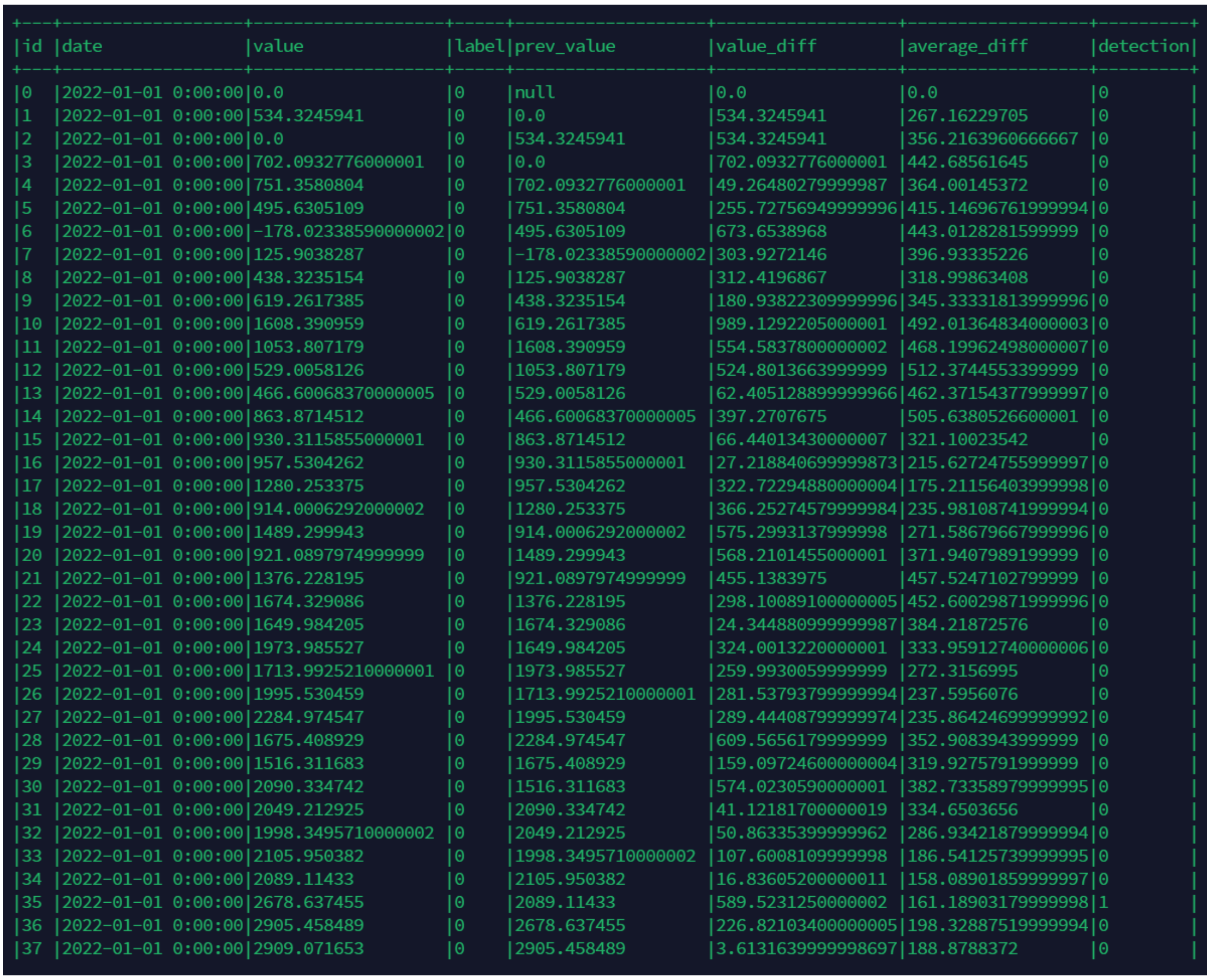
```
|id   |date         |value             |label|timestamp              |
+-----+-------------+------------------+-----+-----------------------+
|11800|1/1/2022 0:01|18660.94992       |0    |2024-02-01 08:53:31.247|
|11801|1/1/2022 0:01|18682.96498       |0    |2024-02-01 08:53:31.247|
|11802|1/1/2022 0:01|18809.644         |0    |2024-02-01 08:53:31.247|
|11803|1/1/2022 0:01|19182.3482        |0    |2024-02-01 08:53:31.247|
|11804|1/1/2022 0:01|19362.322969999997|0    |2024-02-01 08:53:31.247|
|11805|1/1/2022 0:01|19195.85675       |0    |2024-02-01 08:53:31.247|
|11806|1/1/2022 0:01|19041.45044       |0    |2024-02-01 08:53:31.247|
|11807|1/1/2022 0:01|18600.24696       |0    |2024-02-01 08:53:31.247|
|11808|1/1/2022 0:01|18308.04797       |0    |2024-02-01 08:53:31.247|
|11809|1/1/2022 0:01|17776.550580000003|0    |2024-02-01 08:53:31.247|
|11810|1/1/2022 0:01|17505.04881       |0    |2024-02-01 08:53:31.247|
|11811|1/1/2022 0:01|17550.024419999998|0    |2024-02-01 08:53:31.247|
|11812|1/1/2022 0:01|17751.873209999998|0    |2024-02-01 08:53:31.247|
|11813|1/1/2022 0:01|18330.411630000002|0    |2024-02-01 08:53:31.247|
|11814|1/1/2022 0:01|18904.96487       |0    |2024-02-01 08:53:31.247|
+-----+-------------+------------------+-----+-----------------------+
only showing top 15 rows

+-----+-------------+------------------+-----+-----------------------+
|id   |date         |value             |label|timestamp              |
+-----+-------------+------------------+-----+-----------------------+
|11900|1/1/2022 0:01|18400.58771       |0    |2024-02-01 08:53:32.249|
|11901|1/1/2022 0:01|18458.92463       |0    |2024-02-01 08:53:32.249|
|11902|1/1/2022 0:01|18225.03954       |0    |2024-02-01 08:53:32.249|
|11903|1/1/2022 0:01|17708.71946       |0    |2024-02-01 08:53:32.249|
|11904|1/1/2022 0:01|17363.922580000002|0    |2024-02-01 08:53:32.249|
|11905|1/1/2022 0:01|16723.75716       |0    |2024-02-01 08:53:32.249|
|11906|1/1/2022 0:01|16660.77462       |0    |2024-02-01 08:53:32.249|
|11907|1/1/2022 0:01|16783.71276       |0    |2024-02-01 08:53:32.249|
|11908|1/1/2022 0:01|16800.26409       |0    |2024-02-01 08:53:32.249|
|11909|1/1/2022 0:01|17467.02624       |0    |2024-02-01 08:53:32.249|
|11910|1/1/2022 0:01|18040.82537       |0    |2024-02-01 08:53:32.249|
|11911|1/1/2022 0:01|18742.82473       |0    |2024-02-01 08:53:32.249|
|11912|1/1/2022 0:01|19237.48905       |0    |2024-02-01 08:53:32.249|
|11913|1/1/2022 0:01|19541.88987       |0    |2024-02-01 08:53:32.249|
|11914|1/1/2022 0:01|19611.372669999997|0    |2024-02-01 08:53:32.249|
+-----+-------------+------------------+-----+-----------------------+
only showing top 15 rows
```

- حال می‌خواهیم به سراغ تشخیص ناهنجاری داده‌ها برویم. به event هایی ناهنجار می‌گوییم که مقدار value خیلی متفاوتی با مقادیر قبلی داشته باشد و به اصطلاح داده پرت باشد. به این داده‌ها outlier datas می‌گوییم.
  تا اینجا به عنوان تلاش اول من سعی به تشخیص ناهنجاری از روی یک داده‌ی قبل از هر رکورد داشتم.

برای تلاش دوم تشخیص را به کمک میانگین اختلاف داده‌های قبلی و درواقع شیب نمودار انجام می‌دهیم. یک پنجره می‌سازیم و روی آن حرکت می‌کنیم. یک ستون برای value قبلی، یکی برای اختلاف دو value پشت هم و یکی برای میانگین اختلاف داده‌های قبلی درنظر می‌گیریم. و در آخر یک ستون به نام detection می‌زنیم که نشان‌دهنده‌ی ناهنجار بودن یا نبودن است. و در انتها df هر batch را داخل یک تاپیک کافکا می‌ریزیم. که بعدا برای خروجی گرفتن و ساخت csv از آن استفاده کنیم.
کد این بخش بصورت زیر است:

```
def show_batch(df, epoch_id):

    df = df.select(col("id").cast("integer"),"date", col("value").cast("dou-
ble"), "label")
    window_spec = Window().orderBy("id")  # Last three records
    df = df.withColumn("prev_value", lag("value").over(window_spec))
    df = df.withColumn("value_diff", when(col("prev_value").isNotNull(),
abs(col("value") - col("prev_value"))).otherwise(lit(0)))
    df = df.withColumn("avg_3_diff", avg(col("value_diff")).over(win-
dow_spec.rowsBetween(-4, 0)))
    threshold_multiplier = 2.6
    df = df.withColumn("detection", lit(0))
    df = df.withColumn("detection", when(
        (col("value_diff") > threshold_multiplier * col("avg_3_diff")) &
        (lag("detection").over(window_spec) != 1),
        1).otherwise(0))
    df = df.select("id", "date", "value", "label", "detection")


    df.selectExpr("CAST(id AS STRING) AS key", "to_json(struct(*)) AS value")
\
        .write \
        .format("kafka") \
        .option("kafka.bootstrap.servers", "localhost:9092") \
        .option("topic", "OT") \
        .save()
```

که نتایجی که بدست می‌آید مانند زیر است:



```
+---+------------------+-------------------+-----+-------------------+------------------+------------------+---------+
|id |date              |value              |label|prev_value         |value_diff        |average_diff      |detection|
+---+------------------+-------------------+-----+-------------------+------------------+------------------+---------+
|0  |2022-01-01 0:00:00|0.0                |0    |null               |0.0               |0.0               |0        |
|1  |2022-01-01 0:00:00|534.3245941        |0    |0.0                |534.3245941       |267.16229705      |0        |
|2  |2022-01-01 0:00:00|0.0                |0    |534.3245941        |534.3245941       |356.2163960666667 |0        |
|3  |2022-01-01 0:00:00|702.0932776000001  |0    |0.0                |702.0932776000001 |442.68561645      |0        |
|4  |2022-01-01 0:00:00|751.3580804        |0    |702.0932776000001  |49.26480279999987 |364.00145372      |0        |
|5  |2022-01-01 0:00:00|495.6305109        |0    |751.3580804        |255.72756949999996|415.14696761999994|0        |
|6  |2022-01-01 0:00:00|-178.02338590000002|0    |495.6305109        |673.6538968       |443.0128281599999 |0        |
|7  |2022-01-01 0:00:00|125.9038287        |0    |-178.02338590000002|303.9272146       |396.93335226      |0        |
|8  |2022-01-01 0:00:00|438.3235154        |0    |125.9038287        |312.4196867       |318.99863408      |0        |
|9  |2022-01-01 0:00:00|619.2617385        |0    |438.3235154        |180.93822309999996|345.33331813999996|0        |
|10 |2022-01-01 0:00:00|1608.390959        |0    |619.2617385        |989.1292205000001 |492.01364834000003|0        |
|11 |2022-01-01 0:00:00|1053.807179        |0    |1608.390959        |554.5837800000002 |468.19962498000007|0        |
|12 |2022-01-01 0:00:00|529.0058126        |0    |1053.807179        |524.8013663999999 |512.3744553399999 |0        |
|13 |2022-01-01 0:00:00|466.60068370000005 |0    |529.0058126        |62.405128899999966|462.37154377999997|0        |
|14 |2022-01-01 0:00:00|863.8714512        |0    |466.60068370000005 |397.2707675       |505.6380526600001 |0        |
|15 |2022-01-01 0:00:00|930.3115855000001  |0    |863.8714512        |66.44013430000007 |321.10023542      |0        |
|16 |2022-01-01 0:00:00|957.5304262        |0    |930.3115855000001  |27.218840699999873|215.62724755999997|0        |
|17 |2022-01-01 0:00:00|1280.253375        |0    |957.5304262        |322.72294880000004|175.21156403999998|0        |
|18 |2022-01-01 0:00:00|914.0006292000002  |0    |1280.253375        |366.25274579999984|235.98108741999994|0        |
|19 |2022-01-01 0:00:00|1489.299943        |0    |914.0006292000002  |575.2993137999998 |271.58679667999996|0        |
|20 |2022-01-01 0:00:00|921.0897974999999  |0    |1489.299943        |568.2101455000001 |371.9407989199999 |0        |
|21 |2022-01-01 0:00:00|1376.228195        |0    |921.0897974999999  |455.1383975       |457.5247102799999 |0        |
|22 |2022-01-01 0:00:00|1674.329086        |0    |1376.228195        |298.10089100000005|452.60029871999996|0        |
|23 |2022-01-01 0:00:00|1649.984205        |0    |1674.329086        |24.344880999999987|384.21872576      |0        |
|24 |2022-01-01 0:00:00|1973.985527        |0    |1649.984205        |324.0013220000001 |333.95912740000006|0        |
|25 |2022-01-01 0:00:00|1713.9925210000001 |0    |1973.985527        |259.9930059999999 |272.3156995       |0        |
|26 |2022-01-01 0:00:00|1995.530459        |0    |1713.9925210000001 |281.53793799999994|237.5956076       |0        |
|27 |2022-01-01 0:00:00|2284.974547        |0    |1995.530459        |289.44408799999974|235.86424699999992|0        |
|28 |2022-01-01 0:00:00|1675.408929        |0    |2284.974547        |609.5656179999999 |352.9083943999999 |0        |
|29 |2022-01-01 0:00:00|1516.311683        |0    |1675.408929        |159.09724600000004|319.9275791999999 |0        |
|30 |2022-01-01 0:00:00|2090.334742        |0    |1516.311683        |574.0230590000001 |382.73358979999995|0        |
|31 |2022-01-01 0:00:00|2049.212925        |0    |2090.334742        |41.12181700000019 |334.6503656       |0        |
|32 |2022-01-01 0:00:00|1998.3495710000002 |0    |2049.212925        |50.86335399999962 |286.93421879999994|0        |
|33 |2022-01-01 0:00:00|2105.950382        |0    |1998.3495710000002 |107.6008109999998 |186.54125739999995|0        |
|34 |2022-01-01 0:00:00|2089.11433         |0    |2105.950382        |16.83605200000011 |158.08901859999997|0        |
|35 |2022-01-01 0:00:00|2678.637455        |0    |2089.11433         |589.5231250000002 |161.18903179999998|1        |
|36 |2022-01-01 0:00:00|2905.458489        |0    |2678.637455        |226.82103400000005|198.32887519999994|0        |
|37 |2022-01-01 0:00:00|2909.071653        |0    |2905.458489        |3.6131639999998697|188.8788372       |0        |
```

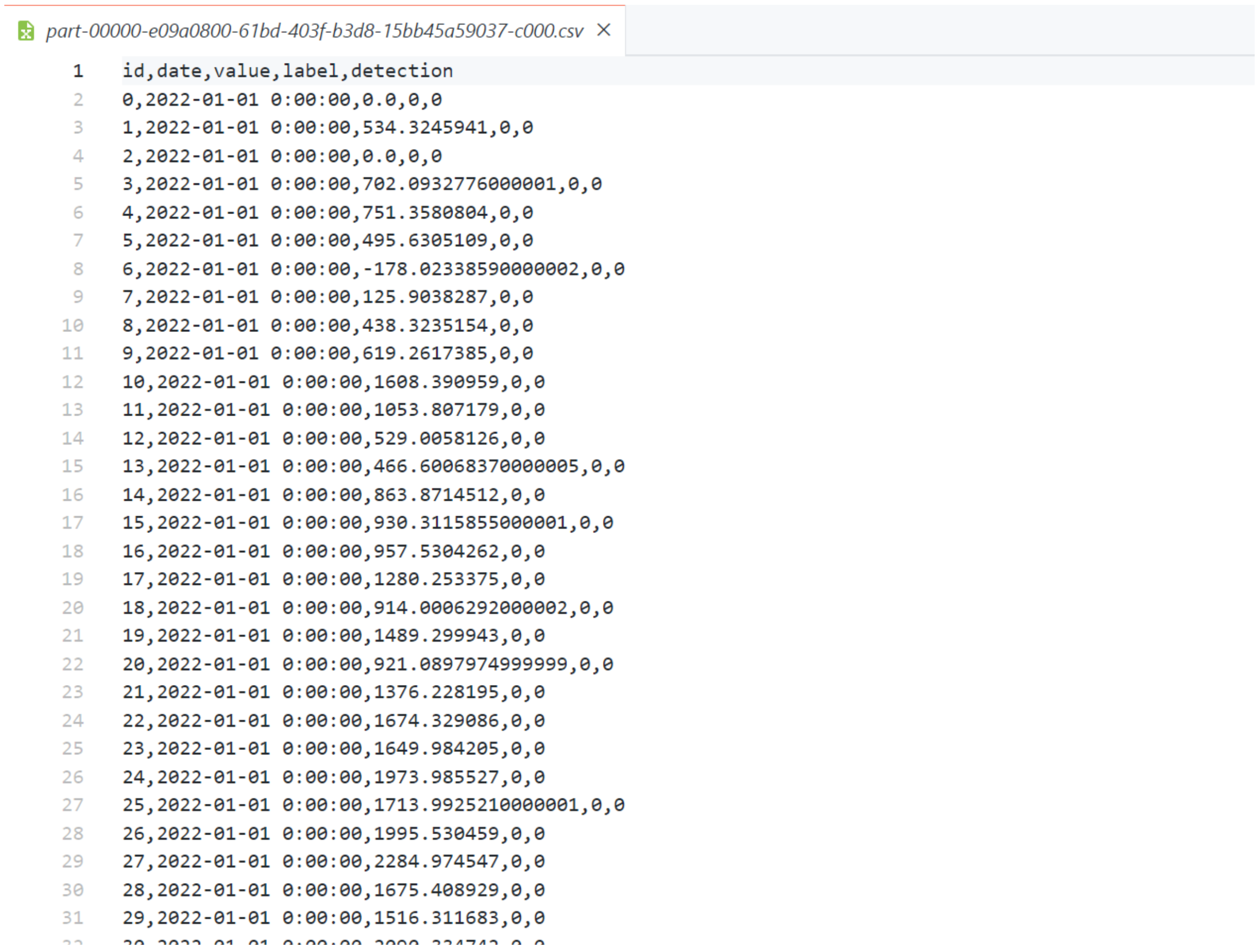
ساخت csv از تاپیک خروجی نوشته شده در کافکا

در آخر توسط فایل kafka_to_csv از تاپیک خروجی تولید شده می‌خوانیم و توسط writestream آن را در قالب csv می‌نویسیم.

کد این بخش بصورت زیر است:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, expr
from pyspark.sql.types import StructType

spark = SparkSession.builder \
    .appName("TopicToCSV") \
    .config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-
10_2.12:3.2.0") \
    .config("spark.sql.streaming.statefulOperator.checkCorrectness.enabled",
"false") \
    .getOrCreate()

spark.sparkContext.setLogLevel("ERROR")

schema = StructType().add("id", "string").add("date", "string").add("value",
"string").add("label", "string").add("detection", "string")

df = spark.readStream.format("kafka") \
    .option("kafka.bootstrap.servers", "localhost:9092") \
    .option("subscribe", "OT") \
    .option("startingOffsets", "earliest") \
    .option("failOnDataLoss", "false") \
    .load()

df = df.selectExpr("CAST(value AS STRING)")

df = df.select(
    expr("CAST(get_json_object(value, '$.id') AS STRING)").alias("id"),
    expr("CAST(get_json_object(value, '$.date') AS STRING)").alias("date"),
    expr("CAST(get_json_object(value, '$.value') AS STRING)").alias("value"),
    expr("CAST(get_json_object(value, '$.label') AS STRING)").alias("label"),
    expr("CAST(get_json_object(value, '$.detection') AS
STRING)").alias("detection")
)

query = df.writeStream \
    .format("csv") \
    .option("path", "output") \
    .option("header", "true") \
    .option("checkpointLocation", "/checkpoints/test2") \
    .start()

query.awaitTermination()
spark.stop()
```

فایل csv تولید شده بصورت زیر است:



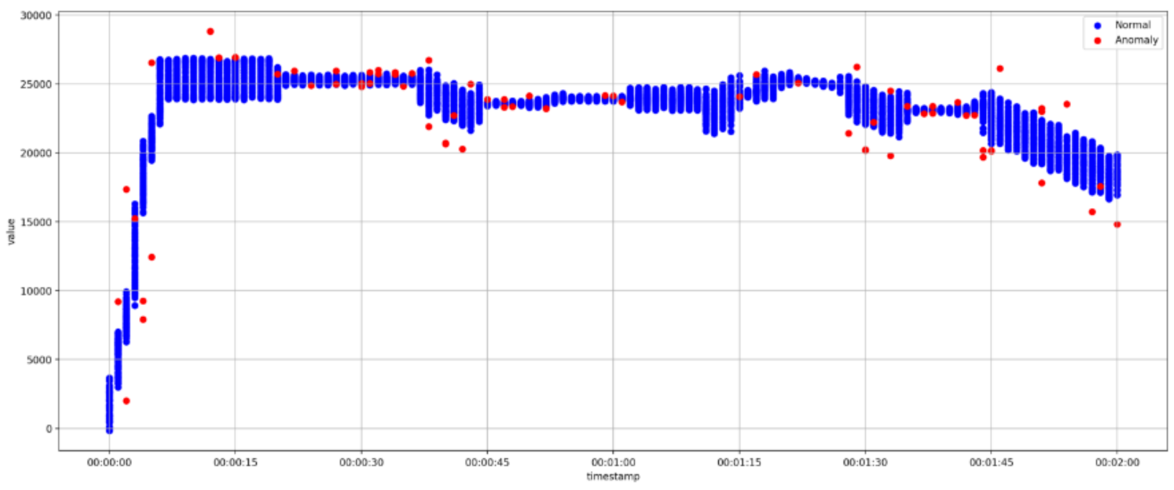
part-00000-e09a0800-61bd-403f-b3d8-15bb45a59037-c000.csv

```
id,date,value,label,detection
0,2022-01-01 0:00:00,0.0,0,0
1,2022-01-01 0:00:00,534.3245941,0,0
2,2022-01-01 0:00:00,0.0,0,0
3,2022-01-01 0:00:00,702.0932776000001,0,0
4,2022-01-01 0:00:00,751.3580804,0,0
5,2022-01-01 0:00:00,495.6305109,0,0
6,2022-01-01 0:00:00,-178.02338590000002,0,0
7,2022-01-01 0:00:00,125.9038287,0,0
8,2022-01-01 0:00:00,438.3235154,0,0
9,2022-01-01 0:00:00,619.2617385,0,0
10,2022-01-01 0:00:00,1608.390959,0,0
11,2022-01-01 0:00:00,1053.807179,0,0
12,2022-01-01 0:00:00,529.0058126,0,0
13,2022-01-01 0:00:00,466.60068370000005,0,0
14,2022-01-01 0:00:00,863.8714512,0,0
15,2022-01-01 0:00:00,930.3115855000001,0,0
16,2022-01-01 0:00:00,957.5304262,0,0
17,2022-01-01 0:00:00,1280.253375,0,0
18,2022-01-01 0:00:00,914.0006292000002,0,0
19,2022-01-01 0:00:00,1489.299943,0,0
20,2022-01-01 0:00:00,921.0897974999999,0,0
21,2022-01-01 0:00:00,1376.228195,0,0
22,2022-01-01 0:00:00,1674.329086,0,0
23,2022-01-01 0:00:00,1649.984205,0,0
24,2022-01-01 0:00:00,1973.985527,0,0
25,2022-01-01 0:00:00,1713.9925210000001,0,0
26,2022-01-01 0:00:00,1995.530459,0,0
27,2022-01-01 0:00:00,2284.974547,0,0
28,2022-01-01 0:00:00,1675.408929,0,0
29,2022-01-01 0:00:00,1516.311683,0,0
```

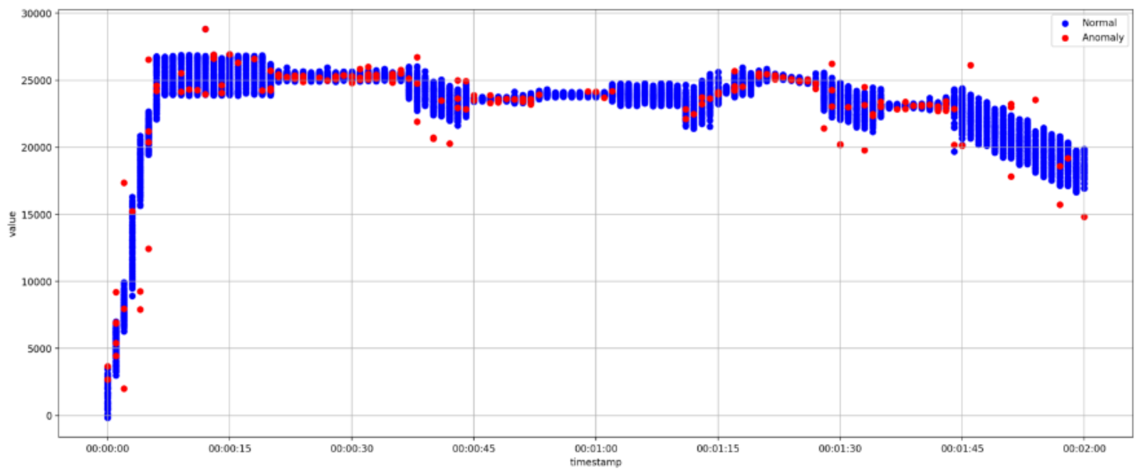
نمایش نمودارهای ورودی و خروجی تولید شده به همراه چاپ metric ها

بعد از ساخت csv توسط streamlit و کتابخانه‌ی pandas و matplotlib دیتاست‌ها را می خوانیم و نمودارها را رسم می‌کنیم و accuracy, recall, precision را چاپ می‌کنیم.

PLOT of INPUT dataset



PLOT of OUTPUT dataset



accuracy : `98.80823401950163`

recall : `75.64102564102564`

precision : `32.240437158469945`