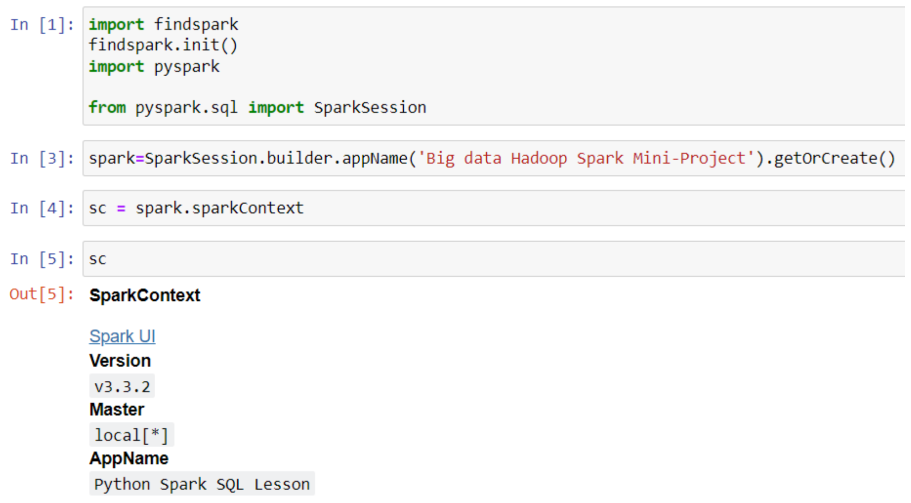
Spark Walmart STOCK Data Analysis Project

1. Start a simple Spark Session



```
In [1]: import findspark
        findspark.init()
        import pyspark

        from pyspark.sql import SparkSession

In [3]: spark=SparkSession.builder.appName('Big data Hadoop Spark Mini-Project').getOrCreate()

In [4]: sc = spark.sparkContext

In [5]: sc
```

Out[5]: **SparkContext**

Spark UI

Version

`v3.3.2`

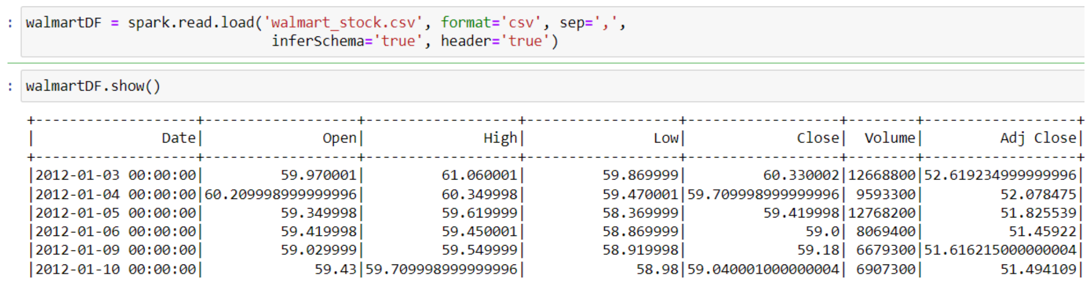
Master

`local[*]`

AppName

`Python Spark SQL Lesson`

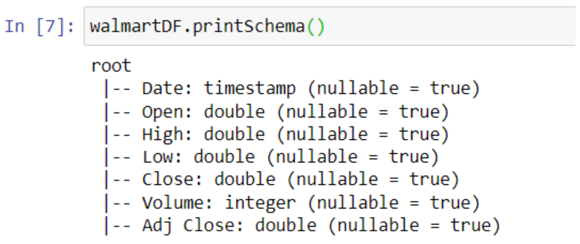
2. Load the Walmart Stock CSV File, have Spark infer the data types. What are the column names?



```
: walmartDF = spark.read.load('walmart_stock.csv', format='csv', sep=',',
                             inferSchema='true', header='true')

: walmartDF.show()
```

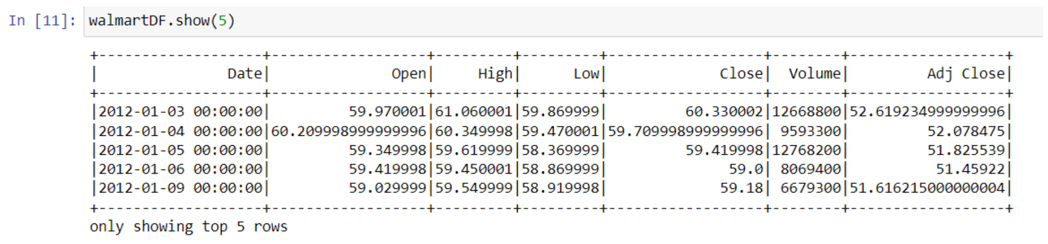
```
+-------------------+------------------+------------------+------------------+------------------+--------+------------------+
|               Date|              Open|              High|               Low|             Close|  Volume|         Adj Close|
+-------------------+------------------+------------------+------------------+------------------+--------+------------------+
|2012-01-03 00:00:00|         59.970001|         61.060001|         59.869999|         60.330002|12668800|52.619234999999996|
|2012-01-04 00:00:00|60.209998999999996|         60.349998|         59.470001|59.709998999999996| 9593300|         52.078475|
|2012-01-05 00:00:00|         59.349998|         59.619999|         58.369999|         59.419998|12768200|         51.825539|
|2012-01-06 00:00:00|         59.419998|         59.450001|         58.869999|              59.0| 8069400|          51.45922|
|2012-01-09 00:00:00|         59.029999|         59.549999|         58.919998|             59.18| 6679300|51.616215000000004|
|2012-01-10 00:00:00|             59.43|59.709998999999996|             58.98|59.040001000000004| 6907300|         51.494109|
```

3. What does the Schema look like?



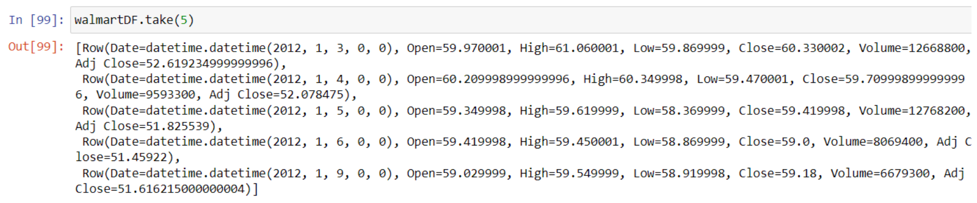
```
In [7]: walmartDF.printSchema()

root
 |-- Date: timestamp (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: integer (nullable = true)
 |-- Adj Close: double (nullable = true)
```

4. Print out the first 5 columns.



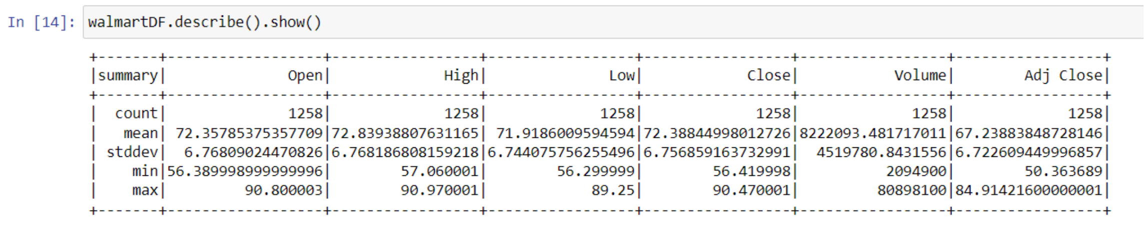
```
In [11]: walmartDF.show(5)

+-------------------+------------------+---------+---------+------------------+--------+------------------+
|               Date|              Open|     High|      Low|             Close|  Volume|         Adj Close|
+-------------------+------------------+---------+---------+------------------+--------+------------------+
|2012-01-03 00:00:00|         59.970001|61.060001|59.869999|         60.330002|12668800|52.619234999999996|
|2012-01-04 00:00:00|60.209998999999996|60.349998|59.470001|59.709998999999996| 9593300|         52.078475|
|2012-01-05 00:00:00|         59.349998|59.619999|58.369999|         59.419998|12768200|         51.825539|
|2012-01-06 00:00:00|         59.419998|59.450001|58.869999|              59.0| 8069400|          51.45922|
|2012-01-09 00:00:00|         59.029999|59.549999|58.919998|             59.18| 6679300|51.616215000000004|
+-------------------+------------------+---------+---------+------------------+--------+------------------+
only showing top 5 rows
```

OR



```
In [99]: walmartDF.take(5)

Out[99]: [Row(Date=datetime.datetime(2012, 1, 3, 0, 0), Open=59.970001, High=61.060001, Low=59.869999, Close=60.330002, Volume=12668800,
         Adj Close=52.619234999999996),
          Row(Date=datetime.datetime(2012, 1, 4, 0, 0), Open=60.209998999999996, High=60.349998, Low=59.470001, Close=59.70999899999999
         6, Volume=9593300, Adj Close=52.078475),
          Row(Date=datetime.datetime(2012, 1, 5, 0, 0), Open=59.349998, High=59.619999, Low=58.369999, Close=59.419998, Volume=12768200,
         Adj Close=51.825539),
          Row(Date=datetime.datetime(2012, 1, 6, 0, 0), Open=59.419998, High=59.450001, Low=58.869999, Close=59.0, Volume=8069400, Adj C
         lose=51.45922),
          Row(Date=datetime.datetime(2012, 1, 9, 0, 0), Open=59.029999, High=59.549999, Low=58.919998, Close=59.18, Volume=6679300, Adj
         Close=51.616215000000004)]
```

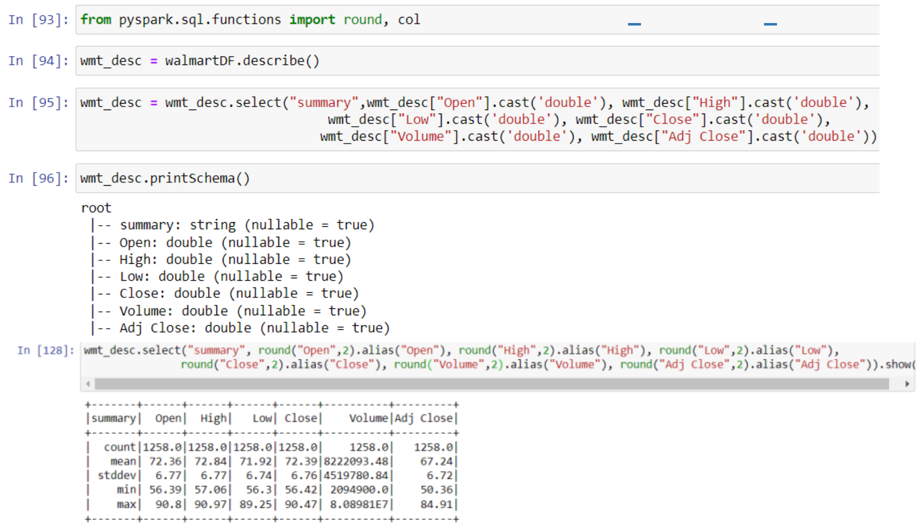
5. **Use describe() to learn about the DataFrame.**



```
In [14]: walmartDF.describe().show()

         +-------+------------------+-----------------+-----------------+-----------------+-----------------+-----------------+
         |summary|              Open|             High|              Low|            Close|           Volume|        Adj Close|
         +-------+------------------+-----------------+-----------------+-----------------+-----------------+-----------------+
         |  count|              1258|             1258|             1258|             1258|             1258|             1258|
         |   mean| 72.35785375357709|72.83938807631165| 71.9186009594594|72.38844998012726|8222093.481717011|67.23883848728146|
         | stddev|  6.76809024470826|6.768186808159218|6.744075756255496|6.756859163732991|  4519780.8431556|6.722609449996857|
         |    min|56.389998999999996|        57.060001|        56.299999|        56.419998|          2094900|        50.363689|
         |    max|         90.800003|        90.970001|            89.25|        90.470001|         80898100|84.91421600000001|
         +-------+------------------+-----------------+-----------------+-----------------+-----------------+-----------------+
```

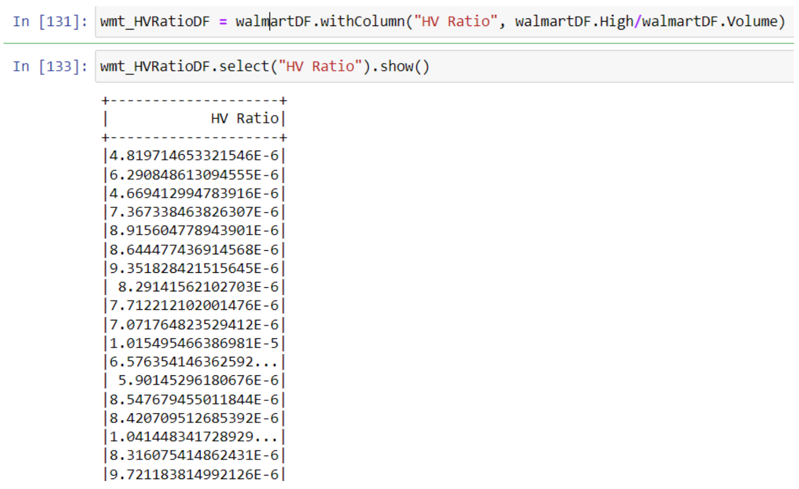
6. **Bonus Question!**

There are too many decimal places for mean and stddev in the describe() dataframe. Format the numbers to just show up to two decimal places. Pay careful attention to the datatypes that .describe() returns, we didn't cover how to do this exact formatting, but we covered something very similar.



```
In [93]: from pyspark.sql.functions import round, col

In [94]: wmt_desc = walmartDF.describe()

In [95]: wmt_desc = wmt_desc.select("summary",wmt_desc["Open"].cast('double'), wmt_desc["High"].cast('double'),
                                    wmt_desc["Low"].cast('double'), wmt_desc["Close"].cast('double'),
                                   wmt_desc["Volume"].cast('double'), wmt_desc["Adj Close"].cast('double'))

In [96]: wmt_desc.printSchema()

         root
          |-- summary: string (nullable = true)
          |-- Open: double (nullable = true)
          |-- High: double (nullable = true)
          |-- Low: double (nullable = true)
          |-- Close: double (nullable = true)
          |-- Volume: double (nullable = true)
          |-- Adj Close: double (nullable = true)

In [128]: wmt_desc.select("summary", round("Open",2).alias("Open"), round("High",2).alias("High"), round("Low",2).alias("Low"),
                          round("Close",2).alias("Close"), round("Volume",2).alias("Volume"), round("Adj Close",2).alias("Adj Close")).show(

          +-------+------+------+------+------+----------+---------+
          |summary|  Open|  High|   Low| Close|    Volume|Adj Close|
          +-------+------+------+------+------+----------+---------+
          |  count|1258.0|1258.0|1258.0|1258.0|    1258.0|   1258.0|
          |   mean| 72.36| 72.84| 71.92| 72.39|8222093.48|    67.24|
          | stddev|  6.77|  6.77|  6.74|  6.76|4519780.84|     6.72|
          |    min| 56.39| 57.06|  56.3| 56.42| 2094900.0|    50.36|
          |    max|  90.8| 90.97| 89.25| 90.47| 8.08981E7|    84.91|
          +-------+------+------+------+------+----------+---------+
```

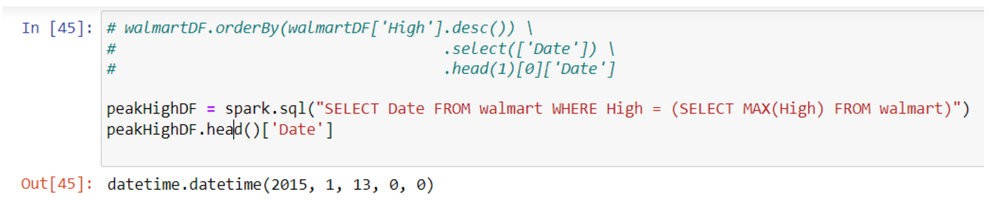
7. **Create a new dataframe with a column called HV Ratio that is the ratio of the High Price versusvolume of stock traded for a day.**



```
In [131]: wmt_HVRatioDF = walmartDF.withColumn("HV Ratio", walmartDF.High/walmartDF.Volume)

In [133]: wmt_HVRatioDF.select("HV Ratio").show()

          +--------------------+
          |            HV Ratio|
          +--------------------+
          |4.819714653321546E-6|
          |6.290848613094555E-6|
          |4.669412994783916E-6|
          |7.367338463826307E-6|
          |8.915604778943901E-6|
          |8.644477436914568E-6|
          |9.351828421515645E-6|
          | 8.29141562102703E-6|
          |7.712212102001476E-6|
          |7.071764823529412E-6|
          |1.015495466386981E-5|
          |6.576354146362592...|
          | 5.90145296180676E-6|
          |8.547679455011844E-6|
          |8.420709512685392E-6|
          |1.041448341728929...|
          |8.316075414862431E-6|
          |9.721183814992126E-6|
```

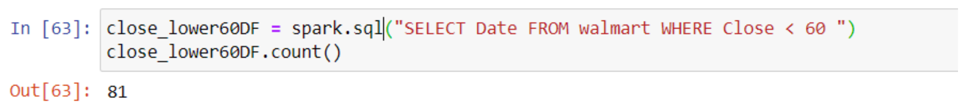
===================================   PROBLEM STATEMENTS ================================

1. What day had the Peak High in Price?



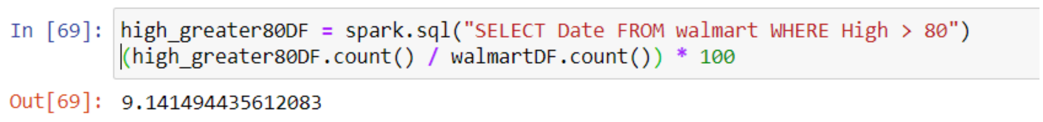
```
In [45]: # walmartDF.orderBy(walmartDF['High'].desc()) \
         #                                     .select(['Date']) \
         #                                     .head(1)[0]['Date']

         peakHighDF = spark.sql("SELECT Date FROM walmart WHERE High = (SELECT MAX(High) FROM walmart)")
         peakHighDF.head()['Date']

Out[45]: datetime.datetime(2015, 1, 13, 0, 0)
```

2. How many days was the Close lower than 60 dollars?



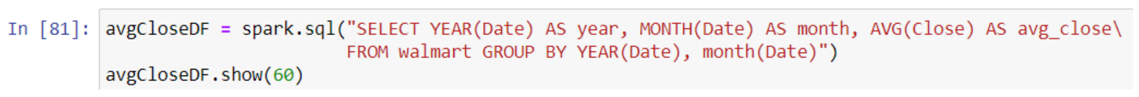
```
In [63]: close_lower60DF = spark.sql("SELECT Date FROM walmart WHERE Close < 60 ")
         close_lower60DF.count()

Out[63]: 81
```

3. What percentage of the time was the High greater than 80 dollars ?

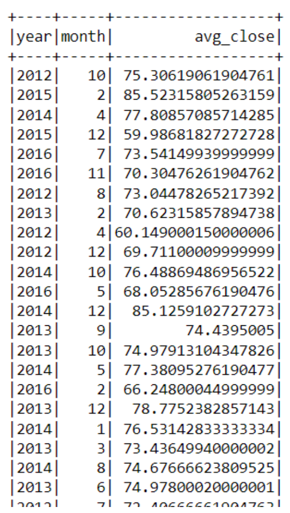


```
In [69]: high_greater80DF = spark.sql("SELECT Date FROM walmart WHERE High > 80")
         (high_greater80DF.count() / walmartDF.count()) * 100

Out[69]: 9.141494435612083
```

4. What is the average Close for each Calendar Month?

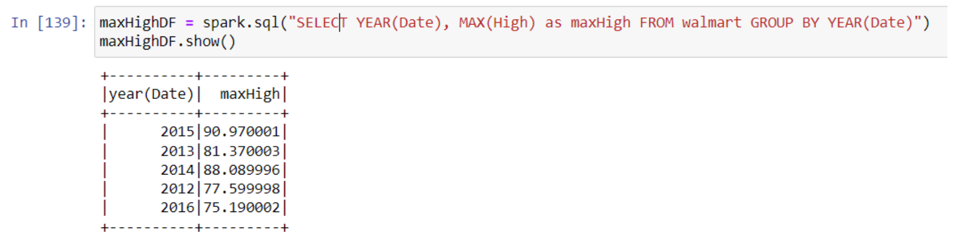


```
In [81]: avgCloseDF = spark.sql("SELECT YEAR(Date) AS year, MONTH(Date) AS month, AVG(Close) AS avg_close\
                                FROM walmart GROUP BY YEAR(Date), month(Date)")
         avgCloseDF.show(60)
```



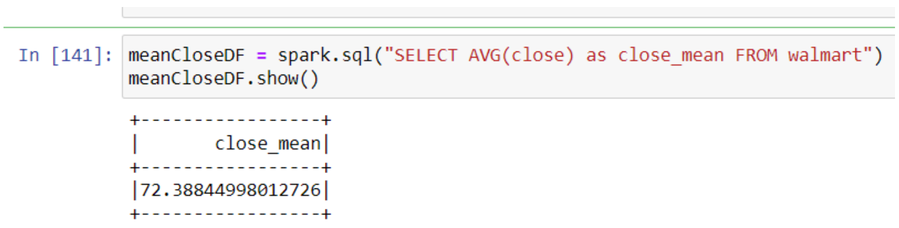
```
+----+-----+-----------------+
|year|month|        avg_close|
+----+-----+-----------------+
|2012|   10| 75.30619061904761|
|2015|    2| 85.52315805263159|
|2014|    4| 77.80857085714285|
|2015|   12| 59.98681827272728|
|2016|    7| 73.54149939999999|
|2016|   11| 70.30476261904762|
|2012|    8| 73.04478265217392|
|2013|    2| 70.62315857894738|
|2012|    4|60.149000150000006|
|2012|   12| 69.71100009999999|
|2014|   10| 76.48869486956522|
|2016|    5| 68.05285676190476|
|2014|   12|  85.1259102727273|
|2013|    9|        74.4395005|
|2013|   10| 74.97913104347826|
|2014|    5| 77.38095276190477|
|2016|    2| 66.24800044999999|
|2013|   12|  78.7752382857143|
|2014|    1| 76.53142833333334|
|2013|    3| 73.43649940000002|
|2014|    8| 74.67666623809525|
|2013|    6| 74.97800020000001|
```

5. What is the max High per year?



```
In [139]: maxHighDF = spark.sql("SELECT YEAR(Date), MAX(High) as maxHigh FROM walmart GROUP BY YEAR(Date)")
          maxHighDF.show()

+----------+---------+
|year(Date)|  maxHigh|
+----------+---------+
|      2015|90.970001|
|      2013|81.370003|
|      2014|88.089996|
|      2012|77.599998|
|      2016|75.190002|
+----------+---------+
```

6. What is the mean of the Close column?

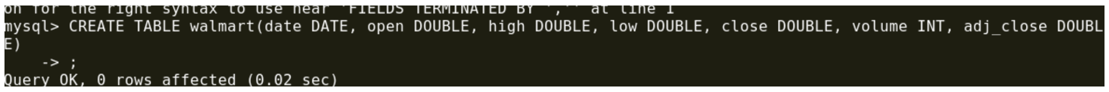


```
In [141]: meanCloseDF = spark.sql("SELECT AVG(close) as close_mean FROM walmart")
          meanCloseDF.show()

+-----------------+
|       close_mean|
+-----------------+
|72.38844998012726|
+-----------------+
```

** using HIVE ***

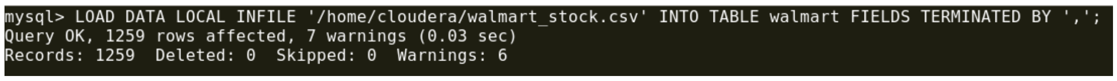
1. Put the data in MySQL

1.1 CREATE TABLE in MySQL:
CREATE TABLE walmart(date DATE, open DOUBLE, high DOUBLE, low DOUBLE, close DOUBLE, volume INT, adj_close DOUBLE);



```
on for the right syntax to use near 'FIELDS TERMINATED BY ','' at line 1
mysql> CREATE TABLE walmart(date DATE, open DOUBLE, high DOUBLE, low DOUBLE, close DOUBLE, volume INT, adj_close DOUBL
E)
    -> ;
Query OK, 0 rows affected (0.02 sec)
```

1.2 Loading file from LFS (VM) --------> MySQL:
LOAD DATA LOCAL INFILE '/home/cloudera/walmart_stock.csv' INTO TABLE walmart FIELDS TERMINATED BY ',';

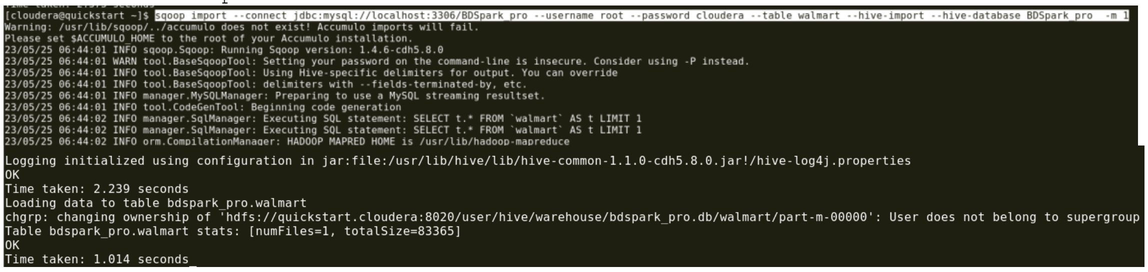


```
mysql> LOAD DATA LOCAL INFILE '/home/cloudera/walmart_stock.csv' INTO TABLE walmart FIELDS TERMINATED BY ',';
Query OK, 1259 rows affected, 7 warnings (0.03 sec)
Records: 1259  Deleted: 0  Skipped: 0  Warnings: 6
```

===

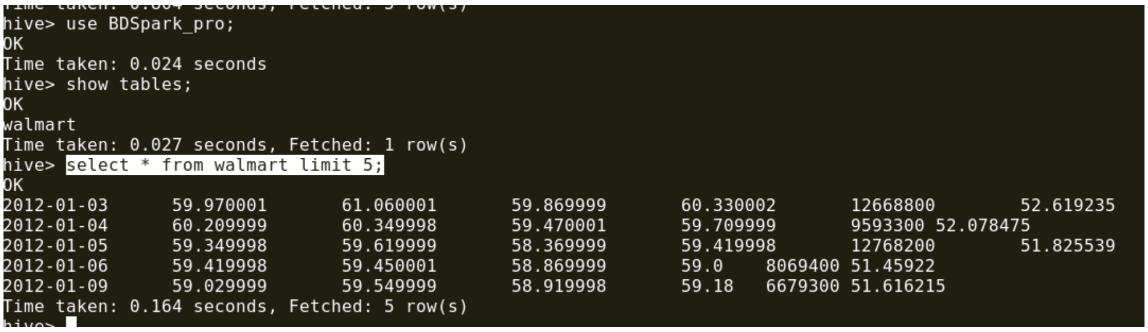
2. SQOOP Pipeline for MySQL -----------> Hive:

sqoop import --connect jdbc:mysql://localhost:3306/BDSpark_pro --username root --password cloudera --table walmart --hive-import -m 1



```
[cloudera@quickstart ~]$ sqoop import --connect jdbc:mysql://localhost:3306/BDSpark_pro --username root --password cloudera --table walmart --hive-import --hive-database BDSpark_pro  -m 1
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
23/05/25 06:44:01 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.8.0
23/05/25 06:44:01 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
23/05/25 06:44:01 INFO tool.BaseSqoopTool: Using Hive-specific delimiters for output. You can override
23/05/25 06:44:01 INFO tool.BaseSqoopTool: delimiters with --fields-terminated-by, etc.
23/05/25 06:44:01 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
23/05/25 06:44:01 INFO tool.CodeGenTool: Beginning code generation
23/05/25 06:44:02 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `walmart` AS t LIMIT 1
23/05/25 06:44:02 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `walmart` AS t LIMIT 1
23/05/25 06:44:02 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/lib/hadoop-mapreduce

Logging initialized using configuration in jar:file:/usr/lib/hive/lib/hive-common-1.1.0-cdh5.8.0.jar!/hive-log4j.properties
OK
Time taken: 2.239 seconds
Loading data to table bdspark_pro.walmart
chgrp: changing ownership of 'hdfs://quickstart.cloudera:8020/user/hive/warehouse/bdspark_pro.db/walmart/part-m-00000': User does not belong to supergroup
Table bdspark_pro.walmart stats: [numFiles=1, totalSize=83365]
OK
Time taken: 1.014 seconds
```

Table in Hive:



```
hive> use BDSpark_pro;
OK
Time taken: 0.024 seconds
hive> show tables;
OK
walmart
Time taken: 0.027 seconds, Fetched: 1 row(s)
hive> select * from walmart limit 5;
OK
2012-01-03      59.970001       61.060001       59.869999       60.330002       12668800        52.619235
2012-01-04      60.209999       60.349998       59.470001       59.709999       9593300 52.078475
2012-01-05      59.349998       59.619999       58.369999       59.419998       12768200        51.825539
2012-01-06      59.419998       59.450001       58.869999       59.0    8069400 51.45922
2012-01-09      59.029999       59.549999       58.919998       59.18   6679300 51.616215
Time taken: 0.164 seconds, Fetched: 5 row(s)
hive>
```

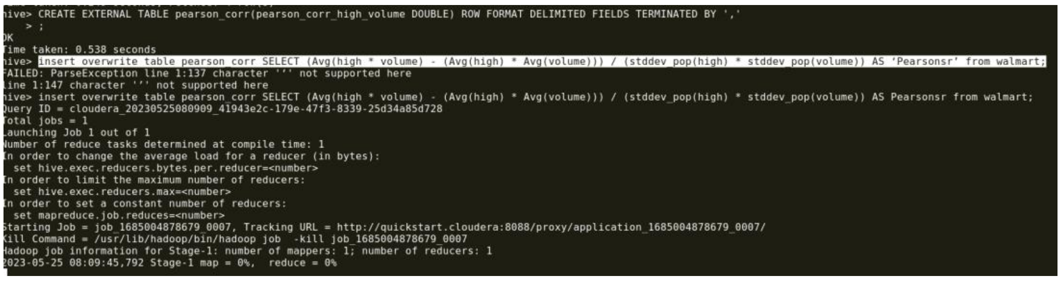
==
=======

3. Performing Queries:

7. What is the Pearson correlation between High and Volume?

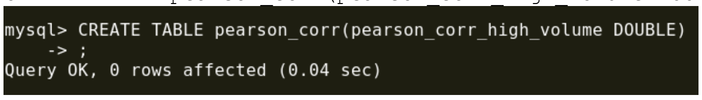
7.1-> Ext Table Creation:
CREATE EXTERNAL TABLE pearson_corr(pearson_corr_high_volume DOUBLE) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

7.2-> Transferring o/p to ext table:
insert overwrite table pearson_corr SELECT (Avg(high * volume) - (Avg(high) * Avg(volume))) / (stddev_pop(high) * stddev_pop(volume)) AS 'Pearsonsr' from walmart;



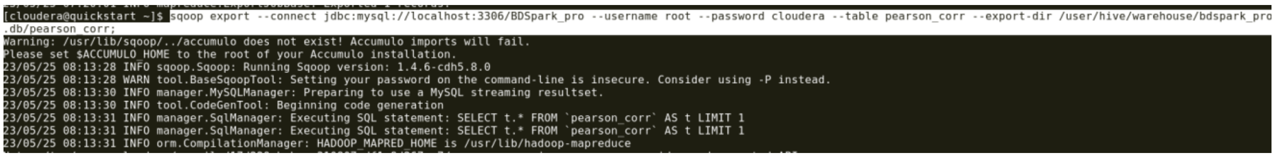
```
hive> CREATE EXTERNAL TABLE pearson_corr(pearson_corr_high_volume DOUBLE) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
    > ;
OK
Time taken: 0.538 seconds
hive> insert overwrite table pearson_corr SELECT (Avg(high * volume) - (Avg(high) * Avg(volume))) / (stddev_pop(high) * stddev_pop(volume)) AS 'Pearsonsr' from walmart;
FAILED: ParseException line 1:137 character ''' not supported here
line 1:147 character ''' not supported here
hive> insert overwrite table pearson_corr SELECT (Avg(high * volume) - (Avg(high) * Avg(volume))) / (stddev_pop(high) * stddev_pop(volume)) AS Pearsonsr from walmart;
Query ID = cloudera_20230525080909_41943e2c-179e-47f3-8339-25d34a85d728
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1685004878679_0007, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1685004878679_0007/
Kill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1685004878679_0007
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2023-05-25 08:09:45,792 Stage-1 map = 0%,  reduce = 0%
```

7.3-> MySQL Table Created (Client's DB):
CREATE TABLE pearson_corr(pearson_corr_high_volume DOUBLE);



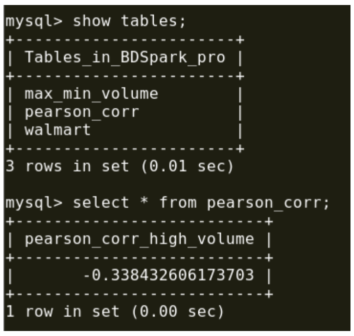
```
mysql> CREATE TABLE pearson_corr(pearson_corr_high_volume DOUBLE)
    -> ;
Query OK, 0 rows affected (0.04 sec)
```

7.4-> Sqoop command to transfer o/p table Hive ----------> MySQL:
sqoop export --connect jdbc:mysql://localhost:3306/BDSpark_pro --username root --password cloudera --table max_min_volume --export-dir /user/hive/warehouse/bdspark_pro.db/max_min_volume;



```
[cloudera@quickstart ~]$ sqoop export --connect jdbc:mysql://localhost:3306/BDSpark_pro --username root --password cloudera --table pearson_corr --export-dir /user/hive/warehouse/bdspark_pro
.db/pearson_corr;
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
23/05/25 08:13:28 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.8.0
23/05/25 08:13:28 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
23/05/25 08:13:30 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
23/05/25 08:13:30 INFO tool.CodeGenTool: Beginning code generation
23/05/25 08:13:31 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `pearson_corr` AS t LIMIT 1
23/05/25 08:13:31 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `pearson_corr` AS t LIMIT 1
23/05/25 08:13:31 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/lib/hadoop-mapreduce
```

Table in MySQL:



```
mysql> show tables;
+----------------------+
| Tables_in_BDSpark_pro |
+----------------------+
| max_min_volume       |
| pearson_corr         |
| walmart              |
+----------------------+
3 rows in set (0.01 sec)

mysql> select * from pearson_corr;
+-------------------------+
| pearson_corr_high_volume |
+-------------------------+
|       -0.338432606173703 |
+-------------------------+
1 row in set (0.00 sec)
```

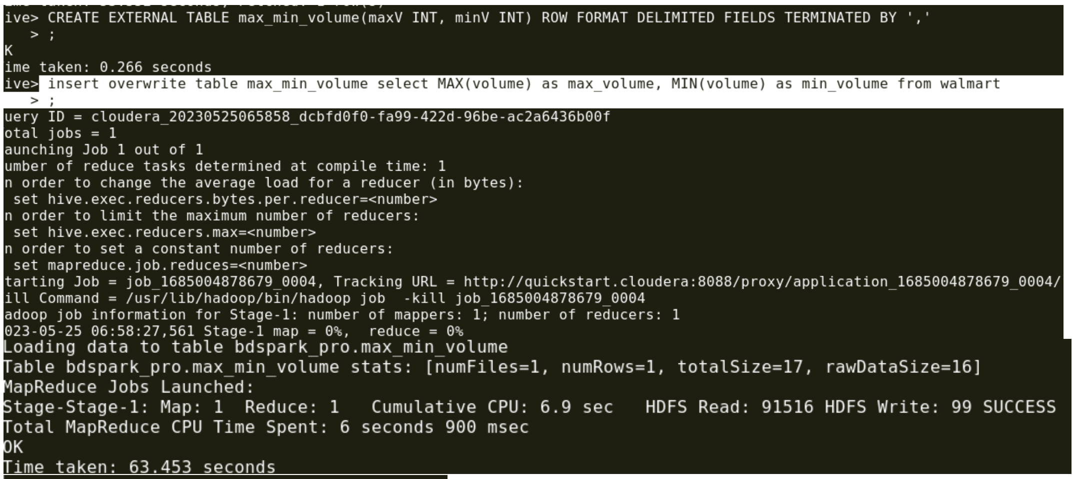
8. What is the max and min of the Volume column?

8.1-> Ext Table Creation:

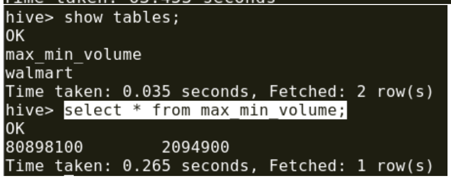
```
CREATE EXTERNAL TABLE max_min_volume(maxV INT, minV INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

8.2-> Transferring o/p to ext table:

```
insert overwrite table max_min_volume select MAX(volume) as max_volume, MIN(volume) as min_volume from walmart
;
```



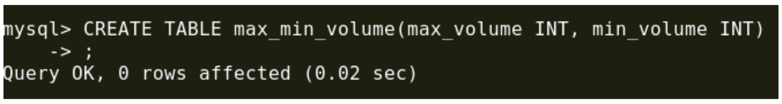
```
ive> CREATE EXTERNAL TABLE max_min_volume(maxV INT, minV INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
   > ;
K
ime taken: 0.266 seconds
ive> insert overwrite table max_min_volume select MAX(volume) as max_volume, MIN(volume) as min_volume from walmart
   > ;
uery ID = cloudera_20230525065858_dcbfd0f0-fa99-422d-96be-ac2a6436b00f
otal jobs = 1
aunching Job 1 out of 1
umber of reduce tasks determined at compile time: 1
n order to change the average load for a reducer (in bytes):
 set hive.exec.reducers.bytes.per.reducer=<number>
n order to limit the maximum number of reducers:
 set hive.exec.reducers.max=<number>
n order to set a constant number of reducers:
 set mapreduce.job.reduces=<number>
tarting Job = job_1685004878679_0004, Tracking URL = http://quickstart.cloudera:8088/proxy/application_1685004878679_0004/
ill Command = /usr/lib/hadoop/bin/hadoop job  -kill job_1685004878679_0004
adoop job information for Stage-1: number of mappers: 1; number of reducers: 1
023-05-25 06:58:27,561 Stage-1 map = 0%,  reduce = 0%
Loading data to table bdspark_pro.max_min_volume
Table bdspark_pro.max_min_volume stats: [numFiles=1, numRows=1, totalSize=17, rawDataSize=16]
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 6.9 sec   HDFS Read: 91516 HDFS Write: 99 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 900 msec
OK
Time taken: 63.453 seconds
```



```
hive> show tables;
OK
max_min_volume
walmart
Time taken: 0.035 seconds, Fetched: 2 row(s)
hive> select * from max_min_volume;
OK
80898100        2094900
Time taken: 0.265 seconds, Fetched: 1 row(s)
```

8.3-> MySQL Table Created (Client's DB):

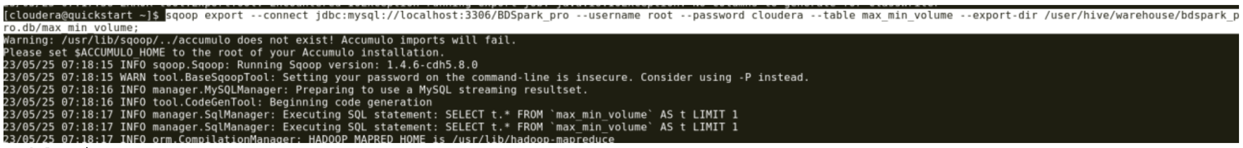
```
CREATE TABLE max_min_volume(max_volume INT, min_volume INT)
```



```
mysql> CREATE TABLE max_min_volume(max_volume INT, min_volume INT)
    -> ;
Query OK, 0 rows affected (0.02 sec)
```

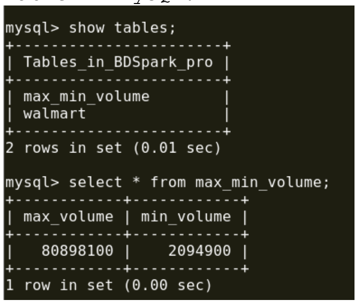
8.4-> Sqoop command to transfer o/p table Hive ----------> MySQL:

```
sqoop export --connect jdbc:mysql://localhost:3306/BDSpark_pro --username root --password cloudera --table
max_min_volume --export-dir /user/hive/warehouse/bdspark_pro.db/max_min_volume;
```



```
[cloudera@quickstart ~]$ sqoop export --connect jdbc:mysql://localhost:3306/BDSpark_pro --username root --password cloudera --table max_min_volume --export-dir /user/hive/warehouse/bdspark_p
ro.db/max_min_volume;
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
23/05/25 07:18:15 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.8.0
23/05/25 07:18:15 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
23/05/25 07:18:16 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
23/05/25 07:18:16 INFO tool.CodeGenTool: Beginning code generation
23/05/25 07:18:17 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `max_min_volume` AS t LIMIT 1
23/05/25 07:18:17 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `max_min_volume` AS t LIMIT 1
23/05/25 07:18:17 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/lib/hadoop-mapreduce
```

Table in MySQL:

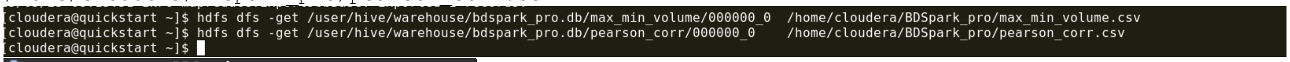


```
mysql> show tables;
+----------------------+
| Tables_in_BDSpark_pro |
+----------------------+
| max_min_volume       |
| walmart              |
+----------------------+
2 rows in set (0.01 sec)

mysql> select * from max_min_volume;
+------------+------------+
| max_volume | min_volume |
+------------+------------+
|   80898100 |    2094900 |
+------------+------------+
1 row in set (0.00 sec)
```

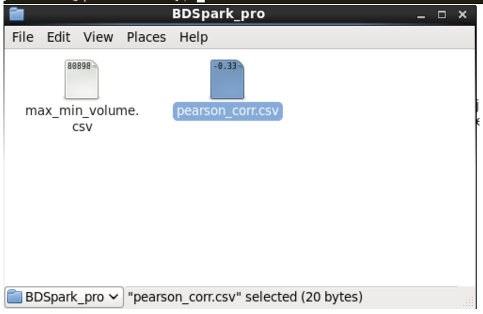
===

4. Getting o/p data files stored in HDFS -------------> Client machine(here, LFS):

```
hdfs dfs -get /user/hive/warehouse/bdspark_pro.db/max_min_volume/000000_0
/home/cloudera/BDSpark_pro/max_min_volume.csv
hdfs dfs -get /user/hive/warehouse/bdspark_pro.db/pearson_corr/000000_0
/home/cloudera/BDSpark_pro/pearson_corr.csv
```



```
[cloudera@quickstart ~]$ hdfs dfs -get /user/hive/warehouse/bdspark_pro.db/max_min_volume/000000_0  /home/cloudera/BDSpark_pro/max_min_volume.csv
[cloudera@quickstart ~]$ hdfs dfs -get /user/hive/warehouse/bdspark_pro.db/pearson_corr/000000_0    /home/cloudera/BDSpark_pro/pearson_corr.csv
[cloudera@quickstart ~]$
```



"pearson_corr.csv" selected (20 bytes)

THANK YOU!