Project1 Description :

Spark Streaming can be used for Real time Analysis. I used this feature of Spark Streaming along with Spark SQL to analyse real time Stock Market Data. We can feed amount of real time stock data to DataFrame API of Spark for real time processing.

Folder Description :

Data : Dataset used in Project : aaon.us.txt, cetv.us.txt, hmta.us.txt, admp.us.txt, gsbc.us.txt
Jar : jar file for Project1 - Project1-1.0-SNAPSHOT.jar
SourceCode : Zip folder for Source Code

Goal : Used the datasets of company stock in the data folder and calculate Average Closing price/year and calculate company with highest closing price/year.

The entry point into all functionality in Spark is the Spark Session class. To create a basic SparkSession, just use SparkSession.builder():

```
SparkSession spark = SparkSession.builder().appName("Spark SQL").config("spark.some.config.option", "some-value").master("local[*]").getOrCreate();
```

```
SQLContext sqlcontext = new org.apache.spark.sql.SQLContext(spark);
```

I created a java bean class for Stock :

Spark SQL supports automatically converting an RDD of Java beans into a DataFrame. The BeanInfo, obtained using reflection, defines the schema of the table.

```
JavaRDD<Stock> aaonstockRDD = spark.read()
      .textFile("/home/2018/spring/nyu/6513/iv447/Project1/data/aaon.us.txt")
      .javaRDD()
      .map(line -> {
         String[] parts = line.split(",");
         Stock stock = new Stock();
         stock.setDt(parts[0]);
         stock.setOpenprice(Double.parseDouble(parts[1].trim()));
         stock.setHighprice(Double.parseDouble(parts[2].trim()));
         stock.setLowprice(Double.parseDouble(parts[3].trim()));
         stock.setCloseprice(Double.parseDouble(parts[4].trim()));
         stock.setVolume(Double.parseDouble(parts[5].trim()));
         stock.setAdjcloseprice(Double.parseDouble(parts[6].trim()));
         return stock;
      });
```

We can create Dataset from the above RDD :

```
Dataset<Row> aaonstockDF = spark.createDataFrame(aaonstockRDD, Stock.class);
```

We can create following datasets the same way :

```
Dataset<Row> admpstockDF = spark.createDataFrame(admpstockRDD, Stock.class);
Dataset<Row> cetvstockDF = spark.createDataFrame(cetvstockRDD, Stock.class);
Dataset<Row> gsbcstockDF = spark.createDataFrame(gsbcstockRDD, Stock.class);
Dataset<Row> hmtastockDF = spark.createDataFrame(hmtastockRDD, Stock.class);
```

Creating temporary table view from Dataset :

```
aaonstockDF.createTempView("AAONSTOCK");
admpstockDF.createTempView("ADMPSTOCK");
cetvstockDF.createTempView("CETVSTOCK");
gsbcstockDF.createTempView("GSBCSTOCK");
hmtastockDF.createTempView("HMTASTOCK");
```

We can query the above created Tables :

```
Dataset<Row> aaonstock_result = spark.sql("SELECT dt, openprice, highprice, lowprice, closeprice,volume FROM AAONSTOCK WHERE abs(closeprice - Openprice) > 2");
aaonstock_result.show();
```

Query : For joining table

```
Dataset<Row> stock_join_result = spark.sql("SELECT AAONSTOCK.dt, AAONSTOCK.closeprice as AAONClose , ADMPSTOCK.closeprice as ADMPClose, CETVSTOCK.closeprice as CETVClose from AAONSTOCK join ADMPSTOCK on AAONSTOCK.dt = ADMPSTOCK.dt join CETVSTOCK on AAONSTOCK.dt = CETVSTOCK.dt" );
stock_join_result.show();
```

We can also store the result in parquet table and then read those parquet tables and create dataframe from it :

Eg:
Saving the stock join result in parquet table :

```
stock_join_result.write().mode(SaveMode.Overwrite).parquet("/home/2018/spring/nyu/6513/iv447/Project1/data/join_result.parquet");
```

Reading from Parquet Table :

```
Dataset<Row> parquetFileDF = spark.read().parquet("/home/2018/spring/nyu/6513/iv447/Project1/data/join_result.parquet");
parquetFileDF.createTempView("ParquetTable");
```

Final Queries :

calculate Average Closing price/year

```
Dataset<Row> newTable = sqlcontext.sql("SELECT year(ParquetTable.dt) as year, avg(ParquetTable.AAONClose) as AAON, avg(ParquetTable.ADMPClose) as ADMP, avg(ParquetTable.CETVClose) as CETV from ParquetTable group by year(ParquetTable.dt) order by year(ParquetTable.dt)");
newTable.show();
newTable.createTempView("NEWTable");
```

calculate comany with highest closing price/year.

```
Dataset<Row> BestCompany = sqlcontext.sql("SELECT year, GREATEST((AAON),(ADMP),(CETV)) as BestCompany FROM NEWTable");
BestCompany.show();
```

Running the jar from IBM Cloud :

Goto Spark bin folder and execute following command :

```
./spark-submit --master local[*] --class StockAnalysis $HOME/Project1/jar/Project1-1.0-SNAPSHOT.jar
```