

# Assignment 9.2

May 13, 2023

## 0.1 Assignment 9.2

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

import pandas as pd
import time
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
from pyspark.sql.types import StringType, TimestampType, DoubleType, \
    StructField, StructType
from pyspark.sql.functions import udf, window, mean, from_json, col

current_dir = Path(os.getcwd()).absolute()
checkpoint_dir = current_dir.joinpath('checkpoints')
locations_windowed_checkpoint_dir = checkpoint_dir.
    joinpath('locations-windowed')

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

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

### 0.1.1 Configuration Parameters

**TODO:** Change the configuration parameters 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['windowed_topic'] = '{}-windowed'.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',
      'windowed_topic': 'MeyerJake-windowed'}
```

### 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('windowed')

```

Topic "MeyerJake-windowed" 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.

```

[15]: 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()

## 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()

```

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.

```
[16]: 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)
```

See <http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#window-operations-on-event-time> for details on how to implement windowed operations.

The following code selects the `timestamp` column from the `df_locations` dataframe that reads from the `LastnameFirstname-locations` topic and parses the binary value using the `udf_parse_location` UDF and defines the result to the `json_value` column.

```
df_locations \
    .select(
        col('timestamp'),
        udf_parse_location(df_locations['value']).alias('json_value')
    )
```

From here, you can select data from the `json_value` column using the `select` method. For instance, if you saved the results of the previous code snippet to `df_locations_parsed` you could select columns from the `json_value` field and assign them aliases using the following code.

```
df_locations_parsed.select(
    col('timestamp'),
```

```

    col('json_value.ride_id').alias('ride_id'),
    col('json_value.uuid').alias('uuid'),
    col('json_value.speed').alias('speed')
)

```

Next, you will want to add a watermark and group by `ride_id` and `speed` using a window duration of *30 seconds* and a slide duration of *15 seconds*. Use the `withWatermark` method in conjunction with the `groupBy` method. The [Spark streaming documentation](#) should provide examples of how to do this.

Next use the `mean` aggregation method to compute the average values and rename the column `avg(speed)` to `value` and the column `ride_id` to `key`. The reason you are renaming these values is that the PySpark Kafka API expects `key` and `value` as inputs. In a production example, you would setup serialization that would handle these details for you.

When you are finished, you should have a streaming query with `key` and `value` as columns.

```

[17]: ## Run the code from the previous cell for df_locations_parsed.
df_locations_parsed = df_locations \
    .select(
        col('timestamp'),
        udf_parse_location(df_locations['value']).alias('json_value')
    )
print(df_locations_parsed)

```

```

DataFrame[timestamp: timestamp, json_value: struct<offset:double,id:string,ride_id:string,uuid:string,course:double,latitude:double,longitude:double,geohash:string,speed:double,accuracy:double>]

```

```

[18]: print(df_locations_parsed.printSchema())

```

```

root
 |-- timestamp: timestamp (nullable = true)
 |-- json_value: struct (nullable = true)
 |   |-- offset: double (nullable = true)
 |   |-- id: string (nullable = true)
 |   |-- ride_id: string (nullable = true)
 |   |-- uuid: string (nullable = true)
 |   |-- course: double (nullable = true)
 |   |-- latitude: double (nullable = true)
 |   |-- longitude: double (nullable = true)
 |   |-- geohash: string (nullable = true)
 |   |-- speed: double (nullable = true)
 |   |-- accuracy: double (nullable = true)

```

None

```

[19]: ## Run the code from the previous cell for df_locations_parsed.
df_locations_parsed = df_locations_parsed.select(

```

```

    col('timestamp'),
    col('json_value.ride_id').alias('ride_id'),
    col('json_value.uuid').alias('uuid'),
    col('json_value.speed').alias('speed')
  )
print(df_locations_parsed)

```

DataFrame[timestamp: timestamp, ride\_id: string, uuid: string, speed: double]

```
[20]: print(df_locations_parsed.printSchema())
```

```

root
|-- timestamp: timestamp (nullable = true)
|-- ride_id: string (nullable = true)
|-- uuid: string (nullable = true)
|-- speed: double (nullable = true)

```

None

```

[21]: ## Follow example for windowed speeds from resources
windowedSpeeds = df_locations_parsed \
    .withWatermark("timestamp", "30 seconds") \
    .groupBy(window(df_locations_parsed.timestamp, "30 seconds", "15 seconds"),
    ↪ df_locations_parsed.ride_id,
        df_locations_parsed.speed) \
    .agg(mean(df_locations_parsed.speed).alias("value"),
    ↪ mean(df_locations_parsed.ride_id).alias("key"))

windowedSpeeds

```

```
[21]: DataFrame>window: struct<start:timestamp,end:timestamp>, ride_id: string, speed:
double, value: double, key: double]
```

In the previous Jupyter cells, you should have created the `windowedSpeeds` streaming query. Next, you will need to write that to the `LastnameFirstname-windowed` topic. If you created the `windowsSpeeds` streaming query correctly, the following should publish the results to the `LastnameFirstname-windowed` topic.

```

[22]: ds_locations_windowed = windowedSpeeds \
    .selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)") \
    .writeStream \
    .outputMode("update") \
    .format("kafka") \
    .option("kafka.bootstrap.servers", "kafka.kafka.svc.cluster.local:9092") \
    .option("topic", config['windowed_topic']) \
    .option("checkpointLocation", str(locations_windowed_checkpoint_dir)) \
    .start()

```

```

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

```

```

23/05/13 13:59:43 WARN ResolveWriteToStream: spark.sql.adaptive.enabled is not
supported in streaming DataFrames/Datasets and will be disabled.
23/05/13 13:59:43 WARN AdminClientConfig: The configuration 'key.deserializer'
was supplied but isn't a known config.
23/05/13 13:59:43 WARN AdminClientConfig: The configuration 'value.deserializer'
was supplied but isn't a known config.
23/05/13 13:59:43 WARN AdminClientConfig: The configuration 'enable.auto.commit'
was supplied but isn't a known config.
23/05/13 13:59:43 WARN AdminClientConfig: The configuration 'max.poll.records'
was supplied but isn't a known config.
23/05/13 13:59:43 WARN AdminClientConfig: The configuration 'auto.offset.reset'
was supplied but isn't a known config.
23/05/13 13:59:43 ERROR MicroBatchExecution: Query [id =
1b938d75-1604-4381-88d0-008663ad0849, runId =
7e649eb0-b4d5-4145-a63f-8e2869be34e2] terminated with error
java.lang.NoClassDefFoundError: org/apache/kafka/clients/admin/OffsetSpec
    at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.$anonfun$fetchEarliestOffsets$2(KafkaOffsetReaderAdmin.scala:289)
    at
scala.collection.TraversableLike.$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.$anonfun$fetchEarliestOffsets$1(KafkaOffsetReaderAdmin.scala:289)
    at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.$anonfun$partitionsAssignedToAdmin$1(KafkaOffsetReaderAdmin.scala:501)
    at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.withRetries(KafkaOffsetReaderAdmin.scala:518)
    at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.partitionsAssignedToAdmin(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$sp.java:23)
      at org.apache.spark.sql.Session.withActive(Session.scala:827)
      at org.apache.spark.sql.execution.streaming.StreamExecution.org$apache$s
park$sql$execution$streaming$StreamExecution$$runStream(StreamExecution.scala:28
4)
      at org.apache.spark.sql.execution.streaming.StreamExecution$$anon$1.run(
StreamExecution.scala:207)
Caused by: java.lang.ClassNotFoundException:
org.apache.kafka.clients.admin.OffsetSpec
    ... 58 more
Exception in thread "stream execution thread for [id =
1b938d75-1604-4381-88d0-008663ad0849, runId =
7e649eb0-b4d5-4145-a63f-8e2869be34e2]" java.lang.NoClassDefFoundError:
org/apache/kafka/clients/admin/OffsetSpec
    at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.$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$1(KafkaOffsetReaderAdmin.scala:501)
    at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.withRetries(KafkaOffsetReaderAdmin.scala:518)
    at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.partitionsAssignedToAdmin(KafkaOffsetReaderAdmin.scala:498)
    at org.apache.spark.sql.kafka010.KafkaOffsetReaderAdmin.fetchEarliestOffsets(KafkaOffsetReaderAdmin.scala:288)
    at org.apache.spark.sql.kafka010.KafkaMicroBatchStream.$anonfun$getOrCreateInitialPartitionOffsets$1(KafkaMicroBatchStream.scala:249)
    at scala.Option.getOrElse(Option.scala:189)
    at org.apache.spark.sql.kafka010.KafkaMicroBatchStream.getOrCreateInitialPartitionOffsets(KafkaMicroBatchStream.scala:246)
    at org.apache.spark.sql.kafka010.KafkaMicroBatchStream.initialOffset(KafkaMicroBatchStream.scala:98)
    at org.apache.spark.sql.execution.streaming.MicroBatchExecution.$anonfun$getStartOffset$2(MicroBatchExecution.scala:455)
    at scala.Option.getOrElse(Option.scala:189)
    at org.apache.spark.sql.execution.streaming.MicroBatchExecution.getStartOffset(MicroBatchExecution.scala:455)
    at org.apache.spark.sql.execution.streaming.MicroBatchExecution.$anonfun$constructNextBatch$4(MicroBatchExecution.scala:489)
    at org.apache.spark.sql.execution.streaming.ProgressReporter.reportTimeTaken(ProgressReporter.scala:411)
    at org.apache.spark.sql.execution.streaming.ProgressReporter.reportTimeTaken$(ProgressReporter.scala:409)
    at org.apache.spark.sql.execution.streaming.StreamExecution.reportTimeTaken(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$(IterableLike.scala:73)
    at scala.collection.AbstractIterable.foreach(IterableLike.scala:56)
    at scala.collection.TraversableLike.map(TraversableLike.scala:286)
    at scala.collection.TraversableLike.map$(TraversableLike.scala:279)
    at scala.collection.AbstractTraversable.map(TraversableLike.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.withProgressLocked(MicroBatchExecution.scala:802)
    at org.apache.spark.sql.execution.streaming.MicroBatchExecution.constructNextBatch(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$sp.java:23)
    at org.apache.spark.sql.Session.withActive(Session.scala:827)
    at org.apache.spark.sql.execution.streaming.StreamExecution.org$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[22], line 12

```

1 ds_locations_windowed = windowedSpeeds \
2   .selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)") \
3   .writeStream \
4   (...)
5   .option("checkpointLocation", str(locations_windowed_checkpoint_dir)) \
6   .start()
7
8 try:
9
10  ----> 12 ds_locations_windowed.awaitTermination()
11 except KeyboardInterrupt:
12     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 =
↪1b938d75-1604-4381-88d0-008663ad0849, runId =
↪7e649eb0-b4d5-4145-a63f-8e2869be34e2] terminated with exception: org/apache/
↪kafka/clients/admin/OffsetSpec

```

```
[23]: print(ds_locations_windowed)
```

```
<pyspark.sql.streaming.query.StreamingQuery object at 0x7fd39b82db40>
```

```
[24]: print(type(ds_locations_windowed))
```

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