
Sumo Notebooks Documentation

Release 0.1

Armin Wasicek

Sep 19, 2018

Contents:

1	Getting Started	3
1.1	Running the Sumo Notebooks Docker Container	3
1.2	Setting the Access Keys	4
2	Data Science Workflow	7
2.1	Data Exploration using Spark SQL	7
2.2	Clustering Example	8
2.3	Clustering Example using Jupyter	10
3	Troubleshooting	13
3.1	Common Errors	13
3.2	Debugging	13
4	Limitations	15
5	Other Documentation	17
6	Indices and tables	19

Sumo Notebooks provide a way to seamlessly access data stored in Sumo for the purpose of data exploration and statistical analysis. The notebooks provide an interactive way to gain and share insights of a dataset. Built on top of Apache Zeppelin and Jupyter, Sumo Notebooks provide a state-of-the-art user experience coupled with access to the most recent machine learning frameworks such as Apache Spark, tensorflow, etc to unlock the value of machine data.

Note: This is an experimental feature under development.

Sumo Notebooks are organized as a docker container that assembles all dependencies in a single container. This simplifies installing and updating. A Sumo Notebooks container gets access to an organization's Sumo data store via Sumo's REST API. For that purpose, we require you to create an access key for Sumo as described in [this guide](#). After creating access credentials, please follow these steps to install and run the Sumo Notebooks container.

1.1 Running the Sumo Notebooks Docker Container

0. Open a shell or terminal on your computer
1. Load the SumoLab docker container on your computer: `docker pull sumologic/notebooks:latest`

Note: It is a prerequisite to have a working [docker](#) installed.

2. Start the container. API access id and access key have to be submitted via command line to work with the Jupyter notebook but can be either submitted via the command line or entered via the Spark interpreter configuration menu in Zeppelin.

```
docker run -d -it -p 8088:8080 -e SUMO_ACCESS_ID='XXX' -e SUMO_ACCESS_KEY='XXX' -e SUMO_ENDPOINT='XXX' sumologic/notebooks:latest
```

3. Open the Zeppelin UI and find some sample notebooks under the 'Notebook' drop down menu or see the 'Demo.ipynb' on opening Jupyter on the browser.

Application	Ports	Link
Zeppelin	8088	http://localhost:8088
Spark Web UI	4040	http://localhost:4040
Tensorboard	XXXX	http://localhost:XXXX
Jupyter	4000	http://localhost:4000

This is it, happy coding!

1.1.1 Docker Container Environment Variables

There is a set of environment variables for Sumo Notebooks that can be set when starting the docker container. The `docker push` command provides the `-e` switch to define these variables.

Variable	Description
SUMO_ACCESS_ID	Access Id token from Sumo, usually a base64url encoded string.
SUMO_ACCESS_KEY	Access key token from Sumo, usually a base64url encoded string.
SUMO_ENDPOINT	A https URL denoting the Sumo deployment to connect to. (Needed by Zeppelin)
ZEP- PELIN_SPARK_WEBUI	This variable controls where the “Spark Job” link in a paragraph points. (Needed by Zeppelin)

1.2 Setting the Access Keys

Sharing access id/key with the Sumo Notebooks container can be done using two methods:

- Submitting SUMO_ACCESS_ID and SUMO_ACCESS_KEY environment variables to the container as shown in the previous section
- Setting or changing the access id/key pair within the Zeppelin web UI as shown below. This is not available in Jupyter.

1.2.1 Step 1

Click on the username in the right top corner of the Zeppelin web UI. Scroll down or enter `spark` in the search bar to get to the Spark configuration page.

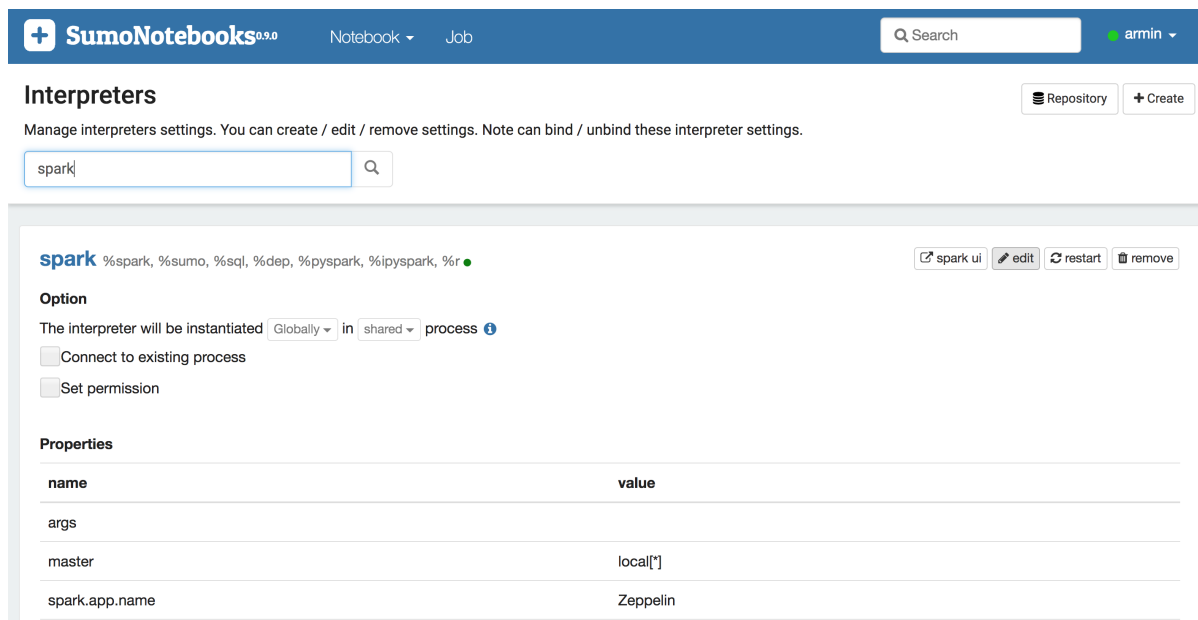
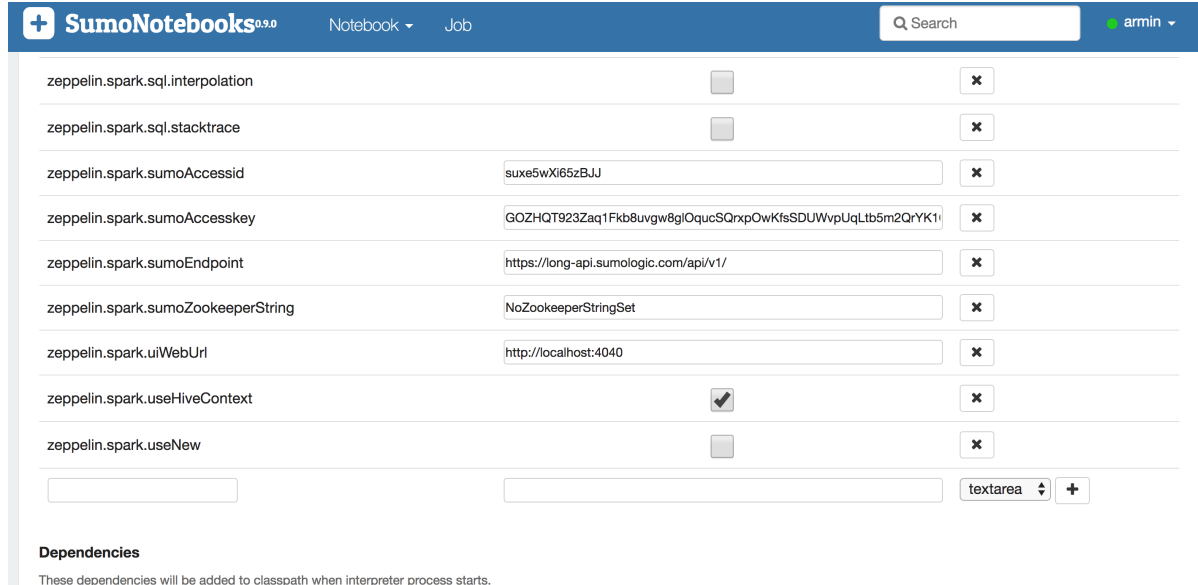


Fig. 1: Open the Spark interpreter configuration page

1.2.2 Step 2

On the Spark configuration page click the `edit` button and then enter access id, access key, and http endpoint in the according text fields. An overview of the Sumo endpoints for the different deployments is listed on [this page](#). Finally, save the configuration and the interpreter will restart with the new configuration.



The screenshot shows the SumoNotebooks 0.9.0 interface. The top navigation bar includes a search bar and a user dropdown menu showing 'admin'. The main content area is a configuration table with the following rows:

Property	Value	Action
zeppelin.spark.sql.interpolation	<input type="checkbox"/>	✕
zeppelin.spark.sql.stacktrace	<input type="checkbox"/>	✕
zeppelin.spark.sumoAccessid	suxe5wXi65zBJJ	✕
zeppelin.spark.sumoAccesskey	GOZHQ7923Zaq1Fkb8uvvgw8glOqucSQrxpOwKfsSDUWpUqLtb5m2QrYK1	✕
zeppelin.spark.sumoEndpoint	https://long-api.sumologic.com/api/v1/	✕
zeppelin.spark.sumoZookeeperString	NoZookeeperStringSet	✕
zeppelin.spark.uiWebUrl	http://localhost:4040	✕
zeppelin.spark.useHiveContext	<input checked="" type="checkbox"/>	✕
zeppelin.spark.useNew	<input type="checkbox"/>	✕

Below the table, there are two empty text input fields and a 'textarea' button with a plus icon.

Dependencies
These dependencies will be added to classpath when interpreter process starts.

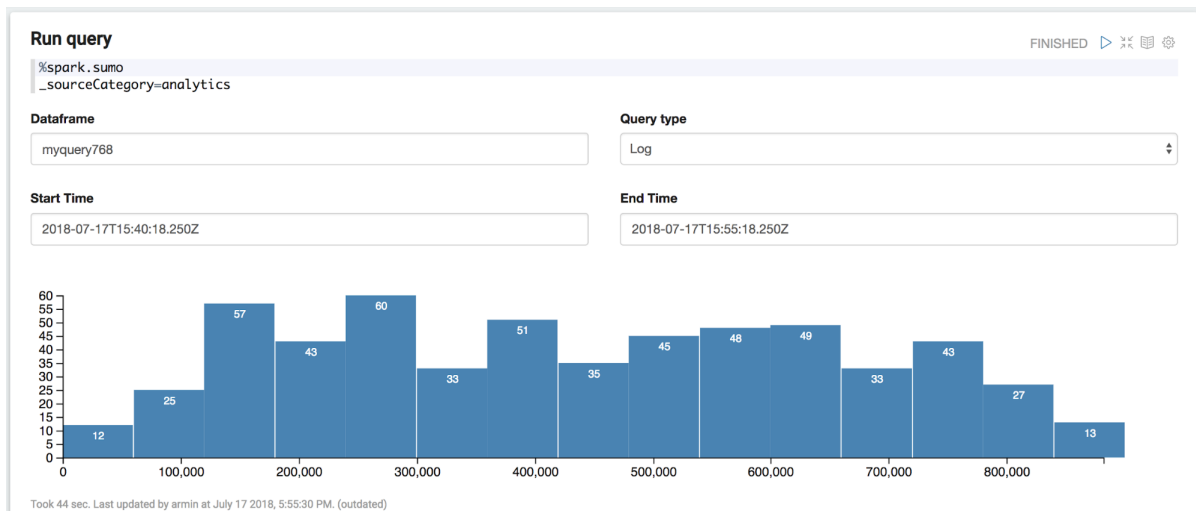
Fig. 2: Enter the access id/key pair and save

Data Science Workflow

The foundational data structure for Sumo notebooks is a data frame. A typical data science workflow manipulates data frames in many ways. For instance, data frames might be transformed for feature generation and statistical analysis, or joined with another dataset for enrichment. Therefore, a Sumo notebook returns query results in a Spark dataframe. This enables users to tap into Spark’s development universe, or – using the `toPandas` method – switch over to a python-native approach for data analytics.

2.1 Data Exploration using Spark SQL

This workflow focuses on loading log data from Sumo and then performing data exploration using Spark SQL.



First thing is to instruct Zeppelin to use the Sumo interpreter by entering `%spark.sumo` in the first line of the paragraph. This annotation indicates that the paragraph is routed to the Sumo interpreter running in the backend. This interpreter checks the query, connects to Sumo using access id and access key and retrieves the data. The data is represented as a

`Spark DataFrame` and can be used as such through the name displayed in the DataFrame field. In this example this is `myquery768`.

Note: In fact it uses a customized version of the `sumo-java-client`, therefore it has the same restrictions.

A DataFrame bound to this name exists in the scope of the Spark Scala shell, this is can be manipulated via the Spark Scala API. When sharing the reference through the Zeppelin context, it can also be used in PySpark (see next tutorial). The DataFrame as also registered as a temporary table in Spark SQL.

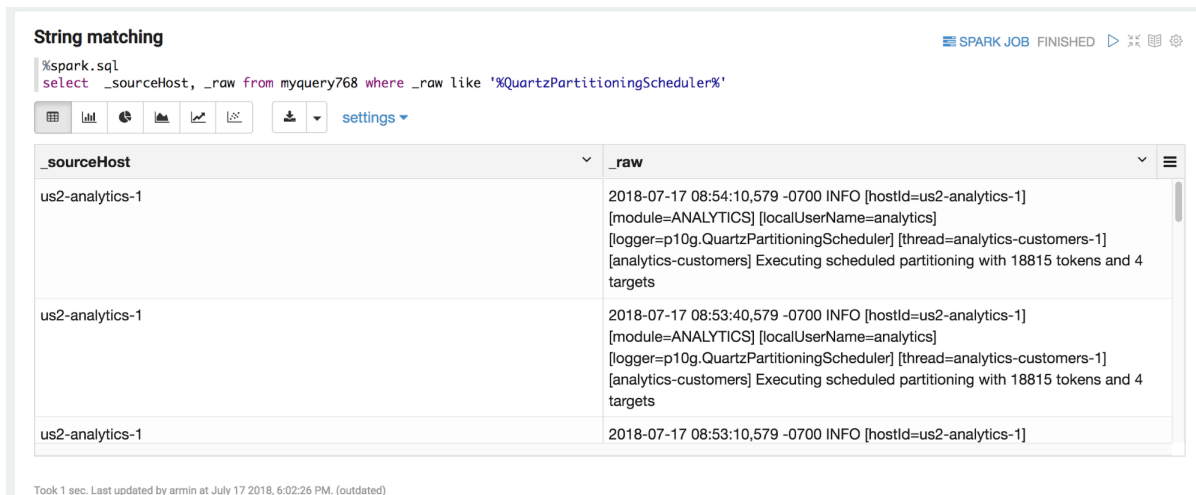


Aggregation

```
%spark.sql
select _sourceHost, count(*) from myquery768 group by _sourceHost
```

_sourceHost	count(1)
us2-analytics-2	74
us2-analytics-1	306
us2-analytics-3	81
us2-analytics-4	113

Starting the paragraph with `%spark.sql` designates the Spark SQL interpreter and SQL queries can be entered to explore the data. This example produces counts from raw data. Counting is a common pre-processing task for subsequent statistical learning task.



String matching

```
%spark.sql
select _sourceHost, _raw from myquery768 where _raw like '%QuartzPartitioningScheduler%'
```

_sourceHost	_raw
us2-analytics-1	2018-07-17 08:54:10,579 -0700 INFO [hostId=us2-analytics-1] [module=ANALYTICS] [localUserName=analytics] [logger=p10g.QuartzPartitioningScheduler] [thread=analytics-customers-1] [analytics-customers] Executing scheduled partitioning with 18815 tokens and 4 targets
us2-analytics-1	2018-07-17 08:53:40,579 -0700 INFO [hostId=us2-analytics-1] [module=ANALYTICS] [localUserName=analytics] [logger=p10g.QuartzPartitioningScheduler] [thread=analytics-customers-1] [analytics-customers] Executing scheduled partitioning with 18815 tokens and 4 targets
us2-analytics-1	2018-07-17 08:53:10,579 -0700 INFO [hostId=us2-analytics-1]

Took 1 sec. Last updated by armin at July 17 2018, 6:02:26 PM. (outdated)

Another common operation on logs is string matching. Spark SQL's `SELECT` provides a set of operations to filter and aggregate data.

2.2 Clustering Example

This example is about leveraging the python interpreter to perform a basic clustering operation on metrics data. As usual, `%spark.sumo` leads in a Sumo query. This time a metrics query is submitted. Metrics queries can be specified by selecting `_Metrics_` via the drop down menu.

Load metrics data from Sumo

```
%spark.sumo
_sourceCategory=search metric=CPU_Stolen | avg by _sourceHost
```

myquery865

Query type

Metrics

Start Time

2018-07-17T15:57:15.921Z

End Time

2018-07-17T16:0:15.921Z

```
myquery865: org.apache.spark.sql.DataFrame = [timestamp: bigint, 4b456d: double ... 41 more fields]
```

Took 1 sec. Last updated by armin at July 17 2018, 6:13:17 PM. (outdated)

Similarly to the log queries, the result is a *DataFrame*. As this *DataFrame* lives in the Spark Scala world we need to share it via the *Zeppelin context* with the python interpreter

Share data frame in Zeppelin Context

```
z.put("myquery865", myquery865)
```

Took 0 sec. Last updated by armin at July 17 2018, 6:16:12 PM. (outdated)

Import Spark DataFrame in python and convert to pandas data frame

```
%spark.pyspark
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pyspark.sql import DataFrame

dfp = DataFrame(z.get("myquery865"), sqlContext)

df = dfp.toPandas()

df['timestamp'] = pd.to_datetime(df['timestamp'], unit='ms')
df = df.set_index('timestamp')

df
```

4b456d

6b4d7e

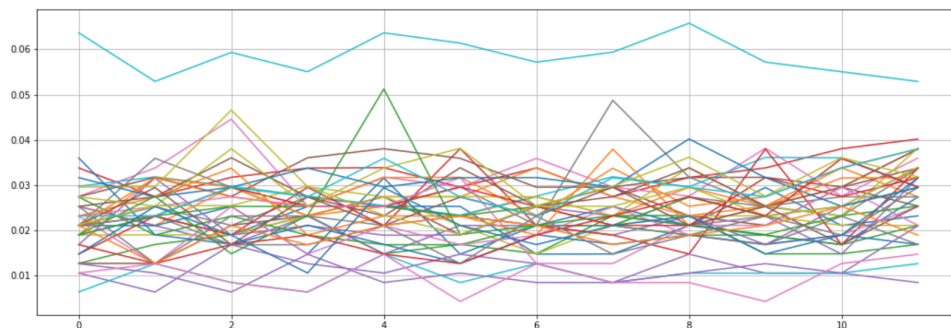
67e7e4

63d47c

After retrieving the *DataFrame* in the python interpreter and loading it as a pandas data frame, the powerful world python machine learning frameworks opens up! First, some visual exploration using matplotlib reveals is done.

Plot metrics data over time

```
%spark.pyspark
plt.figure(figsize=(18,6))
plt.plot(df.values)
plt.grid(True)
```



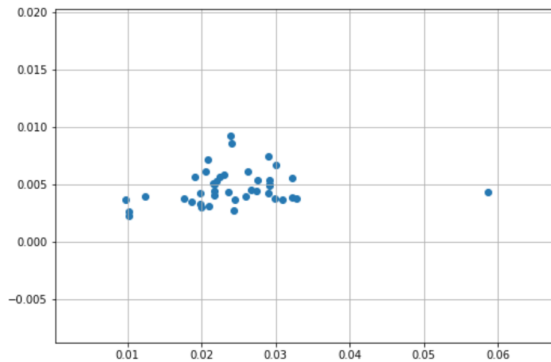
Next, each time series is featurized to simple (mean, std) pairs and plotted as a scatter plot. Visual inspection reveals that there might be some clusters!

Scatter plot of (mean, std) pair for each instance

FINISHED ▶ 🔍 📄 ⚙️

```
%spark.pyspark
matrix = np.array([df.mean().values, df.std().values])

plt.scatter(matrix[0,:], matrix[1,:])
plt.grid(True)
```



Using this intuition, a first try is to run the dbscan algorithm from the sklearn package.

Use sklearn and dbscan to cluster instances

FINISHED ▶ 🔍 📄 ⚙️

```
%spark.pyspark
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

X = StandardScaler().fit_transform(matrix.transpose())

dbscan = DBSCAN(eps=0.4, min_samples=4).fit(X)

core_samples_mask = np.zeros_like(dbscan.labels_, dtype=bool)
core_samples_mask[dbscan.core_sample_indices_] = True
labels = dbscan.labels_
```

Took 0 sec. Last updated by armin at July 17 2018, 6:36:10 PM. (outdated)

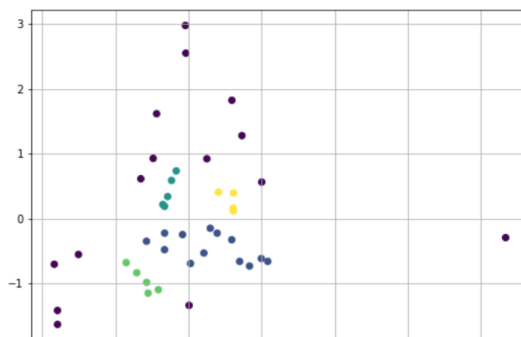
And there we are, yes there are a couple of clusters in the that particular metric.

Scatter plot of results with color coding

FINISHED ▶ 🔍 📄 ⚙️

```
%spark.pyspark

plt.scatter(X[:,0], X[:,1], c=labels)
plt.grid(True)
```



2.3 Clustering Example using Jupyter

We can see the same clustering example shown above as executed on the Jupyter notebook.

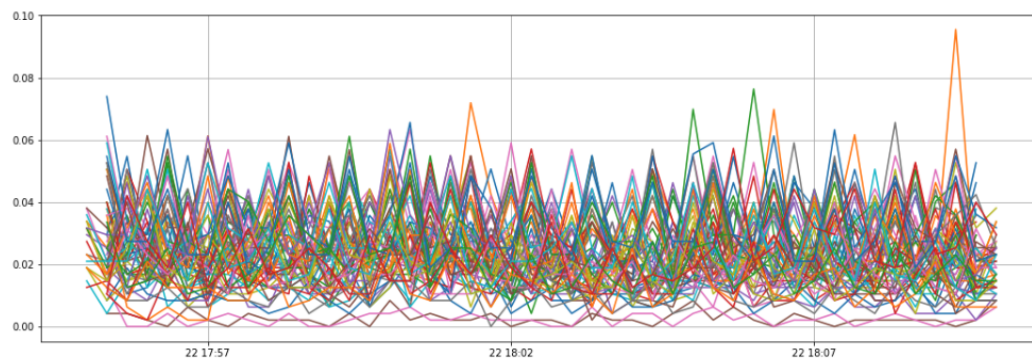
We initialize a SumoLab object to get a simple interface to enter the query and time range parameters. Once added we can *Run* the query.

```
In [1]: from sumolab import *
%matplotlib notebook
sm_objA = SumoLab('metric')
sm_objA.load_query()
```

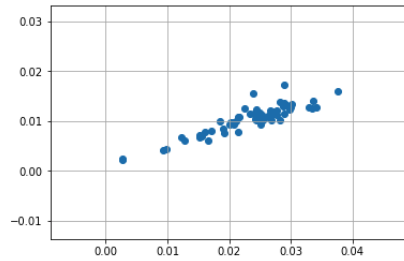
Query:	<input type="text" value="_sourceCategory=search metric=CPU_Stolen avg by _sourceHost"/>	Type:	<input type="text" value="metric"/>
Start Time:	<input type="text" value="2018-08-22T10:55:11.522890-07:00"/>	End Time:	<input type="text" value="2018-08-22T11:10:11.522866-07:00"/>
		TimeZone:	<input type="text" value="PST"/>
<input type="button" value="Run"/>			

Once we have the data-frame we can follow the same procedure explained above to perform clustering of the data.

```
In [3]: data = sm_objA.data.resample('20S').sum(min_count=1)
plt.figure(figsize=(18,6))
plt.plot(data)
plt.grid(True)
```



```
In [7]: matrix = np.array([data.mean().values, data.std().values])
plt.scatter(matrix[0,:], matrix[1,:])
plt.grid(True)
```



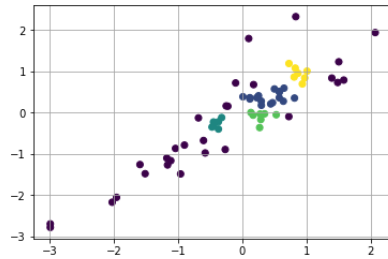
```
In [17]: from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

X = StandardScaler().fit_transform(matrix.transpose())

dbscan = DBSCAN(eps=0.2, min_samples=4).fit(X)

core_samples_mask = np.zeros_like(dbscan.labels_, dtype=bool)
core_samples_mask[dbscan.core_sample_indices_] = True
labels = dbscan.labels_

plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.grid(True)
```



3.1 Common Errors

No data or exception:

- Make sure to have the right access credentials in place

TTransportException or timeout

- Restart the interpreter: Use black, top right gear for pulling up interpreter menu, then push restart icon on the left of the blue bar listing the interpreter group for spark, finally save on the bottom.

3.2 Debugging

The SparkSumoMemoryCache object is a key value store that holds the context of the most recent operations. It can be used to inquire on exceptions and results that are retrieve behind the scenes.

Key	Type	Description
lastDataFrame	DataFrame	Holds a reference to the last data frame that has been created
lastLogQueryException	Exception	References the last exception (if any) that the log backend threw
lastMetricsQueryException	Exception	References the last exception (if any) that the log backend threw
lastMetricsDimensions	HashMap[String, String]	Dictionary to resolve the metrics column header to the actual dimensions
lastQueryJob	JobId	References the last job id returned from the search api
lastTriplet	QueryTriplet	Last processed query
metricsClient	SumoMetricsClient	Client used to retrieve metrics from Sumo
sumoClient	SumoClient	Client used to retrieve logs from Sumo
sparkSession	SparkSession	Spark session that is used to ingest the data

3.2.1 Code example

```
val spark = SparkSumoMemoryCache.get("sparkSession").get.asInstanceOf[SparkSession]
```

CHAPTER 4

Limitations

Current limits of the REST API are documented [here](#).

Zeppelin is started with these parameters:

- `ZEPPELIN_INTP_MEM="-Xmx10g"`
- `SPARK_SUBMIT_OPTIONS="-driver-memory 2g"`

CHAPTER 5

Other Documentation

- [Apache Zeppelin Documentation](#)
- [HortonWorks: How to diagnose zeppelin](#)
- [MapR: Troubleshooting Zeppelin](#)
- [Qubole: Debugging Notebook Issues](#)

CHAPTER 6

Indices and tables

- `genindex`
- `modindex`
- `search`