Assignment 9.3

May 13, 2023

0.1 Assignment 9.3

[1]:
```
import os
import shutil
import json
from pathlib import Path

import pandas as pd
import warnings
warnings.filterwarnings('ignore')

from kafka import KafkaProducer, KafkaAdminClient
from kafka.admin.new_topic import NewTopic
from kafka.errors import TopicAlreadyExistsError

from pyspark.sql import SparkSession
from pyspark.streaming import StreamingContext
from pyspark import SparkConf
from pyspark.sql.functions import window, from_json, col, expr, to_json,␣
 ↪struct, when
from pyspark.sql.types import StringType, TimestampType, DoubleType,␣
 ↪StructField, StructType
from pyspark.sql.functions import udf

current_dir = Path(os.getcwd()).absolute()
checkpoint_dir = current_dir.joinpath('checkpoints')
joined_checkpoint_dir = checkpoint_dir.joinpath('joined')

if joined_checkpoint_dir.exists():
    shutil.rmtree(joined_checkpoint_dir)

joined_checkpoint_dir.mkdir(parents=True, exist_ok=True)
```

0.1.1 Configuration Parameters

TODO: Change the configuration prameters to the appropriate values for your setup.

[2]:
```
config = dict(
    bootstrap_servers=['kafka.kafka.svc.cluster.local:9092'],
    first_name='Jake',
    last_name='Meyer'
)

config['client_id'] = '{}{}'.format(
    config['last_name'],
    config['first_name']
)
config['topic_prefix'] = '{}{}'.format(
    config['last_name'],
    config['first_name']
)

config['locations_topic'] = '{}-locations'.format(config['topic_prefix'])
config['accelerations_topic'] = '{}-accelerations'.
 ↪format(config['topic_prefix'])
config['joined_topic'] = '{}-joined'.format(config['topic_prefix'])

config
```

[2]:
```
{'bootstrap_servers': ['kafka.kafka.svc.cluster.local:9092'],
 'first_name': 'Jake',
 'last_name': 'Meyer',
 'client_id': 'MeyerJake',
 'topic_prefix': 'MeyerJake',
 'locations_topic': 'MeyerJake-locations',
 'accelerations_topic': 'MeyerJake-accelerations',
 'joined_topic': 'MeyerJake-joined'}
```

0.1.2 Create Topic Utility Function

The `create_kafka_topic` helps create a Kafka topic based on your configuration settings. For instance, if your first name is *John* and your last name is *Doe*, `create_kafka_topic('locations')` will create a topic with the name `DoeJohn-locations`. The function will not create the topic if it already exists.

[3]:
```
def create_kafka_topic(topic_name, config=config, num_partitions=1,␣
 ↪replication_factor=1):
    bootstrap_servers = config['bootstrap_servers']
    client_id = config['client_id']
    topic_prefix = config['topic_prefix']
    name = '{}-{}'.format(topic_prefix, topic_name)

    admin_client = KafkaAdminClient(
        bootstrap_servers=bootstrap_servers,
```

```
        client_id=client_id
    )

    topic = NewTopic(
        name=name,
        num_partitions=num_partitions,
        replication_factor=replication_factor
    )

    topic_list = [topic]
    try:
        admin_client.create_topics(new_topics=topic_list)
        print('Created topic "{}"'.format(name))
    except TopicAlreadyExistsError as e:
        print('Topic "{}" already exists'.format(name))

create_kafka_topic('joined')
```

```
Topic "MeyerJake-joined" already exists
```

TODO: This code is identical to the code used in 9.1 to publish acceleration and location data to the `LastnameFirstname-simple` topic. You will need to add in the code you used to create the `df_accelerations` dataframe. In order to read data from this topic, make sure that you are running the notebook you created in assignment 8 that publishes acceleration and location data to the LastnameFirstname-simple topic.

[21]:
```
spark = SparkSession\
    .builder\
    .appName("Assignment09")\
    .getOrCreate()

df_locations = spark \
  .readStream \
  .format("kafka") \
  .option("kafka.bootstrap.servers", "kafka.kafka.svc.cluster.local:9092") \
  .option("subscribe", config['locations_topic']) \
  .option("startingOffsets", "earliest") \
  .load()
## .option("failOnDataLoss", "false") \

## TODO: Add code to create the df_accelerations dataframe
df_accelerations = spark \
  .readStream \
  .format("kafka") \
  .option("kafka.bootstrap.servers", "kafka.kafka.svc.cluster.local:9092") \
  .option("subscribe", config['accelerations_topic']) \
  .option("startingOffsets", "earliest") \
  .load()
```

```
## .option("failOnDataLoss", "false") \
```

The following code defines a Spark schema for location and acceleration data as well as a user-defined function (UDF) for parsing the location and acceleration JSON data.

[22]:
```
location_schema = StructType([
    StructField('offset', DoubleType(), nullable=True),
    StructField('id', StringType(), nullable=True),
    StructField('ride_id', StringType(), nullable=True),
    StructField('uuid', StringType(), nullable=True),
    StructField('course', DoubleType(), nullable=True),
    StructField('latitude', DoubleType(), nullable=True),
    StructField('longitude', DoubleType(), nullable=True),
    StructField('geohash', StringType(), nullable=True),
    StructField('speed', DoubleType(), nullable=True),
    StructField('accuracy', DoubleType(), nullable=True),
])

acceleration_schema = StructType([
    StructField('offset', DoubleType(), nullable=True),
    StructField('id', StringType(), nullable=True),
    StructField('ride_id', StringType(), nullable=True),
    StructField('uuid', StringType(), nullable=True),
    StructField('x', DoubleType(), nullable=True),
    StructField('y', DoubleType(), nullable=True),
    StructField('z', DoubleType(), nullable=True),
])

udf_parse_acceleration = udf(lambda x: json.loads(x.decode('utf-8')),␣
 ↪acceleration_schema)
udf_parse_location = udf(lambda x: json.loads(x.decode('utf-8')),␣
 ↪location_schema)
```

TODO:

- Complete the code to create the `accelerationsWithWatermark` dataframe.
  - Select the `timestamp` field with the alias `acceleration_timestamp`
  - Use the `udf_parse_acceleration` UDF to parse the JSON values
  - Select the `ride_id` as `acceleration_ride_id`
  - Select the `x`, `y`, and `z` columns
  - Use the same watermark timespan used in the `locationsWithWatermark` dataframe

[23]:
```
locationsWithWatermark = df_locations \
  .select(
    col('timestamp').alias('location_timestamp'),
    udf_parse_location(df_locations['value']).alias('json_value')
   ) \
  .select(
```

```
    col('location_timestamp'),
    col('json_value.ride_id').alias('location_ride_id'),
    col('json_value.speed').alias('speed'),
    col('json_value.latitude').alias('latitude'),
    col('json_value.longitude').alias('longitude'),
    col('json_value.geohash').alias('geohash'),
    col('json_value.accuracy').alias('accuracy')
  ) \
 .withWatermark('location_timestamp', "2 seconds")

## Follow similar suite as locationsWithWatermark as above, but for specified␣
 ↪directions in previous cell.
accelerationsWithWatermark = df_accelerations \
  .select(
    col('timestamp').alias('acceleration_timestamp'),
    udf_parse_acceleration(df_accelerations['value']).alias('json_value')
   ) \
  .select(
    col('acceleration_timestamp'),
    col('json_value.ride_id').alias('acceleration_ride_id'),
    col('json_value.x').alias('x'),
    col('json_value.y').alias('y'),
    col('json_value.z').alias('z'),
  ) \
 .withWatermark('acceleration_timestamp', "2 seconds")
```

TODO:

- Complete the code to create the `df_joined` dataframe. See http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#stream-stream-joins for additional information.

[24]:
```
## Follow the examples in the resource for df_joined.
df_joined = locationsWithWatermark.join(
    accelerationsWithWatermark, expr("""location_ride_id =␣
 ↪acceleration_ride_id""")
)
df_joined
```

[24]:
```
DataFrame[location_timestamp: timestamp, location_ride_id: string, speed:
double, latitude: double, longitude: double, geohash: string, accuracy: double,
acceleration_timestamp: timestamp, acceleration_ride_id: string, x: double, y:
double, z: double]
```

If you correctly created the `df_joined` dataframe, you should be able to use the following code to create a streaming query that outputs results to the `LastnameFirstname-joined` topic.

[25]:
```
## Updated ride_id to acceleration_ride_id due to errors surfacing.
ds_joined = df_joined \
  .withColumn(
    'value',
    to_json(
        struct(
            'acceleration_ride_id', 'location_timestamp', 'speed',
            'latitude', 'longitude', 'geohash', 'accuracy',
            'acceleration_timestamp', 'x', 'y', 'z'
        )
    )
    ).withColumn(
     'key', col('acceleration_ride_id')
    ) \
  .selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)") \
  .writeStream \
  .format("kafka") \
  .option("kafka.bootstrap.servers", "kafka.kafka.svc.cluster.local:9092") \
  .option("topic", config['joined_topic']) \
  .option("checkpointLocation", str(joined_checkpoint_dir)) \
  .start()

try:
    ds_joined.awaitTermination()
except KeyboardInterrupt:
    print("STOPPING STREAMING DATA")
```

```
23/05/13 14:00:51 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not
supported in streaming DataFrames/Datasets and will be disabled.
23/05/13 14:00:51 WARN AdminClientConfig: The configuration 'key.deserializer'
was supplied but isn't a known config.
23/05/13 14:00:51 WARN AdminClientConfig: The configuration 'value.deserializer'
was supplied but isn't a known config.
23/05/13 14:00:51 WARN AdminClientConfig: The configuration 'enable.auto.commit'
was supplied but isn't a known config.
23/05/13 14:00:51 WARN AdminClientConfig: The configuration 'max.poll.records'
was supplied but isn't a known config.
23/05/13 14:00:51 WARN AdminClientConfig: The configuration 'auto.offset.reset'
was supplied but isn't a known config.
23/05/13 14:00:51 ERROR MicroBatchExecution: Query [id =
a258885d-cde2-40d4-9afb-9a6554ccef27, runId =
0c7fa857-60bf-455c-b9c8-cedebea52a49] terminated with error
java.lang.NoClassDefFoundError: org/apache/kafka/clients/admin/OffsetSpec
        at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.$anonfun$fetchEa
rliestOffsets$2(KafkaOffsetReaderAdmin.scala:289)
        at
scala.collection.TraversableLike.$anonfun$map$1(TraversableLike.scala:286)
```

```
        at scala.collection.Iterator.foreach(Iterator.scala:943)
        at scala.collection.Iterator.foreach$(Iterator.scala:943)
        at scala.collection.AbstractIterator.foreach(Iterator.scala:1431)
        at scala.collection.IterableLike.foreach(IterableLike.scala:74)
        at scala.collection.IterableLike.foreach$(IterableLike.scala:73)
        at scala.collection.AbstractIterable.foreach(Iterable.scala:56)
        at scala.collection.TraversableLike.map(TraversableLike.scala:286)
        at scala.collection.TraversableLike.map$(TraversableLike.scala:279)
        at scala.collection.mutable.AbstractSet.scala$collection$SetLike$$super$
map(Set.scala:50)
        at scala.collection.SetLike.map(SetLike.scala:105)
        at scala.collection.SetLike.map$(SetLike.scala:105)
        at scala.collection.mutable.AbstractSet.map(Set.scala:50)
        at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.$anonfun$fetchEa
rliestOffsets$1(KafkaOffsetReaderAdmin.scala:289)
        at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.$anonfun$partiti
onsAssignedToAdmin$1(KafkaOffsetReaderAdmin.scala:501)
        at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.withRetries(Kafk
aOffsetReaderAdmin.scala:518)
        at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.partitionsAssign
edToAdmin(KafkaOffsetReaderAdmin.scala:498)
        at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.fetchEarliestOff
sets(KafkaOffsetReaderAdmin.scala:288)
        at org.apache.spark.sql.kafka010.KafkaMicroBatchStream.$anonfun$getOrCre
ateInitialPartitionOffsets$1(KafkaMicroBatchStream.scala:249)
        at scala.Option.getOrElse(Option.scala:189)
        at org.apache.spark.sql.kafka010.KafkaMicroBatchStream.getOrCreateInitia
lPartitionOffsets(KafkaMicroBatchStream.scala:246)
        at org.apache.spark.sql.kafka010.KafkaMicroBatchStream.initialOffset(Kaf
kaMicroBatchStream.scala:98)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.$anonfun
$getStartOffset$2(MicroBatchExecution.scala:455)
        at scala.Option.getOrElse(Option.scala:189)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.getStart
Offset(MicroBatchExecution.scala:455)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.$anonfun
$constructNextBatch$4(MicroBatchExecution.scala:489)
        at org.apache.spark.sql.execution.streaming.ProgressReporter.reportTimeT
aken(ProgressReporter.scala:411)
        at org.apache.spark.sql.execution.streaming.ProgressReporter.reportTimeT
aken$(ProgressReporter.scala:409)
        at org.apache.spark.sql.execution.streaming.StreamExecution.reportTimeTa
ken(StreamExecution.scala:67)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.$anonfun
$constructNextBatch$2(MicroBatchExecution.scala:488)
        at
scala.collection.TraversableLike.$anonfun$map$1(TraversableLike.scala:286)
        at scala.collection.Iterator.foreach(Iterator.scala:943)
```

```
        at scala.collection.Iterator.foreach$(Iterator.scala:943)
        at scala.collection.AbstractIterator.foreach(Iterator.scala:1431)
        at scala.collection.IterableLike.foreach(IterableLike.scala:74)
        at scala.collection.IterableLike.foreach$(IterableLike.scala:73)
        at scala.collection.AbstractIterable.foreach(Iterable.scala:56)
        at scala.collection.TraversableLike.map(TraversableLike.scala:286)
        at scala.collection.TraversableLike.map$(TraversableLike.scala:279)
        at scala.collection.AbstractTraversable.map(Traversable.scala:108)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.$anonfun
$constructNextBatch$1(MicroBatchExecution.scala:477)
        at
scala.runtime.java8.JFunction0$mcZ$sp.apply(JFunction0$mcZ$sp.java:23)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.withProg
ressLocked(MicroBatchExecution.scala:802)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.construc
tNextBatch(MicroBatchExecution.scala:473)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.$anonfun
$runActivatedStream$2(MicroBatchExecution.scala:266)
        at
scala.runtime.java8.JFunction0$mcV$sp.apply(JFunction0$mcV$sp.java:23)
        at org.apache.spark.sql.execution.streaming.ProgressReporter.reportTimeT
aken(ProgressReporter.scala:411)
        at org.apache.spark.sql.execution.streaming.ProgressReporter.reportTimeT
aken$(ProgressReporter.scala:409)
        at org.apache.spark.sql.execution.streaming.StreamExecution.reportTimeTa
ken(StreamExecution.scala:67)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.$anonfun
$runActivatedStream$1(MicroBatchExecution.scala:247)
        at org.apache.spark.sql.execution.streaming.ProcessingTimeExecutor.execu
te(TriggerExecutor.scala:67)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.runActiv
atedStream(MicroBatchExecution.scala:237)
        at org.apache.spark.sql.execution.streaming.StreamExecution.$anonfun$run
Stream$1(StreamExecution.scala:306)
        at
scala.runtime.java8.JFunction0$mcV$sp.apply(JFunction0$mcV$sp.java:23)
        at org.apache.spark.sql.SparkSession.withActive(SparkSession.scala:827)
        at org.apache.spark.sql.execution.streaming.StreamExecution.org$apache$s
park$sql$execution$streaming$StreamExecution$$runStream(StreamExecution.scala:28
4)
        at org.apache.spark.sql.execution.streaming.StreamExecution$$anon$1.run(
StreamExecution.scala:207)
Caused by: java.lang.ClassNotFoundException:
org.apache.kafka.clients.admin.OffsetSpec
        … 58 more
Exception in thread "stream execution thread for [id =
a258885d-cde2-40d4-9afb-9a6554ccef27, runId =
0c7fa857-60bf-455c-b9c8-cedebea52a49]" java.lang.NoClassDefFoundError:
```

```
org/apache/kafka/clients/admin/OffsetSpec
        at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.$anonfun$fetchEa
rliestOffsets$2(KafkaOffsetReaderAdmin.scala:289)
        at
scala.collection.TraversableLike.$anonfun$map$1(TraversableLike.scala:286)
        at scala.collection.Iterator.foreach(Iterator.scala:943)
        at scala.collection.Iterator.foreach$(Iterator.scala:943)
        at scala.collection.AbstractIterator.foreach(Iterator.scala:1431)
        at scala.collection.IterableLike.foreach(IterableLike.scala:74)
        at scala.collection.IterableLike.foreach$(IterableLike.scala:73)
        at scala.collection.AbstractIterable.foreach(Iterable.scala:56)
        at scala.collection.TraversableLike.map(TraversableLike.scala:286)
        at scala.collection.TraversableLike.map$(TraversableLike.scala:279)
        at scala.collection.mutable.AbstractSet.scala$collection$SetLike$$super$
map(Set.scala:50)
        at scala.collection.SetLike.map(SetLike.scala:105)
        at scala.collection.SetLike.map$(SetLike.scala:105)
        at scala.collection.mutable.AbstractSet.map(Set.scala:50)
        at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.$anonfun$fetchEa
rliestOffsets$1(KafkaOffsetReaderAdmin.scala:289)
        at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.$anonfun$partiti
onsAssignedToAdmin$1(KafkaOffsetReaderAdmin.scala:501)
        at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.withRetries(Kafk
aOffsetReaderAdmin.scala:518)
        at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.partitionsAssign
edToAdmin(KafkaOffsetReaderAdmin.scala:498)
        at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.fetchEarliestOff
sets(KafkaOffsetReaderAdmin.scala:288)
        at org.apache.spark.sql.kafka010.KafkaMicroBatchStream.$anonfun$getOrCre
ateInitialPartitionOffsets$1(KafkaMicroBatchStream.scala:249)
        at scala.Option.getOrElse(Option.scala:189)
        at org.apache.spark.sql.kafka010.KafkaMicroBatchStream.getOrCreateInitia
lPartitionOffsets(KafkaMicroBatchStream.scala:246)
        at org.apache.spark.sql.kafka010.KafkaMicroBatchStream.initialOffset(Kaf
kaMicroBatchStream.scala:98)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.$anonfun
$getStartOffset$2(MicroBatchExecution.scala:455)
        at scala.Option.getOrElse(Option.scala:189)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.getStart
Offset(MicroBatchExecution.scala:455)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.$anonfun
$constructNextBatch$4(MicroBatchExecution.scala:489)
        at org.apache.spark.sql.execution.streaming.ProgressReporter.reportTimeT
aken(ProgressReporter.scala:411)
        at org.apache.spark.sql.execution.streaming.ProgressReporter.reportTimeT
aken$(ProgressReporter.scala:409)
        at org.apache.spark.sql.execution.streaming.StreamExecution.reportTimeTa
ken(StreamExecution.scala:67)
```

```
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.$anonfun
$constructNextBatch$2(MicroBatchExecution.scala:488)
        at
scala.collection.TraversableLike.$anonfun$map$1(TraversableLike.scala:286)
        at scala.collection.Iterator.foreach(Iterator.scala:943)
        at scala.collection.Iterator.foreach$(Iterator.scala:943)
        at scala.collection.AbstractIterator.foreach(Iterator.scala:1431)
        at scala.collection.IterableLike.foreach(IterableLike.scala:74)
        at scala.collection.IterableLike.foreach$(IterableLike.scala:73)
        at scala.collection.AbstractIterable.foreach(Iterable.scala:56)
        at scala.collection.TraversableLike.map(TraversableLike.scala:286)
        at scala.collection.TraversableLike.map$(TraversableLike.scala:279)
        at scala.collection.AbstractTraversable.map(Traversable.scala:108)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.$anonfun
$constructNextBatch$1(MicroBatchExecution.scala:477)
        at
scala.runtime.java8.JFunction0$mcZ$sp.apply(JFunction0$mcZ$sp.java:23)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.withProg
ressLocked(MicroBatchExecution.scala:802)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.construc
tNextBatch(MicroBatchExecution.scala:473)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.$anonfun
$runActivatedStream$2(MicroBatchExecution.scala:266)
        at
scala.runtime.java8.JFunction0$mcV$sp.apply(JFunction0$mcV$sp.java:23)
        at org.apache.spark.sql.execution.streaming.ProgressReporter.reportTimeT
aken(ProgressReporter.scala:411)
        at org.apache.spark.sql.execution.streaming.ProgressReporter.reportTimeT
aken$(ProgressReporter.scala:409)
        at org.apache.spark.sql.execution.streaming.StreamExecution.reportTimeTa
ken(StreamExecution.scala:67)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.$anonfun
$runActivatedStream$1(MicroBatchExecution.scala:247)
        at org.apache.spark.sql.execution.streaming.ProcessingTimeExecutor.execu
te(TriggerExecutor.scala:67)
        at org.apache.spark.sql.execution.streaming.MicroBatchExecution.runActiv
atedStream(MicroBatchExecution.scala:237)
        at org.apache.spark.sql.execution.streaming.StreamExecution.$anonfun$run
Stream$1(StreamExecution.scala:306)
        at
scala.runtime.java8.JFunction0$mcV$sp.apply(JFunction0$mcV$sp.java:23)
        at org.apache.spark.sql.SparkSession.withActive(SparkSession.scala:827)
        at org.apache.spark.sql.execution.streaming.StreamExecution.org$apache$s
park$sql$execution$streaming$StreamExecution$$runStream(StreamExecution.scala:28
4)
        at org.apache.spark.sql.execution.streaming.StreamExecution$$anon$1.run(
StreamExecution.scala:207)
Caused by: java.lang.ClassNotFoundException:
```

```
org.apache.kafka.clients.admin.OffsetSpec
        … 58 more
```

```
---------------------------------------------------------------------------
StreamingQueryException                   Traceback (most recent call last)
Cell In[25], line 24
      2 ds_joined = df_joined \
      3   .withColumn(
      4     'value',
   (…)
     20   .option("checkpointLocation", str(joined_checkpoint_dir)) \
     21   .start()
     23 try:
---> 24     ds_joined.awaitTermination()
     25 except KeyboardInterrupt:
     26     print("STOPPING STREAMING DATA")

File /opt/conda/lib/python3.10/site-packages/pyspark/sql/streaming/query.py:201
 ↪in StreamingQuery.awaitTermination(self, timeout)
    199     return self._jsq.awaitTermination(int(timeout * 1000))
    200 else:
--> 201     return self._jsq.awaitTermination()

File /opt/conda/lib/python3.10/site-packages/py4j/java_gateway.py:1322, in
 ↪JavaMember.__call__(self, *args)
   1316 command = proto.CALL_COMMAND_NAME +\
   1317     self.command_header +\
   1318     args_command +\
   1319     proto.END_COMMAND_PART
   1321 answer = self.gateway_client.send_command(command)
-> 1322 return_value = get_return_value(
   1323     answer, self.gateway_client, self.target_id, self.name)
   1325 for temp_arg in temp_args:
   1326     if hasattr(temp_arg, "_detach"):

File /opt/conda/lib/python3.10/site-packages/pyspark/errors/exceptions/captured
 ↪py:175, in capture_sql_exception.<locals>.deco(*a, **kw)
    171 converted = convert_exception(e.java_exception)
    172 if not isinstance(converted, UnknownException):
    173     # Hide where the exception came from that shows a non-Pythonic
    174     # JVM exception message.
--> 175     raise converted from None
    176 else:
    177     raise
```

```
StreamingQueryException: [STREAM_FAILED] Query [id =␣
↪a258885d-cde2-40d4-9afb-9a6554ccef27, runId =␣
↪0c7fa857-60bf-455c-b9c8-cedebea52a49] terminated with exception: org/apache/
↪kafka/clients/admin/OffsetSpec
```

[26]:
```
print(ds_joined)
```

```
<pyspark.sql.streaming.query.StreamingQuery object at 0x7fca24e86650>
```

[27]:
```
print(type(ds_joined))
```

```
<class 'pyspark.sql.streaming.query.StreamingQuery'>
```