# File Formats

|  |  |  |  |
| --- | --- | --- | --- |
| **File Format** | **Action** | **Procedure** | **Remarks** |
| Text File | Read | sc.textFile('ages.csv')  spark.read.text('sql/text-test.txt') |  |
| Write | codec = "org.apache.hadoop.io.compress.GzipCodec"  rdd.saveAsTextFile(tempFile3.name, codec)  OR  df.write.mode(“overwrite”).text(“path”) |  |
| CSV File | Read | spark.read.csv('python/ages.csv')  spark.read.format("csv").option("header","False").option("compression","none").option("delimiter",",").load("path") |  |
| Write | dfWrite.coalesce(numPartitions).write.mode("overwrite").format("csv").option("compression",compression).option("header",true).option("delimiter",",").option("quote","'").save(path) |  |
| JSON File | Read | spark.read.json("people.json")  spark.read.format('json').load('people.json') |  |
| Write | dfWrite.coalesce(numPartitions).write.partitionBy(partition).mode("overwrite").format("json").option("compression", compression).save("path") |  |
| SEQUENCE FILE | Read | spark.sparkContext.sequenceFile("./tester", "org.apache.hadoop.io.IntWritable", "org.apache.hadoop.io.Text").toDF(["col1","col2"]) | Cat the file to figure out key, value classes |
| Write | df.rdd.map(lambda rw: (key, value)).saveAsSequenceFile("Path",None)  Default compression is None  **Note : Else use full qualified class name of compression**  The codec full name can also be figured out from file core-site.xml at /etc/hadoop/conf | Pair RDDs can only be saved as Sequence File |
| PARQUET FILE | Read | spark.read.parquet("...") |  |
| Write | dfWrite.coalesce(numPartitions).write.partitionBy(partition).mode("overwrite").format("parquet").option("compression", compression).save("path") |  |
| ORC FILE | Read | spark.read.orc('sql/orc\_partitioned') |  |
| Write | dfWrite.coalesce(numPartitions).write.partitionBy(partition).mode("overwrite").format("orc").option("compression", compression).save("path") |  |
| AVRO FILE | Read | spark.read.format("avro").load("/tmp/episodes.avro") |  |
| Write | import com.databricks.spark.avro.\_; |  |
| Hive Table | Read | spark.read.table('tmpTable')  spark.sql('SELECT \* from tmpTable’) |  |
| Write | dataFrame.write.mode(SaveMode.Append).partitionBy("entity","date").format("orc").saveAsTable("baseTable") |  |
| JDBC | Read | spark.read.format("jdbc").option("url", "jdbd://").option("driver","org.mysql").option("dbtable", "table").option("user", "root").option("password", "cloudera").load() |  |
| Write | dfWrite.coalesce(numPartitions).write.partitionBy(partition).mode("overwrite").format("jdbc").option("url","jdbdc:://" ).option("driver", "org.mysql.").option("user", "root").option("password", "password").option("dbtable", "table").save() |  |

<https://spark.apache.org/docs/latest/sql-data-sources.html>

**Reading custom format Text file:**

spark.read.format('csv')\

              .option("sep","\001")\

              .load("./resources/hiveFile")

**Reading multiline custom file separator:**

spark.read.option("lineSep", '\nid,').text('/path/to/retrosheet/file')

<https://stackoverflow.com/questions/31227363/creating-spark-data-structure-from-multiline-record>