PySpark 3.0 Import/Export Quick Guide

Reading and Writing Data Using PySpark

PySpark supports a rich set of input/output data sources including:

Files sources

- Comma Separated Values(CSV) /other separators (Tab,|, etc.)
- Text (and fixed width)
- JSON
- XML
- MS Excel Files (xlsx)
- SAS Datasets (sas7bdat)
- COBOL Copybook Data
- Apache Parquet
- Apache Avro
- Apache ORC
- Images
- Binary Files

Logical Data Stores

- Apache Kafka
- Table (defined in Spark Metastore)
- Delta Lake

RDBMS Sources – Database Tables (batch only)

Any JDBC supported data sources (MySQL/MariaDB, Posgr-eSQL, Oracle, SQLServer, Sybase, Teradata, Hive, DB2,Netezza)

Streaming specific input sources (read)

- Socket source
- Rate source (testing)

Streaming specific output sinks (write)

- Foreach Sink
- ForeachBatch Sink
- Console Sink
- Memory Sink

Dataframe Reader Overview (spark.read batch API)

```
• option(key, value) # specify 1 data source option
• options(**options) # specify multiple options
• format()           # define format of data source
• schema()           # define data schema
• load()             # action loads source file
```

Dataframe Writer Overview (spark.write batch API)

```
• bucketBy(buckets, col, *cols) # buckets output
• sortBy(col, *cols) # Sorts output in each bucket
• partitionBy(*cols) # partitions by columns
• option(key, value) # specify 1 data source option
• options(**options) # multiple data source options
• format()           # define the output format
• mode()             # (append, overwrite, error, ignore)
• save()             # action saves DataFrame to file
• saveAsTable()      # action saves DataFrame to table
• insertInto()       # action inserts into table
```

Structured Streaming Source Overview

```
• option(key, value) # specify 1 data source option
• options(**options) # specify multiple options
• format()           # define the format of the source
• schema()           # define data schema
• load()             # action loads source file
```

Structured Streaming Sink Overview

```
• foreach(f)         # process output using writer f
• foreachBatch(func) # process streaming output by func
• option(key, value) # specify 1 sink option
• options(**options) # specify multiple options
• format()           # define the format of the sink
• outputMode()       # mode(append,complete,update)
• partitionBy(*cols) # partitions output by columns
• queryName(name)    # sets StreamingQuery name
• start()            # action starts the streaming query
• trigger()          # sets the trigger of StreamingQuery
```

Re-partitioning For Single File Output

PySpark is a cluster architecture, many file formats create multiple files by default for read/write performance. To create a single output file, use **`.repartition(1)`** before the **`.write`** method call.

Reading CSV / Other Separated Values see docs for all options

Long form

```
df = (spark.read
    .format('csv') # specify csv reader
    .option('inferSchema', True) # optional True/False
    .option('header', True) # optional True/False
    .option('sep', ',') # optional separator, default is ','
    .load('/path/to/acme.csv')) # load file
```

Short (implicit) form

```
df = spark.read.csv('/path/to/acme.csv', inferSchema=True,
    header=True, sep=',')
```

Writing CSV / Other Separated Values see docs for all options

Long form

```
(df.write
    .format('csv') # specify csv writer
    .mode('overwrite') # overwrites existing file if exists
    .option('header', True) # add a header to the file
    .option('sep', True) # optional separator, default ','
    .save('/path/to/acme.csv')) # save to file
```

Short (implicit) form

```
df.write.csv('/path/to/acme.csv', mode='overwrite',
    header=True, sep=',')
```

Reading Text (and fixed width)

Long form

```
df = (spark.read
    .format('text') # specify text reader
    .load('/path/to/acme.dat')) # load file
```

Short (implicit) form

```
df = spark.read.text('/path/to/acme.dat')
```

Fixed Width

```
# Use .withColumn() & substring() to parse desired columns:
df = (spark.read.text('/path/to/acme.dat')
    .withColumn('org_name', substring(col('value'), 1, 10))
    .withColumn('org_ind', substring(col('value'), 11, 10))
    .drop('value'))
```

Writing Text (and fixed width)

Long form

```
df = (spark.write
    .format('text') # specify text writer
    .option('mode', 'overwrite') # overwrites existing file
    .load('/path/to/acme.dat')) # load text file
```

Short (implicit) form

```
df = spark.write.text('/path/to/acme.dat', mode='overwrite')
```

Fixed Width

```
# Use .withColumn() & concat() to create a single string
column:
(df.withColumn('string_column', concat(
        rpad(col('org_name'), 10, ' '),
        rpad(col('org_ind'), 10, ' ')))
    .select('string_column')
    .write.text('/path/to/acme.dat', mode='overwrite'))
```

Reading JSON Data - see docs for all options

Long form

```
df = (spark.read
    .format('json')
    .load('/path/to/acme.json'))
```

Short (implicit) form

```
df = spark.read.json('/path/to/acme.json')
```

Writing JSON Data - see docs for all options

Long form

```
(df.write
    .format('json')
    .mode('overwrite') # overwrites existing file if exists
    .save('/path/to/acme.json'))
```

Short (implicit) form

```
df.write.json('/path/to/acme.json')
```

Reading XML Data (using spark-xml plugin) see docs for all options

```
df = (spark.read
    .format('xml') # specify XML reader
    .options(rowTag='company') # specify row tag
    .load('/path/to/acme.xml')) # action to load file
```

Writing XML Data (using spark-xml plugin) see docs for all options

```
(df.write
    .format('xml') # specify XML writer
    .mode('overwrite') # overwrites existing file if exists
    .options(rowTag='company', rootTag='companies')
    .save('/path/to/acme.xml')) # action to save file
```

Read Excel Data (using spark-excel plugin) see docs for all options

```
df = (spark.read
    .format('com.crealytics.spark.excel') # set excel reader
    .option('header', True) # required
    .option('dataAddress', 'A1:Z256') # optional, 'A1'
    .option('treatEmptyValuesAsNulls', True) # optional bool
    .option('timestampFormat', 'yyyy-mm-dd hh:mm:ss')
    .option('inferSchema', True )# optional True/False
    .option('excerptSize', 10) # inferSchema num of rows, 10
    .schema(customSchema) # optional
    .load('/path/to/acme.xlsx')) # action to load file
```

Writing Excel Data (using spark-excel plugin) see docs for all options

```
(df.write
    .format('com.crealytics.spark.excel')
    .option('dataAddress', "'AcmeStats'!B3:C35")
    .option('header', True)
    .option('dateFormat', 'yy-m-d') # optional
    .option('timestampFormat', 'yyyy-mm-dd hh:mm:ss.000')
    .mode('overwrite') # optional, default: overwrite.
    .save('/path/to/acme.xlsx')) # action to save file
```

Read ing SAS Datasets (using spark-sas7bdat plugin) see docs for all options

```
df = (spark.read
    .format('com.github.saurfang.sas.spark')
    .option('forceLowercaseNames', True)
    .option('inferLong', True) # cols w/o precision to long
    .load('/path/to/acme.sas7bdat')) # action load dataset
```

Read ing COBOL Copybook Data (using Cobrix plugin) see docs for all options

```
df = (spark.read
  .format('cobol')
  .option('copybook', '/path/to/copybook.cob')
  .load('/path/to/acmedata')) # action to load file
```

Reading Apache Parquet Data - see docs for all options

Long form

```
df = (spark.read
    .format('parquet') # specify Parquet reader
    .load('/path/to/acme.parquet')) # action to load file
```

Short (implicit) form

```
df = spark.read.parquet('/path/to/acme.parquet')
```

Writing Apache Parquet Data - see docs for all options

Long form

```
(df.write
    .format('parquet') # specify Parquet writer
    .mode('overwrite') # overwrites existing file if exists
    .save('/path/to/acme.parquet')) # action to save file
```

Short (implicit) form

```
df.write.csv('/path/to/acme.parquet ', mode='overwrite')
```

Reading Apache Avro Data

```
df = (spark.read
    .format('avro') # specify Avro reader
    .load('/path/to/acme.avro')) # action to load file
```

Writing Apache Avro Data

```
(df.write
    .format('avro') # specify Avro writer
    .mode('overwrite') # overwrites existing file if exists
    .save('/path/to/acme.avro')) # action to save to file
```

Reading Apache ORC Data - See docs for all options

Long form

```
df = (spark.read
    .format('orc') # specify ORC reader
    .load('/path/to/acme.orc')) # action to load file
```

Short (implicit) form

```
df = spark.read.orc('/path/to/acme.orc')
```

Writing Apache ORC Data - See docs for all options

Long form

```
(df.write
    .format('orc') # specify ORC writer
    .mode('overwrite') # overwrites existing file if exists
    .save('/path/to/acme.orc')) # action to save file
```

Short (implicit) form

```
df.write.orc('/path/to/acme.orc', mode='overwrite')
```

Reading From Image Files - See docs for all options

```
df = (spark.read
    .format('image')
    .option('dropInvalid', True)
    .load('/path/to/images'))
```

Reading From Binary Files - see docs for all options

```
df = (spark.read
    .format('binaryFile')
    .option('pathGlobFilter', '*.png')
    .load('/path/to/data'))
```

Reading From Apache Kafka - see docs for all options

```
df = (spark.read
    .format('kafka') # specify kafka reader
    .option('kafka.bootstrap.servers',
        'host1:port1,host2:port2')  # specify servers
  .option('subscribe', 'topic1') # subscribe to a topic
  .load()) # action to load data from Kafka
```

Writing To Apache Kafka - see docs for all options

```
(df.select(key, value).write # only supports k/v char cols
    .format('kafka') # specify kafka writer
    .option('kafka.bootstrap.servers',
        'host1:port1,host2:port2') # specify servers
    .option('topic', 'topic1') # write to a topic
    .save()) # action to save data to Kafka
```

Reading From Spark Table

```
df = spark.read.table('acme') # action to read from table
```

Writing To Spark Table (Parquet formatted)

```
(df.write
    .format('parquet') # specify Parquet writer
    .mode('overwrite') # overwrites existing file if exists
    .saveAsTable('acme')) # action to save to table
```

Reading From Delta Table (Delta plugin) See docs for all options

```
df = (spark.read
    .format('delta') # specify Delta reader
    .load('/path/to/acme')) # read from Delta table
```

Writing To Delta Table (Delta plugin) See docs for all options

```
(df.write
    .format('delta') # specify Delta writer
    .mode('overwrite') # overwrites existing file if exists
    .save('/path/to/acme')) # action to write to Delta table
```

Reading From JDBC (e.g. Oracle) - see docs for all options

```
df = (spark.read
    .format('jdbc') # specify JDBC reader
    .option('driver', 'oracle.jdbc.driver.OracleDriver')
    .options(user='usename',password='password')
    .option('url','jdbc:oracle:thin:host:port:SID')
    .option('dbtable', 'acme') # table or query (sub-table)
    .load()) # action to read from table
```

Writing To JDBC (e.g. Oracle) - see docs for all options

```
(df.write
    .format('jdbc') # specify Delta writer
    .mode('append') # append data to existing table
    .options(user='usename',password='password')
    .option('url','jdbc:oracle:thin:host:port:SID')
    .option('dbtable', 'acme') # table or query (sub-table)
    .save()) # action to write to table
```