**Structured Data**

* Extends RDD to a “DataFrame” obj
* DataFrame
  + Row objects
  + Can run SQL queries
  + Has a schema (more efficient storage)
    - We know the type of data
      * Network transfers fasters
      * Optimizations
  + Read and write to JSON, Hive, parquet
  + Communicates with JDBC/ODBC, Tableau

**SparkSQL in Python**

* From pyspark.sql import SQLContext, Row
* hiveContext = HiveContext(sc
* inputData = spark.read.json(dataFile)
* inputData.createOrReplaceTempView(“myStructuredStuff”)
* myResultDataFrame = hiveContext.sql(“””SELECT foo FROM bar ORDER BY foobar”””)

**Programmatic manipulations**

* myDataFrom.show()
* myDataFrom.select(“someField”)
* myDataFrom.filter(myDataFrom(“someField” > 200))
* myDataFrom.mean()
* myDataFrom.groupBy(myDataFrom(“someField”))
* myDataFrom.rdd().map(someFunction)
  + since Data Frames extend RDD’s we can extract underlying RDD and perform operations at the RDD level (lower level)

**Datasets**

* DataFrame is a DataSEt of Row obj
* Can wrap known, typed data
* Unified API between different Spark subsystems
  + Spark Streaming
  + MLlib -> ML
* Can pass between subsystems i.e. streaming -> MLlib

**Shell Access**

* Spark SQL exposes JDBC/ODBC server (if Spark with Hive support)
* sbin/start-thriftserver.sh
* Listens on port 10000 by default
* bin/beeline –u jdbc:hive2://localhost:10000
* Create new tables
  + hiveCtx.cacheTable(“tableName”)

**UDF’s**

* from pyspark.sql.types import IntegerType
* hiveCtx.registerFunction(“square”, lambda x: x\*x, IntegerType())
  + Register a new UDF
* df = hiveCtx.sql(“SELECT square(‘someField’) FROM tableName)