```
from pyspark.sql import SparkSession
from pyspark.sql.avro.functions import from_avro, to_avro
from pyspark.sql.functions import *
from pyspark.sql.types import *
import json
from pyspark.ml import Pipeline,PipelineModel

#Spark Session creation configured to interact with Kfka and MongoDB
spark = SparkSession.builder.appName("pyspark-notebook").\
config("spark.jars.packages","org.apache.spark:spark-sql-kafka-0-
10_2.12:3.0.0,org.apache.spark:spark-
avro_2.12:3.0.0,org.mongodb.spark:mongo-spark-connector_2.12:3.0.0").\
config("spark.mongodb.input.uri","mongodb://docker_mongo_1:27017/twitt
er_db.tweets").\
config("spark.mongodb.output.uri","mongodb://docker_mongo_1:27017/
twitter_db.tweets").\
getOrCreate()

#Read schema file and create schema of string type
json_schema = ''
with open("schema/out/tweet_schema.json") as f:
    new_schema = StructType.fromJson(json.load(f))
    json_schema = new_schema.simpleString()

#Read data from Kafka topic
json_tweets = spark\
  .readStream\
  .format("kafka")\
  .option("kafka.bootstrap.servers", "ec2-34-217-75-40.us-west-
2.compute.amazonaws.com:9092")\
  .option("subscribe", "twitter_demo")\
  .option("startingOffsets", "earliest")\
  .load()\
  .selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")

#Refine raw data red from Kafka topic
refined_tweets = json_tweets\
        .select(from_json("value", json_schema)\
        .alias("data"))\
        .where("data.lang='en'and data.created_at is not null and
data.text is not null")\
        .select("data.text",
                from_unixtime(col("data.timestamp_ms")/1000,'yyyy-MM-
dd HH:mm:ss').alias("timestamp_ms")) #Translate milliseconds to UTC
timestamp
refined_tweets = refined_tweets.withColumn('text',
regexp_replace('text', r'http\S+', ''))
refined_tweets = refined_tweets.withColumn('text',
regexp_replace('text', '@\w+', ''))
refined_tweets = refined_tweets.withColumn('text',
```

```
regexp_replace('text', '#', ''))
refined_tweets = refined_tweets.withColumn('text',
regexp_replace('text', 'RT', ''))
refined_tweets = refined_tweets.withColumn('text',
regexp_replace('text', ':', ''))

dir = "sentiment/"
model = PipelineModel.load(dir)

def process_row(df, epoch_id):
    """Applies model to the df and writes data to MongoDB

    Parameters
    ----------
    df : DataFrame
        Streaming Dataframe
    epoch_id : int
        Unique id for each micro batch/epoch
    """
    predictions = model.transform(df)
    #predictions.show()

predictions.select("timestamp_ms","text","prediction").write.format("m
ongo").mode("append").save()

#Writes streaming dataframe to ForeachBatch console which ingests data
to MongoDB
refined_tweets \
    .writeStream \
    .option("checkpointLocation", "checkpoint/data") \
    .foreachBatch(process_row).start().awaitTermination()

---------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
last)
<ipython-input-8-81c4a13a4b98> in <module>
      3     .writeStream \
      4     .option("checkpointLocation", "checkpoint/data") \
----> 5     .foreachBatch(process_row).start().awaitTermination()

/usr/local/lib/python3.7/dist-packages/pyspark/sql/streaming.py in
awaitTermination(self, timeout)
    101             return self._jsq.awaitTermination(int(timeout *
1000))
    102         else:
--> 103             return self._jsq.awaitTermination()
    104
    105     @property

/usr/local/lib/python3.7/dist-packages/py4j/java_gateway.py in
```

```
__call__(self, *args)
   1301             proto.END_COMMAND_PART
   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)

/usr/local/lib/python3.7/dist-packages/py4j/java_gateway.py in
send_command(self, command, retry, binary)
   1031         connection = self._get_connection()
   1032         try:
-> 1033             response = connection.send_command(command)
   1034             if binary:
   1035                 return response,
self._create_connection_guard(connection)

/usr/local/lib/python3.7/dist-packages/py4j/java_gateway.py in
send_command(self, command)
   1198
   1199         try:
-> 1200             answer = smart_decode(self.stream.readline()[:-1])
   1201             logger.debug("Answer received:
{0}".format(answer))
   1202             if answer.startswith(proto.RETURN_MESSAGE):

/usr/lib/python3.7/socket.py in readinto(self, b)
    587         while True:
    588             try:
--> 589                 return self._sock.recv_into(b)
    590             except timeout:
    591                 self._timeout_occurred = True

KeyboardInterrupt:
```