

Assignment 9.3

February 27, 2022

0.1 Assignment 9.3

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

import pandas as pd

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 parameters to the appropriate values for your setup.

```
[2]: config = dict(
    bootstrap_servers=['kafka.kafka.svc.cluster.local:9092'],
    first_name='Arindam',
```

```

        last_name='Samanta'
    )

    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': 'Arindam',
      'last_name': 'Samanta',
      'client_id': 'SamantaArindam',
      'topic_prefix': 'SamantaArindam',
      'locations_topic': 'SamantaArindam-locations',
      'accelerations_topic': 'SamantaArindam-accelerations',
      'joined_topic': 'SamantaArindam-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 "SamantaArindam-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.

```

[4]: 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']) \
    .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']) \
    .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.

```

[5]: 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

```

[6]: 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")

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('location_timestamp', "2 seconds")

```

```

↳ -----

AnalysisException                                Traceback (most recent call↳
↳ last)

<ipython-input-6-20f0100b02d2> in <module>
    15 .withWatermark('location_timestamp', "2 seconds")
    16
--> 17 accelerationsWithWatermark = df_accelerations \
    18     .select(
    19         col('timestamp').alias('acceleration_timestamp'),

/usr/local/spark/python/pyspark/sql/dataframe.py in withWatermark(self,↳
↳ eventTime, delayThreshold)
    542         if not delayThreshold or type(delayThreshold) is not str:
    543             raise TypeError("delayThreshold should be provided as a↳
↳ string interval")
--> 544         jdf = self._jdf.withWatermark(eventTime, delayThreshold)
    545         return DataFrame(jdf, self.sql_ctx)
    546

/usr/local/spark/python/lib/py4j-0.10.9-src.zip/py4j/java_gateway.py in↳
↳ __call__(self, *args)
    1302
    1303         answer = self.gateway_client.send_command(command)
-> 1304         return_value = get_return_value(
    1305             answer, self.gateway_client, self.target_id, self.name)

```

1306

```
/usr/local/spark/python/pyspark/sql/utils.py in deco(*a, **kw)
135             # Hide where the exception came from that shows a
↳non-Pythonic
136             # JVM exception message.
--> 137             raise_from(converted)
138         else:
139             raise
```

```
/usr/local/spark/python/pyspark/sql/utils.py in raise_from(e)
```

```
AnalysisException: cannot resolve ``location_timestamp`` given input
↳columns: [acceleration_ride_id, acceleration_timestamp, x, y, z];;
'EventTimeWatermark 'location_timestamp, 2 seconds
+- Project [acceleration_timestamp#66, json_value#68.ride_id AS
↳acceleration_ride_id#71, json_value#68.x AS x#72, json_value#68.y AS y#73,
↳json_value#68.z AS z#74]
+- Project [timestamp#33 AS acceleration_timestamp#66, <lambda>(value#29)
↳AS json_value#68]
+- StreamingRelationV2 org.apache.spark.sql.kafka010.
↳KafkaSourceProvider@3fe07403, kafka, org.apache.spark.sql.kafka010.
↳KafkaSourceProvider$KafkaTable@34afae78, org.apache.spark.sql.util.
↳CaseInsensitiveStringMap@6af95ba5, [key#28, value#29, topic#30, partition#31,
↳offset#32L, timestamp#33, timestampType#34], StreamingRelation DataSource(org.
↳apache.spark.sql.
↳SparkSession@2e135f13,kafka,List(),None,List(),None,Map(subscribe ->
↳SamantaArindam-accelerations, kafka.bootstrap.servers -> kafka.kafka.svc.
↳cluster.local:9092),None), kafka, [key#21, value#22, topic#23, partition#24,
↳offset#25L, timestamp#26, timestampType#27]
```

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.

```
[ ]: df_joined = ''
df_joined
```

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.

```

[ ]: ds_joined = df_joined \
    .withColumn(
        'value',
        to_json(
            struct(
                'ride_id', 'location_timestamp', 'speed',
                'latitude', 'longitude', 'geohash', 'accuracy',
                'acceleration_timestamp', 'x', 'y', 'z'
            )
        )
    ).withColumn(
        'key', col('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")

```