1) **Avro Format: Compression**

Import com.databricks.spark.avro._ //Only for scala

- Scala Read an avro: sqlContext.read.avro(“/user/cloudera/avroFile”)
- Python Read an avro:
  sqlContext.read.format("com.databricks.spark.avro").load("filePath")

Supported compression: uncompressed, snappy, deflate (**a -> USD**)

```
sqlContext.setConf("spark.sql.avro.compression.codec","uncompressed")
sqlContext.setConf("spark.sql.avro.compression.codec","snappy")

sqlContext.setConf("spark.sql.avro.compression.codec","deflate")
        sqlContext.setConf(“spark.sql.avro.deflate.level”, “5”)
```

- Python/Scala Write an avro:
  df.write.format("com.databricks.spark.avro").save("output dir")

- To Verify output data:
  For avro format: 1) avro-tools getmeta fullPathOfAvroFileInHDFS
  //OR download the file and provide local file system path
  2) Hadoop fs -cat <file in hdfs> | head

Note: 1) Gzip is not supported. 2) Cloudera: No default compression defined for avro.
3) example full hdfs path for avro file :
hdfs://quickstart.cloudera:8020/practice/order_items_avro/part-r-00001-d1ab8124-ec3d-4673-a119-2c2f24f626c2.avro
4) Verify compression using: sqlContext.getConf(“spark.sql.avro.compression.codec”)

2) ORC Format: Compression

Supported compression: none, zlib, snappy (**o -> ZNS**)

- none, **zlib**, snappy

We have to use a create table approach if we need to have specific compression apart from ZLIB for ORC file format while writing.

- Scala/Python read an ORC: sqlContext.read.orc(“path to location”)

sqlContext.sql(“CREATE TABLE orders_orc_hive STORED AS orc LOCATION ‘/user/hive/warehouse/tableName’ TBLPROPERTIES(‘orc.compress’=**’SNAPPY’**) as SELECT * from order_orc”) //where order_orc is a table created via registerTempTable(“order_orc”)

- Scala write ORC table without compression:
  import org.apache.spark.sql.SaveMode
  productDF.write.mode(SaveMode.Overwrite).format(“orc”).saveAsTable(“prd_orc_table”)

- Scala Write an RDD to ORC format:
  dataRDD.toDF().write.format("orc").save("/path/to/save/file")

- To Verify output data:
  For ORC format: 1) hive --orcfiledump pathoforcfile
  //OR download the file and provide local file system path
  2) Hadoop fs -cat <file in hdfs> | head

Note: 1) default compression for ORC : zlib.
2) As of now, Compression only works with SparkSQL create query.

3) Parquet Format: Compression

- Read parquet in scala: val df = sqlContext.read.parquet("hdfspath/file/")
- Read parquet in python: df = spark.read.parquet("hdfspath/file/")

Supported compression: uncompressed, snappy, gzip (**p -> SGU**)

```
sqlContext.setConf("spark.sql.parquet.compression.codec", "uncompressed");
sqlContext.setConf("spark.sql.parquet.compression.codec", "snappy");
sqlContext.setConf("spark.sql.parquet.compression.codec", "gzip");
```

- Scala write a parquet file:
  resultDF.write.parquet(“output_file_location_in_hdfs”)
  resultDF.write.mode(SaveMode.Overwrite).parquet(“result_parquet_file”)

- Scala write a parquet table
  import org.apache.spark.sql.SaveMode
  resultDF.write.mode(SaveMode.Overwrite).format(“parquet”).saveAsTable(“parquet_tbl”)

- Python write a parquet file:
  resultDF.mode(“overwrite”).parquet(“output_file_location_in_hdfs”)
  resultDF.write.parquet(““output_file_location_in_hdfs”)

- To Verify output data:
  For Parquet format: 1) hadoop parquet.tools.Main meta pathToParquetFileInHDFS

2) Hadoop fs -cat <file in hdfs> | head

Note: 1) default compression for parquet: gzip.
2) Verify using: sqlContext.getConf("spark.sql.parquet.compression.codec")

4) Text file type:

- Python/Scala read text file:
  sc.textFile("/user/cloudera/Problem/file1.txt")

- Python/Scala write text file:
  resRDD.saveAsTextFile("/user/cloudera/Problem/resText")

- Scala write text file with compression:
  import org.apache.hadoop.io.compress.GzipCodec
  resRDD.saveAsTextFile("/user/cloudera/Problem/resTextCompressed",classOf[GzipCodec])

- Scala write dataframe to text file: (delim comma (,))
  resDF.rdd.map(x => x.mkString(",")).saveAsTextFile("/user/cloudera/Problem/DFtoText")

  Using Databricks package: (won't work without this package)
  spark-shell --packages com.databricks:spark-csv_2.10:1.4.0
  resDF.write.format("com.databricks.spark.csv").option("header", "true").save("filePath")

- Python write dataframe to text file: (delim comma (,))
jsonFile.rdd.map(lambda x: ",".join(map(str, x))).saveAsTextFile("/user/cloudera/DFtoText")

  Using Databricks package: (won't work without this package)
  pyspark --packages com.databricks:spark-csv_2.10:1.4.0
  resDF.write.format("com.databricks.spark.csv").option("header", "true").save("filePath")
  jsonFile.write.format('com.databricks.spark.csv').save("/user/cloudera/tempFile/Exp")

- To Verify output data:
  1) Hadoop fs -cat <file in hdfs> | head

5) Sequence file type (Only works with Pair RDD):

- Python/Scala read sequence file:
  Read: sc.sequenceFile("hadoopCca175/problem")

- Python write sequence file:

nonEmpty_lines.map(lambda line: (None, line)).saveAsSequenceFile(“Cca175/seqEx”, "org.apache.hadoop.io.compress.GzipCodec")

- Scala write sequence file:
  val v = sc.parallelize(Array(("owl",3), ("gnu",4), ("dog",1), ("cat",2), ("ant",5)), 2)
  v.saveAsSequenceFile("hd_seq_file", Some(classOf[org.apache.hadoop.io.compress.SnappyCodec])
  Or
  just use: classOf[org.apache.hadoop.io.compress.SnappyCodec]

- To Verify output data:
  1) Hadoop fs -cat <file in hdfs> | head

6) JSON file type:

- Python/Scala Read Json:-
  employeeDF = sqlContext.read.json(“employee.json”)

- Python Write a JSON:-
  employeeDF.toJSON().saveAsTextFile(“employee1”)

- Scala Write a JSON :-
  empoyeeDF.write.json("employee2")

  Scala write Json with Compression :-
  import org.apache.hadoop.io.compress.GzipCodec
  empoyeeDF.toJSON.saveAsTextFile("/tmp/jsonRecords", classOf[GzipCodec])

  Python write Json with Compression
  empJSON.toJSON().saveAsTextFile("/user/cloudera/jsonGzip",compressionCodecClass="org.apache.hadoop.io.compress.GzipCodec")

- To Verify output data:
  1) Hadoop fs -cat <file in hdfs> | head