sparkly Documentation

Release 2.8.2

Tubular

Jun 12, 2020

Sparkly is a library that makes usage of pyspark more convenient and consistent.

A brief tour on Sparkly features:

```
# The main entry point is SparklySession,
# you can think of it as of a combination of SparkSession and SparkSession.builder.
from sparkly import SparklySession


# Define dependencies in the code instead of messing with `spark-submit`.
class MySession(SparklySession):
    # Spark packages and dependencies from Maven.
    packages = [
        'datastax:spark-cassandra-connector:2.0.0-M2-s_2.11',
        'mysql:mysql-connector-java:5.1.39',
    ]

    # Jars and Hive UDFs
    jars = ['/path/to/brickhouse-0.7.1.jar'],
    udfs = {
        'collect_max': 'brickhouse.udf.collect.CollectMaxUDAF',
    }


spark = MySession()

# Operate with interchangeable URL-like data source definitions:
df = spark.read_ext.by_url('mysql://<my-sql.host>/my_database/my_database')
df.write_ext('parquet:s3://<my-bucket>/<path>/data?partition_by=<field_name1>')

# Interact with Hive Metastore via convenient python api,
# instead of verbose SQL queries:
spark.catalog_ext.has_table('my_custom_table')
spark.catalog_ext.get_table_properties('my_custom_table')

# Easy integration testing with Fixtures and base test classes.
from pyspark.sql import types as T
from sparkly.testing import SparklyTest


class TestMyShinySparkScript(SparklyTest):
    session = MySession

    fixtures = [
        MysqlFixture('<my-testing-host>', '<test-user>', '<test-pass>', '/path/to/
↪data.sql', '/path/to/clear.sql')
    ]

   def test_job_works_with_mysql(self):
       df = self.spark.read_ext.by_url('mysql://<my-testing-host>/<test-db>/<test-
↪table>?user=<test-usre>&password=<test-password>')
       res_df = my_shiny_script(df)
       self.assertRowsEqual(
          res_df.collect(),
          [T.Row(fieldA='DataA', fieldB='DataB', fieldC='DataC')],
       )
```

Sparkly Session

`SparklySession` is the main entry point to sparkly's functionality. It's derived from `SparkSession` to provide additional features on top of the default session. The are two main differences between `SparkSession` and `SparklySession`:

1. `SparklySession` doesn't have `builder` attribute, because we prefer declarative session definition over imperative.
2. Hive support is enabled by default.

The example below shows both imperative and declarative approaches:

```
# PySpark-style (imperative)
from pyspark import SparkSession

spark = SparkSession.builder\
    .appName('My App')\
    .master('spark://')\
    .config('spark.sql.shuffle.partitions', 10)\
    .getOrCreate()

# Sparkly-style (declarative)
from sparkly import SparklySession

class MySession(SparklySession):
    options = {
        'spark.app.name': 'My App',
        'spark.master': 'spark://',
        'spark.sql.shuffle.partitions': 10,
    }

spark = MySession()

# In case you want to change default options
spark = MySession({'spark.app.name': 'My Awesome App'})

# In case you want to access the session singleton
spark = MySession.get_or_create()
```

1.1 Installing dependencies

Why: Spark forces you to specify dependencies (spark packages or maven artifacts) when a spark job is submitted (something like `spark-submit --packages=...`). We prefer a code-first approach where dependencies are actually declared as part of the job.

For example: You want to read data from Cassandra.

```
from sparkly import SparklySession


class MySession(SparklySession):
    # Define a list of spark packages or maven artifacts.
    packages = [
        'datastax:spark-cassandra-connector:2.0.0-M2-s_2.11',
    ]

# Dependencies will be fetched during the session initialisation.
spark = MySession()

# Here is how you now can access a dataset in Cassandra.
df = spark.read_ext.by_url('cassandra://<cassandra-host>/<db>/<table>?
↪consistency=QUORUM')
```

1.2 Custom Maven repositories

Why: If you have a private maven repository, this is how to point spark to it when it performs a package lookup. Order in which dependencies will be resolved is next:

- Local cache
- Custom maven repositories (if specified)
- Maven Central

For example: Let's assume your maven repository is available on: http://my.repo.net/maven, and there is some spark package published there, with identifier: *my.corp:spark-handy-util:0.0.1* You can install it to a spark session like this:

```
from sparkly import SparklySession

class MySession(SparklySession):
    repositories = ['http://my.repo.net/maven']
    packages = ['my.corp:spark-handy-util:0.0.1']

spark = MySession()
```

1.3 Tuning options

Why: You want to customise your spark session.

For example:

- `spark.sql.shuffle.partitions` to tune shuffling;
- `hive.metastore.uris` to connect to your own HiveMetastore;

- `spark.hadoop.avro.mapred.ignore.inputs.without.extension` package specific options.

```
from sparkly import SparklySession


class MySession(SparklySession):
    options = {
        # Increase the default amount of partitions for shuffling.
        'spark.sql.shuffle.partitions': 1000,
        # Setup remote Hive Metastore.
        'hive.metastore.uris': 'thrift://<host1>:9083,thrift://<host2>:9083',
        # Ignore files without `avro` extensions.
        'spark.hadoop.avro.mapred.ignore.inputs.without.extension': 'false',
    }

# You can also overwrite or add some options at initialisation time.
spark = MySession({'spark.sql.shuffle.partitions': 10})
```

1.4 Tuning options through shell environment

Why: You want to customize your spark session in a way that depends on the hardware specifications of your worker (or driver) machine(s), so you'd rather define them close to where the actual machine specs are requested / defined. Or you just want to test some new configuration without having to change your code. In both cases, you can do so by using the environmental variable `PYSPARK_SUBMIT_ARGS`. Note that any options defined this way will override any conflicting options from your Python code.

For example:

- `spark.executor.cores` to tune the cores used by each executor;
- `spark.executor.memory` to tune the memory available to each executor.

```
PYSPARK_SUBMIT_ARGS='--conf "spark.executor.cores=32" --conf "spark.executor.
↪memory=160g"' \
    ./my_spark_app.py
```

1.5 Using UDFs

Why: To start using Java UDF you have to import JAR file via SQL query like `add jar ../path/to/file` and then call `registerJavaFunction`. We think it's too many actions for such simple functionality.

For example: You want to import UDFs from brickhouse library.

```
from pyspark.sql.types import IntegerType
from sparkly import SparklySession


def my_own_udf(item):
    return len(item)


class MySession(SparklySession):
    # Import local jar files.
    jars = [
```

```
        '/path/to/brickhouse.jar'
    ]
    # Define UDFs.
    udfs = {
        'collect_max': 'brickhouse.udf.collect.CollectMaxUDAF',  # Java UDF.
        'my_udf': (my_own_udf, IntegerType()),  # Python UDF.
    }

spark = MySession()

spark.sql('SELECT collect_max(amount) FROM my_data GROUP BY ...')
spark.sql('SELECT my_udf(amount) FROM my_data')
```

1.6 Lazy access / initialization

Why: A lot of times you might need access to the sparkly session at a low-level, deeply nested function in your code. A first approach is to declare a global sparkly session instance that you access explicitly, but this usually makes testing painful because of unexpected importing side effects. A second approach is to pass the session instance explicitly as a function argument, but this makes the code ugly since you then need to propagate that argument all the way up to every caller of that function.

Other times you might want to be able to glue together and run one after the other different code segments, where each segment initializes its own sparkly session, despite the sessions being identical. This situation could occur when you are doing investigative work in a notebook.

In both cases, `SparklySession.get_or_create` is the answer, as it solves the problems mentioned above while keeping your code clean and tidy.

For example: You want to use a read function within a transformation.

```
from sparkly import SparklySession


class MySession(SparklySession):
    pass

def my_awesome_transformation():
    df = read_dataset('parquet:s3://path/to/my/data')
    df2 = read_dataset('parquet:s3://path/to/my/other/data')
    # do something with df and df2...

def read_dataset(url):
    spark = MySession.get_or_create()
    return spark.read_ext.by_url(url)
```

1.7 API documentation

CHAPTER 2

Read/write utilities for DataFrames

Sparkly isn't trying to replace any of existing storage connectors. The goal is to provide a simplified and consistent api across a wide array of storage connectors. We also added the way to work with *abstract data sources*, so you can keep your code agnostic to the storages you use.

2.1 Cassandra

Sparkly relies on the official spark cassandra connector and was successfully tested in production using version *2.4.0*.

For using overwrite mode, it is needed to specify confirm.truncate as true. Otherwise, use append mode to update existing data.

```
from sparkly import SparklySession


class MySession(SparklySession):
    # Feel free to play with other versions
    packages = ['datastax:spark-cassandra-connector:2.4.0-s_2.11']

spark = MySession()

# To read data
df = spark.read_ext.cassandra('localhost', 'my_keyspace', 'my_table')
# To write data
df.write_ext.cassandra('localhost', 'my_keyspace', 'my_table')
```

2.2 Elastic

Sparkly relies on the official elastic spark connector and was successfully tested in production using version *6.5.4*.

Package	https://spark-packages.org/package/elastic/elasticsearch-hadoop
Configuration	https://www.elastic.co/guide/en/elasticsearch/hadoop/5.1/configuration.html

```
from sparkly import SparklySession


class MySession(SparklySession):
    # Feel free to play with other versions
    packages = ['org.elasticsearch:elasticsearch-spark-20_2.11:6.5.4']
```

```
spark = MySession()

# To read data
df = spark.read_ext.elastic('localhost', 'my_index', 'my_type', query='?q=awesomeness
↪')
# To write data
df.write_ext.elastic('localhost', 'my_index', 'my_type')
```

2.3 Kafka

Sparkly's reader and writer for Kafka are built on top of the official spark package for Kafka and python library kafka-python . The first one allows us to read data efficiently, the second covers a lack of writing functionality in the official distribution.

Package	https://mvnrepository.com/artifact/org.apache.spark/spark-streaming-kafka-0-8_2.11/2.4.0
Configuration	http://spark.apache.org/docs/2.4.0/streaming-kafka-0-8-integration.html

Note:

- To interact with Kafka, `sparkly` needs the `kafka-python` library. You can get it via: ` pip install sparkly[kafka] `
- Sparkly was tested in production using Apache Kafka **0.10.x**.

```
import json

from sparkly import SparklySession


class MySession(SparklySession):
    packages = [
        'org.apache.spark:spark-streaming-kafka-0-8_2.11:2.4.0',
    ]

spark = MySession()

# To read JSON messaged from Kafka into a dataframe:

#   1. Define a schema of the messages you read.
df_schema = StructType([
    StructField('key', StructType([
        StructField('id', StringType(), True)
    ])),
    StructField('value', StructType([
        StructField('name', StringType(), True),
        StructField('surname', StringType(), True),
    ]))
])

#   2. Specify the schema as a reader parameter.
df = hc.read_ext.kafka(
    'kafka.host',
    topic='my.topic',
    key_deserializer=lambda item: json.loads(item.decode('utf-8')),
```

```
    value_deserializer=lambda item: json.loads(item.decode('utf-8')),
    schema=df_schema,
)

# To write a dataframe to Kafka in JSON format:
df.write_ext.kafka(
    'kafka.host',
    topic='my.topic',
    key_serializer=lambda item: json.dumps(item).encode('utf-8'),
    value_serializer=lambda item: json.dumps(item).encode('utf-8'),
)
```

2.4 MySQL

Basically, it's just a high level api on top of the native jdbc reader and jdbc writer.

Jars	https://mvnrepository.com/artifact/mysql/mysql-connector-java
Configuration	https://dev.mysql.com/doc/connector-j/5.1/en/connector-j-reference-configuration-properties.html

```
from sparkly import SparklySession
from sparkly.utils import absolute_path


class MySession(SparklySession):
    # Feel free to play with other versions.
    packages = ['mysql:mysql-connector-java:6.0.6']


spark = MySession()

# To read data
df = spark.read_ext.mysql('localhost', 'my_database', 'my_table',
                          options={'user': 'root', 'password': 'root'})
# To write data
df.write_ext.mysql('localhost', 'my_database', 'my_table', options={
    'user': 'root',
    'password': 'root',
    'rewriteBatchedStatements': 'true',  # improves write throughput dramatically
})
```

2.5 Redis

Sparkly provides a writer for Redis that is built on top of the official redis python library redis-py . It is currently capable of exporting your DataFrame as a JSON blob per row or group of rows.

Note:

- To interact with Redis, `sparkly` needs the `redis` library. You can get it via: `pip install sparkly[redis]`

```
import json

from sparkly import SparklySession


spark = SparklySession()

# Write JSON.gz data indexed by col1.col2 that will expire in a day
df.write_ext.redis(
    host='localhost',
    port=6379,
    key_by=['col1', 'col2'],
    exclude_key_columns=True,
    expire=24 * 60 * 60,
    compression='gzip',
)
```

2.6 Universal reader/writer

The *DataFrame* abstraction is really powerful when it comes to transformations. You can shape your data from various storages using exactly the same api. For instance, you can join data from Cassandra with data from Elasticsearch and write the result to MySQL.

The only problem - you have to explicitly define sources (or destinations) in order to create (or export) a *DataFrame*. But the source/destination of data doesn't really change the logic of transformations (if the schema is preserved). To solve the problem, we decided to add the universal api to read/write *DataFrames*:

```
from sparkly import SparklyContext

class MyContext(SparklyContext):
    packages = [
        'datastax:spark-cassandra-connector:1.6.1-s_2.10',
        'com.databricks:spark-csv_2.10:1.4.0',
        'org.elasticsearch:elasticsearch-spark_2.10:6.5.4',
    ]

hc = MyContext()

# To read data
df = hc.read_ext.by_url('cassandra://localhost/my_keyspace/my_table?consistency=ONE')
df = hc.read_ext.by_url('csv:s3://my-bucket/my-data?header=true')
df = hc.read_ext.by_url('elastic://localhost/my_index/my_type?q=awesomeness')
df = hc.read_ext.by_url('parquet:hdfs://my.name.node/path/on/hdfs')

# To write data
df.write_ext.by_url('cassandra://localhost/my_keyspace/my_table?consistency=QUORUM&
↪parallelism=8')
df.write_ext.by_url('csv:hdfs://my.name.node/path/on/hdfs')
df.write_ext.by_url('elastic://localhost/my_index/my_type?parallelism=4')
df.write_ext.by_url('parquet:s3://my-bucket/my-data?header=false')
```

2.7 Controlling the load

From the official documentation:

> Don't create too many partitions in parallel on a large cluster; otherwise Spark might crash your external database systems.
>
> link: <https://spark.apache.org/docs/2.0.1/api/java/org/apache/spark/sql/DataFrameReader.html>

It's a very good advice, but in practice it's hard to track the number of partitions. For instance, if you write a result of a join operation to database the number of splits might be changed implicitly via *spark.sql.shuffle.partitions*.

To prevent us from shooting to the foot, we decided to add *parallelism* option for all our readers and writers. The option is designed to control a load on a source we write to / read from. It's especially useful when you are working with data storages like Cassandra, MySQL or Elastic. However, the implementation of the throttling has some drawbacks and you should be aware of them.

The way we implemented it is pretty simple: we use *coalesce* on a dataframe to reduce an amount of tasks that will be executed in parallel. Let's say you have a dataframe with 1000 splits and you want to write no more than 10 task in parallel. In such case *coalesce* will create a dataframe that has 10 splits with 100 original tasks in each. An outcome of this: if any of these 100 tasks fails, we have to retry the whole pack in 100 tasks.

Read more about coalesce

2.8 Reader API documentation

2.9 Writer API documentation

Hive Metastore Utils

3.1 About Hive Metastore

The Hive Metastore is a database with metadata for Hive tables.

To configure `SparklySession` to work with external Hive Metastore, you need to set `hive.metastore.uris` option. You can do this via `hive-site.xml` file in spark config ($SPARK_HOME/conf/hive-site.xml):

```
<property>
  <name>hive.metastore.uris</name>
  <value>thrift://<n.n.n.n>:9083</value>
  <description>IP address (or fully-qualified domain name) and port of the metastore host</description>
</property>
```

or set it dynamically via `SparklySession` options:

```
class MySession(SparklySession):
    options = {
        'hive.metastore.uris': 'thrift://<n.n.n.n>:9083',
    }
```

3.2 Tables management

Why: you need to check if tables exist, rename them, drop them, or even overwrite existing aliases in your catalog.

```
from sparkly import SparklySession


spark = SparklySession()

assert spark.catalog_ext.has_table('my_table') in {True, False}
spark.catalog_ext.rename_table('my_table', 'my_new_table')
spark.catalog_ext.create_table('my_new_table', path='s3://my/parquet/data', source='parquet', mode='overwrite')
spark.catalog_ext.drop_table('my_new_table')
```

3.3 Table properties management

Why: sometimes you want to assign custom attributes for your table, e.g. creation time, last update, purpose, data source. The only way to interact with table properties in spark - use raw SQL queries. We implemented a more convenient interface to make your code cleaner.

```
from sparkly import SparklySession


spark = SparklySession()
spark.catalog_ext.set_table_property('my_table', 'foo', 'bar')
assert spark.catalog_ext.get_table_property('my_table', 'foo') == 'bar'
assert spark.catalog_ext.get_table_properties('my_table') == {'foo': 'bar'}
```

Note properties are stored as strings. In case if you need other types, consider using a serialisation format, e.g. JSON.

3.4 Using non-default database

Why to split your warehouse into logical groups (for example by system components). In all catalog_ext.* methods you can specify full table names <db-name>.<table-name> and it should operate properly

```
from time import time
from sparkly import SparklySession

spark = SparklySession()

if spark.catalog_ext.has_database('my_database'):
    self.catalog_ext.rename_table(
        'my_database.my_badly_named_table',
        'new_shiny_name',
    )
    self.catalog_ext.set_table_property(
        'my_database.new_shiny_name',
        'last_update_at',
        time(),
    )
```

Note be careful using 'USE' statements like: spark.sql('USE my_database'), it's stateful and may lead to weird errors, if code assumes correct current database.

3.5 API documentation

CHAPTER 4

Testing Utils

4.1 Base TestCases

There are two main test cases available in Sparkly:

- `SparklyTest` creates a new session for each test case.
- `SparklyGlobalSessionTest` uses a single sparkly session for all test cases to boost performance.

```
from pyspark.sql import types as T

from sparkly import SparklySession
from sparkly.testing import SparklyTest, SparklyGlobalSessionTest


class MyTestCase(SparklyTest):
    session = SparklySession

    def test(self):
        df = self.spark.read_ext.by_url(...)

        # Compare all fields
        self.assertRowsEqual(
            df.collect(),
            [
                T.Row(col1='row1', col2=1),
                T.Row(col1='row2', col2=2),
            ],
        )

...

class MyTestWithReusableSession(SparklyGlobalSessionTest):
    context = SparklySession

    def test(self):
        df = self.spark.read_ext.by_url(...)

...
```

4.2 DataFrame Assertions

Asserting that the dataframe produced by your transformation is equal to some expected output can be unnecessarily complicated at times. Common issues include:

- Ignoring the order in which elements appear in an array. This could be particularly useful when that array is generated as part of a `groupBy` aggregation, and you only care about all elements being part of the end result, rather than the order in which Spark encountered them.
- Comparing floats that could be arbitrarily nested in complicated datatypes within a given tolerance; exact matching is either fragile or impossible.
- Ignoring whether a field of a complex datatype is nullable. Spark infers this based on the applied transformations, but it is oftentimes inaccurate. As a result, assertions on complex data types might fail, even though in theory they shouldn't have.
- Having rows with different field names compare equal if the values match in alphabetical order of the names (see unit tests for example).
- Unhelpful diffs in case of mismatches.

Sparkly addresses these issues by providing `assertRowsEqual`:

```
from pyspark.sql import types as T

from sparkly import SparklySession
from sparkly.test import SparklyTest


def my_transformation(spark):
    return spark.createDataFrame(
        data=[
            ('row1', {'field': 'value_1'}, [1.1, 2.2, 3.3]),
            ('row2', {'field': 'value_2'}, [4.1, 5.2, 6.3]),
        ],
        schema=T.StructType([
            T.StructField('id', T.StringType()),
            T.StructField(
                'st',
                T.StructType([
                    T.StructField('field', T.StringType()),
                ]),
            ),
            T.StructField('ar', T.ArrayType(T.FloatType())),
        ]),
    )


class MyTestCase(SparklyTest):
    session = SparklySession

    def test(self):
        df = my_transformation(self.spark)

        self.assertRowsEqual(
            df.collect(),
            [
                T.Row(id='row2', st=T.Row(field='value_2'), ar=[6.0, 5.0, 4.0]),
                T.Row(id='row1', st=T.Row(field='value_1'), ar=[2.0, 3.0, 1.0]),
```

```
            ],
            atol=0.5,
        )
```

4.3 Instant Iterative Development

The slowest part in Spark integration testing is context initialisation. `SparklyGlobalSessionTest` allows you to keep the same instance of spark context between different test cases, but it still kills the context at the end. It's especially annoying if you work in TDD fashion. On each run you have to wait 25-30 seconds till a new context is ready. We added a tool to preserve spark context between multiple test runs.

Note: In case if you change `SparklySession` definition (new options, jars or packages) you have to refresh the context via `sparkly-testing refresh`. However, you don't need to refresh context if `udfs` are changed.

4.4 Fixtures

"Fixture" is a term borrowed from Django framework. Fixtures load data to a database before the test execution.

There are several storages supported in Sparkly:

- Elastic
- Cassandra (requires `cassandra-driver`)
- Mysql (requires `PyMySql`)
- Kafka (requires `kafka-python`)

```
from sparkly.test import MysqlFixture, SparklyTest


class MyTestCase(SparklyTest):
    ...
    fixtures = [
        MysqlFixture('mysql.host',
                     'user',
                     'password',
                     '/path/to/setup_data.sql',
                     '/path/to/remove_data.sql')
    ]
    ...
```

Column and DataFrame Functions

A counterpart of pyspark.sql.functions providing useful shortcuts:

- a cleaner alternative to chaining together multiple when/otherwise statements.
- an easy way to join multiple dataframes at once and disambiguate fields with the same name.
- agg function to select a value from the row that maximizes other column(s)

5.1 API documentation

CHAPTER 6

Generic Utils

These are generic utils used in Sparkly.

CHAPTER 7

License

```
                              Apache License
                        Version 2.0, January 2004
                     http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
```

```
      represent, as a whole, an original work of authorship. For the purposes
      of this License, Derivative Works shall not include works that remain
      separable from, or merely link (or bind by name) to the interfaces of,
      the Work and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, including
      the original version of the Work and any modifications or additions
      to that Work or Derivative Works thereof, that is intentionally
      submitted to Licensor for inclusion in the Work by the copyright owner
      or by an individual or Legal Entity authorized to submit on behalf of
      the copyright owner. For the purposes of this definition, "submitted"
      means any form of electronic, verbal, or written communication sent
      to the Licensor or its representatives, including but not limited to
      communication on electronic mailing lists, source code control systems,
      and issue tracking systems that are managed by, or on behalf of, the
      Licensor for the purpose of discussing and improving the Work, but
      excluding communication that is conspicuously marked or otherwise
      designated in writing by the copyright owner as "Not a Contribution."

      "Contributor" shall mean Licensor and any individual or Legal Entity
      on behalf of whom a Contribution has been received by Licensor and
      subsequently incorporated within the Work.

   2. Grant of Copyright License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      copyright license to reproduce, prepare Derivative Works of,
      publicly display, publicly perform, sublicense, and distribute the
      Work and such Derivative Works in Source or Object form.

   3. Grant of Patent License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      (except as stated in this section) patent license to make, have made,
      use, offer to sell, sell, import, and otherwise transfer the Work,
      where such license applies only to those patent claims licensable
      by such Contributor that are necessarily infringed by their
      Contribution(s) alone or by combination of their Contribution(s)
      with the Work to which such Contribution(s) was submitted. If You
      institute patent litigation against any entity (including a
      cross-claim or counterclaim in a lawsuit) alleging that the Work
      or a Contribution incorporated within the Work constitutes direct
      or contributory patent infringement, then any patent licenses
      granted to You under this License for that Work shall terminate
      as of the date such litigation is filed.

   4. Redistribution. You may reproduce and distribute copies of the
      Work or Derivative Works thereof in any medium, with or without
      modifications, and in Source or Object form, provided that You
      meet the following conditions:

      (a) You must give any other recipients of the Work or
          Derivative Works a copy of this License; and

      (b) You must cause any modified files to carry prominent notices
          stating that You changed the files; and

      (c) You must retain, in the Source form of any Derivative Works
```

```
          that You distribute, all copyright, patent, trademark, and
          attribution notices from the Source form of the Work,
          excluding those notices that do not pertain to any part of
          the Derivative Works; and

      (d) If the Work includes a "NOTICE" text file as part of its
          distribution, then any Derivative Works that You distribute must
          include a readable copy of the attribution notices contained
          within such NOTICE file, excluding those notices that do not
          pertain to any part of the Derivative Works, in at least one
          of the following places: within a NOTICE text file distributed
          as part of the Derivative Works; within the Source form or
          documentation, if provided along with the Derivative Works; or,
          within a display generated by the Derivative Works, if and
          wherever such third-party notices normally appear. The contents
          of the NOTICE file are for informational purposes only and
          do not modify the License. You may add Your own attribution
          notices within Derivative Works that You distribute, alongside
          or as an addendum to the NOTICE text from the Work, provided
          that such additional attribution notices cannot be construed
          as modifying the License.

      You may add Your own copyright statement to Your modifications and
      may provide additional or different license terms and conditions
      for use, reproduction, or distribution of Your modifications, or
      for any such Derivative Works as a whole, provided Your use,
      reproduction, and distribution of the Work otherwise complies with
      the conditions stated in this License.

   5. Submission of Contributions. Unless You explicitly state otherwise,
      any Contribution intentionally submitted for inclusion in the Work
      by You to the Licensor shall be under the terms and conditions of
      this License, without any additional terms or conditions.
      Notwithstanding the above, nothing herein shall supersede or modify
      the terms of any separate license agreement you may have executed
      with Licensor regarding such Contributions.

   6. Trademarks. This License does not grant permission to use the trade
      names, trademarks, service marks, or product names of the Licensor,
      except as required for reasonable and customary use in describing the
      origin of the Work and reproducing the content of the NOTICE file.

   7. Disclaimer of Warranty. Unless required by applicable law or
      agreed to in writing, Licensor provides the Work (and each
      Contributor provides its Contributions) on an "AS IS" BASIS,
      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
      implied, including, without limitation, any warranties or conditions
      of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
      PARTICULAR PURPOSE. You are solely responsible for determining the
      appropriateness of using or redistributing the Work and assume any
      risks associated with Your exercise of permissions under this License.

   8. Limitation of Liability. In no event and under no legal theory,
      whether in tort (including negligence), contract, or otherwise,
      unless required by applicable law (such as deliberate and grossly
      negligent acts) or agreed to in writing, shall any Contributor be
      liable to You for damages, including any direct, indirect, special,
      incidental, or consequential damages of any character arising as a
```

```
      result of this License or out of the use or inability to use the
      Work (including but not limited to damages for loss of goodwill,
      work stoppage, computer failure or malfunction, or any and all
      other commercial damages or losses), even if such Contributor
      has been advised of the possibility of such damages.

   9. Accepting Warranty or Additional Liability. While redistributing
      the Work or Derivative Works thereof, You may choose to offer,
      and charge a fee for, acceptance of support, warranty, indemnity,
      or other liability obligations and/or rights consistent with this
      License. However, in accepting such obligations, You may act only
      on Your own behalf and on Your sole responsibility, not on behalf
      of any other Contributor, and only if You agree to indemnify,
      defend, and hold each Contributor harmless for any liability
      incurred by, or claims asserted against, such Contributor by reason
      of your accepting any such warranty or additional liability.

   END OF TERMS AND CONDITIONS

   APPENDIX: How to apply the Apache License to your work.

      To apply the Apache License to your work, attach the following
      boilerplate notice, with the fields enclosed by brackets "[]"
      replaced with your own identifying information. (Don't include
      the brackets!)  The text should be enclosed in the appropriate
      comment syntax for the file format. We also recommend that a
      file or class name and description of purpose be included on the
      same "printed page" as the copyright notice for easier
      identification within third-party archives.

   Copyright 2017 Tubular Labs, Inc.

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.


=========================================================================
Sparkly Subcomponents:

The Sparkly project contains subcomponents with separate copyright
notices and license terms. Your use of the source code for the these
subcomponents is subject to the terms and conditions of the following
licenses.

==========================================================================
Apache licenses
==========================================================================

The following dependencies are provided under a Apache license. See project link for
↪details.
```

```
(Apache License 2.0) Spark (https://github.com/apache/spark)
(Apache License 2.0) cassandra-driver (https://github.com/datastax/python-driver)

========================================================================
BSD-style licenses
========================================================================

The following dependencies are provided under a BSD-style license. See project link␣
↪for details.

(BSD License) mock (https://github.com/testing-cabal/mock)
(PSF License) Sphinx (https://github.com/sphinx-doc/sphinx)

========================================================================
MIT licenses
========================================================================

The following dependencies are provided under the MIT License. See project link for␣
↪details.

(MIT License) sphinx_rtd_theme (https://github.com/snide/sphinx_rtd_theme)
(MIT License) pytest (https://github.com/pytest-dev/pytest)
(MIT License) pytest-cov (https://github.com/pytest-dev/pytest-cov)
(MIT License) PyMySQL (https://github.com/PyMySQL/PyMySQL)
```

CHAPTER 8

Indices and tables

- genindex
- modindex
- search