1

```
import time
class invoiceStreamBatch():
    def __init__(self):
        self.base_data_dir = "/FileStore/data_spark_streaming_scholarnest"

    def cleanData(self):
        spark.sql("drop table if exists invoice_line_items")
        dbutils.fs.rm("/user/hive/warehouse/invoice_line_items", True)
        dbutils.fs.rm(f"{self.base_data_dir}/checkpoint/invoices", True)
        dbutils.fs.ls("/FileStore/data_spark_streaming_scholarnest/data/invoices/")
        #input("Enter A Key!")
        #dbutils.fs.cp(f"{self.base_data_dir}/datasets/invoices/*.json", f"{self.base_data_dir}/data/invoices/")
    def getSchema(self):
        return """InvoiceNumber string, CreatedTime bigint, StoreID string, PosID string, CashierID string,
                CustomerType string, CustomerCardNo string, TotalAmount double, NumberOfItems bigint,
                PaymentMethod string, TaxableAmount double, CGST double, SGST double, CESS double,
                DeliveryType string,
                DeliveryAddress struct<AddressLine string, City string, ContactNumber string, PinCode string,
                State string>,
                InvoiceLineItems array<struct<ItemCode string, ItemDescription string,
                    ItemPrice double, ItemQty bigint, TotalValue double>>
            """
    def readInvoices(self):
        return (spark.readStream
                    .format("json")
                    .schema(self.getSchema())
                    .load(f"{self.base_data_dir}/data/invoices/")
        )
    def explodeInvoices(self, invoiceDF):
        return ( invoiceDF.selectExpr("InvoiceNumber", "CreatedTime", "StoreID", "PosID",
                                      "CustomerType", "PaymentMethod", "DeliveryType", "DeliveryAddress.City",
                                      "DeliveryAddress.State","DeliveryAddress.PinCode",
                                      "explode(InvoiceLineItems) as LineItem")
                                            )
    def flattenInvoices(self, explodedDF):
        from pyspark.sql.functions import expr
        return( explodedDF.withColumn("ItemCode", expr("LineItem.ItemCode"))
                        .withColumn("ItemDescription", expr("LineItem.ItemDescription"))
                        .withColumn("ItemPrice", expr("LineItem.ItemPrice"))
                        .withColumn("ItemQty", expr("LineItem.ItemQty"))
                        .withColumn("TotalValue", expr("LineItem.TotalValue"))
                        .drop("LineItem")
                )
    def appendInvoices(self, flattenedDF, trigger="batch"):
        sQuery = (flattenedDF.writeStream
                    .format("delta")
                    .option("checkpointLocation", f"{self.base_data_dir}/checkpoint/invoices")
                    .outputMode("append")
                    .option("maxFilesPerTrigger", 1)
        )
        if trigger == "batch":
            return (sQuery.trigger(availableNow=True)
                   .toTable("invoice_table_items"))
        else:
            print("Coming Here!")
            return (sQuery.trigger(processingTime=trigger)
```

```
                    .toTable("invoice_table_items"))


    def process(self, trigger = "batch"):
           print(f"Starting Invoice Processing Stream...", end='')
           invoicesDF = self.readInvoices()
           explodedDF = self.explodeInvoices(invoicesDF)
           resultDF = self.flattenInvoices(explodedDF)
           sQuery = self.appendInvoices(resultDF, trigger)
           time.sleep(30)
           print("Done\n")

           return sQuery
```

2

```
iStream = invoiceStreamBatch()
iStream.process("30 seconds")
import time
time.sleep(30)

count = spark.sql("SELECT COUNT(*) FROM invoice_table_items").collect()[0][0]
print("count = ", count)
```

▸ beabb41d-3ecd-4ec3-aee0-3e1bab640b2f *Last updated: 24 hours ago*

```
Starting Invoice Processing Stream...Coming Here!
Done

count =  0
```