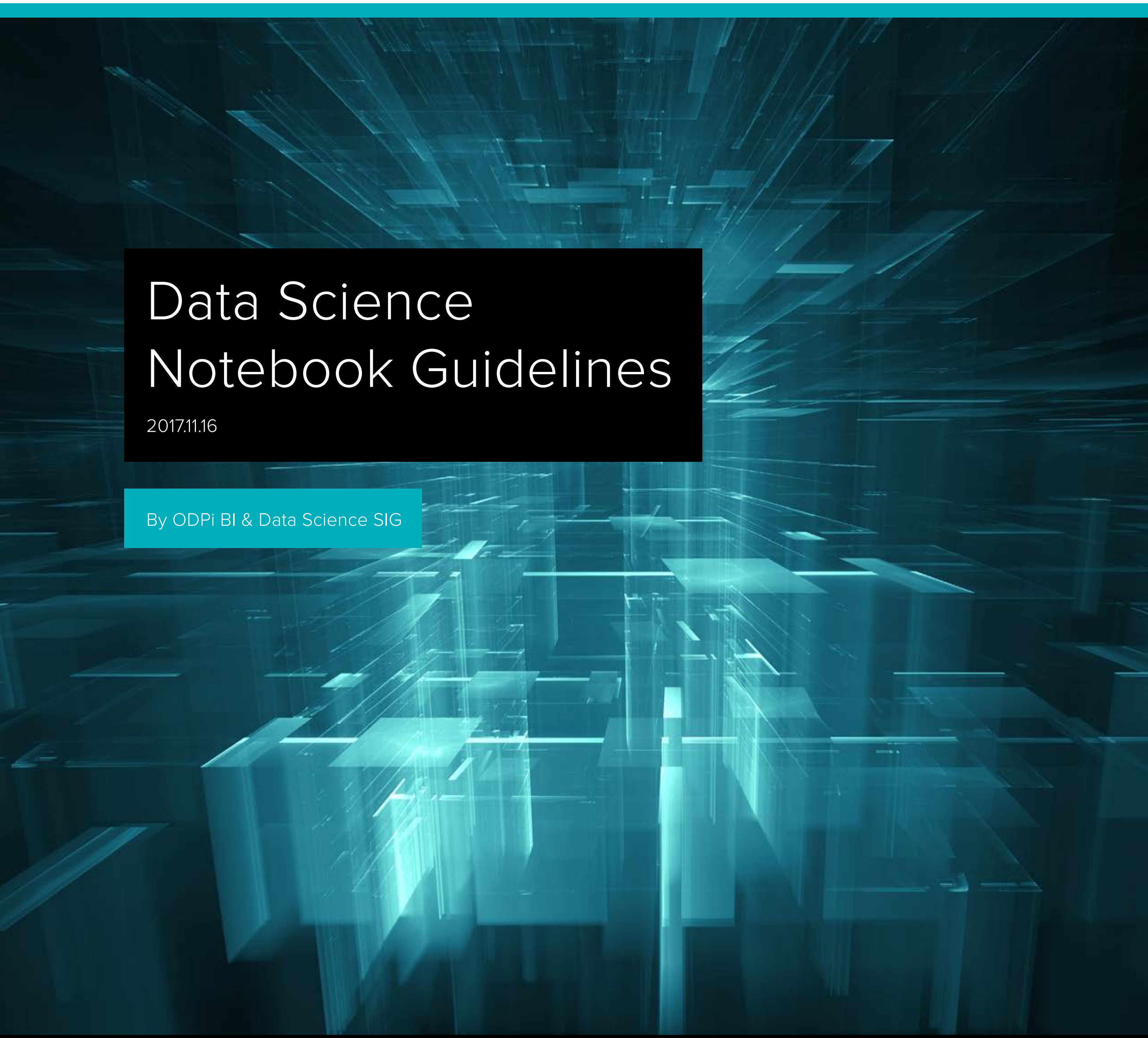


Data Science Notebook Guidelines

2017.11.16

By ODPi BI & Data Science SIG

odpi.org

Table of Contents

Overview of BI & Data Science SIG

Technology for Business Intelligence (BI) has been developed since the 1990s. Most of the BI tools were built because of the growth of Relational Database Management Systems (RDBMS). After many years of development and evolution, BI is now a pretty mature area that is embraced by most modern corporations.

As the Hadoop ecosystem becomes more and more popular, organizations have started to invest in this new technology in the hopes of gaining more business value. Though there are a lot of fundamental differences between traditional RDBMS and Hadoop, they also have a lot of similarities. Moreover, it's almost impossible for existing BI vendors to ignore their RDBMS investment and dive completely into Hadoop. Therefore, considering both in the roadmap is inevitable.

One goal of this SIG is to help bridging the gap so that BI tools can sit harmoniously on top of both Hadoop and RDBMS, while providing the same, or even more, business insight to the BI users who have also Hadoop in the backend. From a BI vendor perspective, this SIG should help to discover an effective way for connecting to, and querying, Hadoop without reinventing the wheel. From a BI user perspective, this SIG should help to provide an objective guideline for evaluating the effectiveness of a BI solution, and/or other related middleware technologies such as Apache Hive, Apache Drill, Presto, Apache Impala, Apache Phoenix… etc

Finally, statistical/data science models can now run on Hadoop in a much more cost effective manner than a few years ago. This results in fostering the convergence of traditional BI and Data Science disciplines (machine learning, artificial intelligence… etc). Hence, this SIG will also explore how this phenomenon is going to impact the Hadoop standard.

Data Science Notebook Overview

Even though there are a lot of interesting topics we can explore, the BI & Data Science SIG members voted and decided our first assignment is to recommend a guideline for Data Science Notebook.

As Data Science becomes more popular, interactive notebooks are becoming more and more prevalent in both the business intelligence and data science communities. They offer a combination of data analysis, presentation and collaboration capabilities that act like an electronic equivalent to notebooks kept by Galileo while observing Jupiter's moons. Or perhaps more personally, the composition notebooks kept during chemistry and physics classes.

One of the main advantages of data science workbooks is easy access to resources, whereby the notebook is hosted in a browser and can operate on large data sets residing in a cluster. Users do not need to be versed in distributed systems programming, and can focus on the tools that they know best like Python, R and SQL.

The ability to collaborate and communicate on projects, with less time spent on the mechanics of moving data around, has resulted in a significant productivity boost for both BI analysts and data scientists who make notebooks part of their workflow. Given these benefits, we are seeing a rapid innovation around open source notebook development and the addition of enterprise features.

Two of the most popular notebooks today are Jupyter and Apache Zeppelin. Jupyter has its roots in the Python programming language and was born in 2014 out of the IPython notebook project, which was originally started in 2001 by Fernando Perez.

The Jupyter project is named for the three most common notebook languages Julia, Python, and R. The name was changed from IPython to Jupyter to make it more obvious that notebooks were not just for Python programmers. Today, Jupyter supports dozens of other languages with a growing list of kernels and has a significant developer and user base. Jupyter is an open source software project carrying a modified BSD license. It is governed by a steering council of approximately 15 members from academia and commercial companies, who are the most active project contributors.

Zeppelin is a top level project in the Apache Software Foundation. It graduated in 2016 from ASF incubating status that it entered in 2014, after starting as a commercial product in 2013. Zeppelin has a modern look with a focus on built-in graphing and visualization capabilities, and an ability to have application-like interaction within the notebook interface, which makes it attractive to BI users. Zeppelin has been traditionally aligned with the Hadoop and Apache Spark communities, and it offers a built-in Spark integration which has gained a lot of loyal users, though the overall community is smaller than that of Jupyter. Zeppelin also supports other languages through available interpreters which connect Zeppelin to a specific language and data processing back end.

Notebook Comparison

Summary

Criteria	Jupyter	Zeppelin
Target user persona	Data science user with programming experience in one of the supported kernels.	Data engineer, data scientist and business users in the same data processing pipeline need to collaborate.
Installation	Easy installation with Anaconda or pip. Standalone, or Hadoop and Spark (via YARN) clusters supported.	Download binary package and start daemon script. Included in HDP.
Configuration	Edit config files or use command line tool for notebook settings. Community maintained language kernels have various configuration workflows.	Edit config files. Interpreters can be configured through GUI.
User interface	Functional notebook user interface that can be used to create readable analyses combining code, images, comments, formulae and plots.	Notebook interface that user can document, run codes, visualize outputs with flexible layout and multiple look and feel.
Supported languages	Large number (dozens) of community supported language kernels.	Various language supports are included in the binary package which Spark, Python, JDBC and etc. 3rd party interpreters are available through online registry.
Multi-user support	Native Jupyter does not support multi-user. However JupyterHub can be used to serve notebooks to users working in separate sessions.	Multiple users can collaborate in real-time on a notebook. Multiple users can work with multiple languages in the same notebook.
Support and community	Mature project with active community and good support. Jupyter project born in 2014 but has roots going back to 2001.	Apache Zeppelin is one of the most active project in Apache Software Foundation. Project born in 2013 and became top level project of ASF in 2015.
Architecture	The notebook server sends code to language kernels, renders in a browser, and stores code/output/Markdown in JSON files.	Zeppelin server daemon manages multiple interpreters (backend integrations). Web application communicates to server using websocket for realtime communication.
Big data ecosystem	Can be connected to a variety of big data execution engines and frameworks: Spark, massively parallel processing (MPP) databases, Hadoop, etc.	Tightly integrated with Apache Spark and other big data processing engines.
Security	Code executed in the notebook is trusted, like any other Python program. Token-based authentication on by default. Root use disabled by default/	User authentication (LDAP, AD integration) Notebook ACL. Interpreter ACL. SSL connection.
Data science readiness	Widely used by data scientists for a variety of tasks including quick exploration, documentation of findings, reproducibility, teaching, and presentations.	Data scientists can collaborate each other. Also business users can login and collaborate with data scientists directly on notebooks.

Details

Target user persona

Jupyter: Python is a requirement (Python 3.3 or greater, or Python 2.7) for installing the Jupyter Notebook. The installation documentation recommends using Anaconda to install Python and Jupyter on the client machine. Alternatively, Python's package manager pip can be used. Or, you can always download directly from the git repo.

Jupyter can be used without Spark or Hadoop. For example, you can connect it to a PostgreSQL database running on your laptop. Jupyter can also be powered up by a remote Spark cluster running under YARN. In this setup, you can leverage multiple Jupyter servers connected to a single cluster using the Jupyter kernel gateway.

Zeppelin: JAVA 1.7 (or later) is a requirement for installing Apache Zeppelin. Run the daemon script after download and extract the package. User can either download 'bin-all' package which includes all interpreters (Spark, Python, JDBC, etc.) or 'bin-netinst' which lets users install certain interpreters from network.

Configuration

Jupyter: Configuration workflow depends on whether you want to install other programming languages, add extensions, or set various configuration parameters. You can edit the notebook config files in the Jupyter folder or use the command line tool. It can take a bit of hunting around and trying various things to get the notebook configured the way that you want, with the desired community supported language kernels installed and working properly.

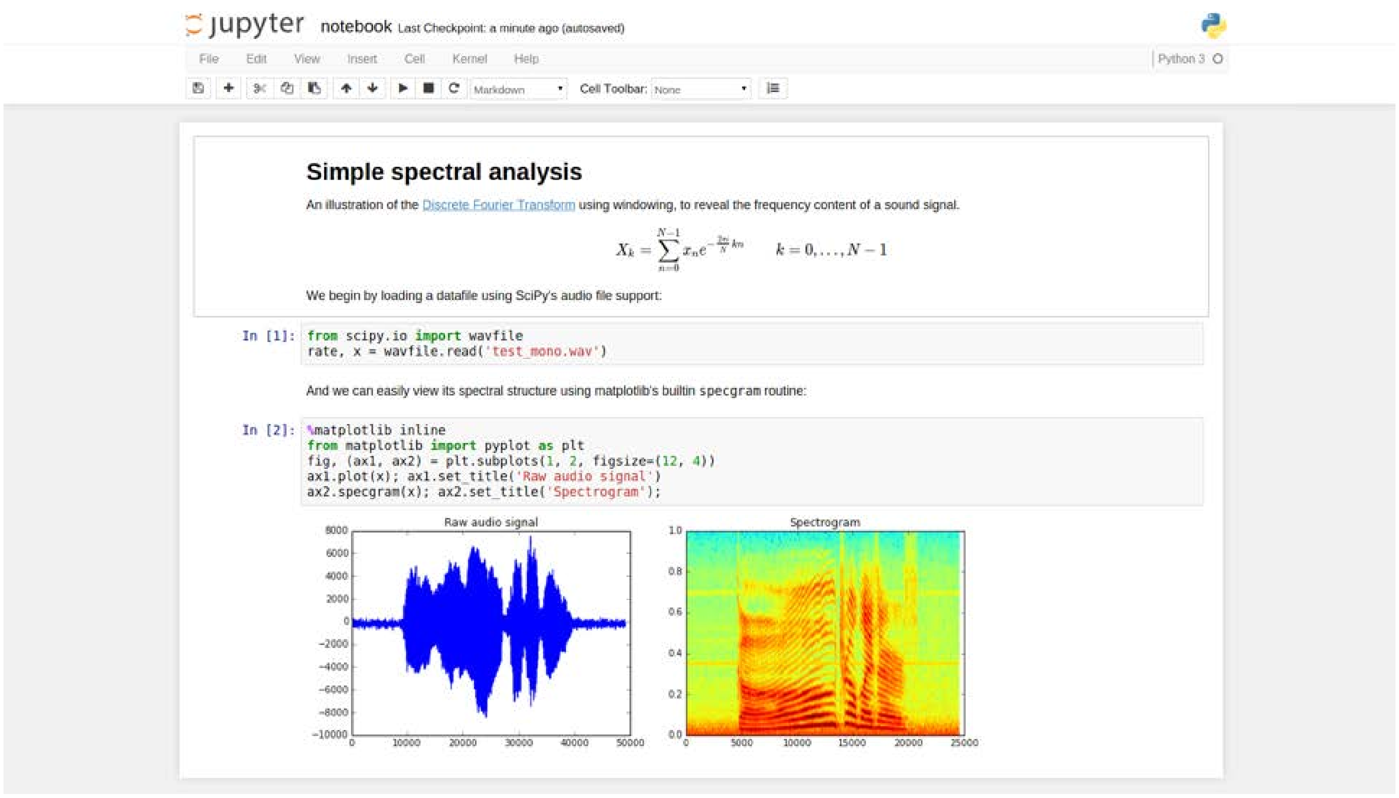
Zeppelin: Zeppelin works with zero configuration. When connection to external resources (like Spark cluster, JDBC, etc) are required, configuration can be done from the 'Interpreter' menu through the user interface.

Other configurations for Zeppelin (such as turning on authentication, configuring SSL, etc.) can be done through configuration files (conf/zeppelin-env.sh, conf/zeppelin-site.xm).

User Interface

Jupyter: Functional notebook user interface that can be used to create readable analyses combining code, images, comments, formulae and plots. The plotting is not built-in but you can use existing plotting libraries from R or Python, for example matplotlib (the de-facto standard). A good set of keyboard shortcuts are available to aid productivity.

There are some limitations that users have identified: No native code session saving when you exit the notebook. Can't drag and drop cells if you want to rearrange. No easy way to hide code or output of cells. Autocompletion would be a helpful feature, as well as some IDE type operations like syntax checking and version control.


jupyter notebook Last Checkpoint: a minute ago (autosaved)
File Edit View Insert Cell Kernel Help
Python 3
Markdown
Cell Toolbar: None

Simple spectral analysis

An illustration of the Discrete Fourier Transform using windowing, to reveal the frequency content of a sound signal.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

We begin by loading a datafile using SciPy's audio file support:

```
In [1]: from scipy.io import wavfile
        rate, x = wavfile.read('test_mono.wav')
```

And we can easily view its spectral structure using matplotlib's builtin specgram routine:

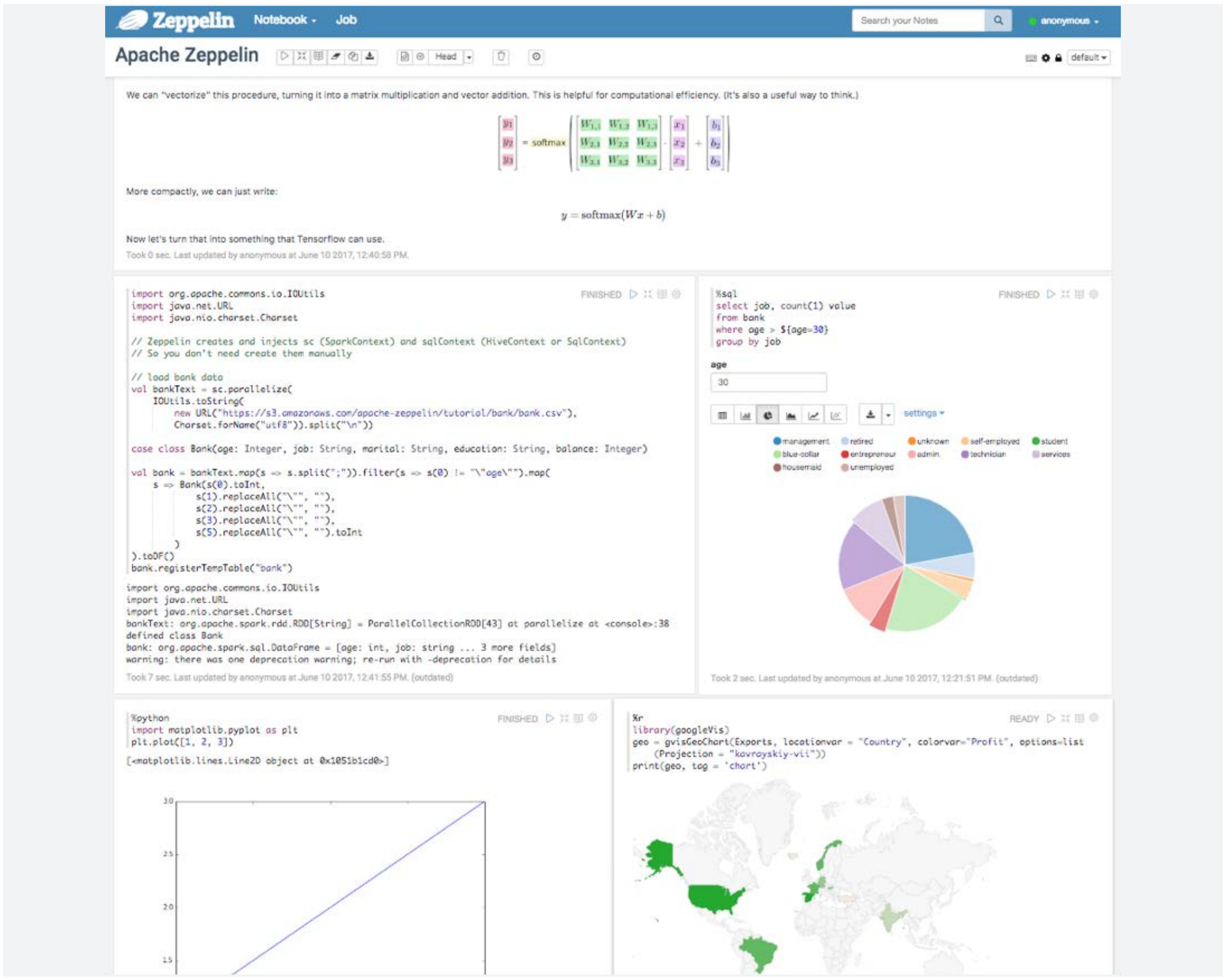
```
In [2]: %matplotlib inline
        from matplotlib import pyplot as plt
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
        ax1.plot(x); ax1.set_title('Raw audio signal')
        ax2.specgram(x); ax2.set_title('Spectrogram');
```

Jupyter Screenshot

Zeppelin: Notebook interface with flexible layout and multiple look and feel. The interface represents data analysis via code, results, forms, formulae, and plots. Supports flexible layout (paragraph side-by-side, change size of plots, show/hide code) and look and feel (report mode) helps deliver analysis to business users in an accessible way.

Zeppelin integrates matplotlib in Python or ggplot2 in R for engineers and scientists. Zeppelin also supports built-in visualization for business users, which only requires mouse click and drag and drop. To extend built-in visualization, Zeppelin community provides Helium online registry. Helium online registry enables additional visualization from the user interface menu with a few clicks. It enables sharing visualizations between developers.

There are some limitations that users have identified: Autocompletion does not always provide meaningful suggestions in many languages in Zeppelin. File naming convention is not human friendly and it makes manually managing notebook files difficult.



We can "vectorize" this procedure, turning it into a matrix multiplication and vector addition. This is helpful for computational efficiency. (It's also a useful way to think.)

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

More compactly, we can just write:

$$y = \text{softmax}(Wx + b)$$

Now let's turn that into something that Tensorflow can use.

Took 0 sec. Last updated by anonymous at June 10 2017, 12:40:58 PM.

```
import org.apache.commons.io.IOUtils
import java.net.URL
import java.nio.charset.Charset

// Zeppelin creates and injects sc (SparkContext) and sqlContext (HiveContext or SqlContext)
// So you don't need create them manually

// load bank data
val bankText = sc.parallelize(
    IOUtils.toString(
        new URL("https://s3.amazonaws.com/apache-zeppelin/tutorial/bank/bank.csv"),
        Charset.forName("utf8")).split("\n"))

case class Bank(age: Integer, job: String, marital: String, education: String, balance: Integer)

val bank = bankText.map(s => s.split(";")).filter(s => s(0) != "\"age\"").map(
    s => Bank(s(0).toInt,
            s(1).replaceAll("\"", ""),
            s(2).replaceAll("\"", ""),
            s(3).replaceAll("\"", ""),
            s(5).replaceAll("\"", "").toInt
        )
).toDF()
bank.registerTempTable("bank")
```

```
import org.apache.commons.io.IOUtils
import java.net.URL
import java.nio.charset.Charset
bankText: org.apache.spark.rdd.RDD[String] = ParallelCollectionRDD[43] at parallelize at <console>:38
defined class Bank
bank: org.apache.spark.sql.DataFrame = [age: int, job: string ... 3 more fields]
warning: there was one deprecation warning; re-run with -deprecation for details
```

Took 7 sec. Last updated by anonymous at June 10 2017, 12:41:55 PM. (outdated)

```
%sql
select job, count(1) value
from bank
where age > ${age=30}
group by job
```

Took 2 sec. Last updated by anonymous at June 10 2017, 12:21:51 PM. (outdated)

```
%python
import matplotlib.pyplot as plt
plt.plot([1, 2, 3])
```

```
[<matplotlib.lines.Line2D object at 0x1051b1cd0>]
```

```
%r
library(googleVis)
geo = gvisGeoChart(Exports, locationvar = "Country", colorvar="Profit", options=list
    (Projection = "kavrayskiy-vii"))
print(geo, tag = 'chart')
```

Zeppelin Screenshot

Supported languages

Jupyter: Jupyter team maintains the IPython kernel since the notebook depends on it to work. There is a large library of dozens of community supported language kernels offering data scientists multiple options to interact with the notebook, such as Scala, Python, Perl, PHP, Matlab, R, C, and SQL.

IPython magic functions can be used to mix multiple languages in the same notebook, for example Python and SQL. R and Python can be used in the same notebook using rpy2.

Zeppelin: Zeppelin community provides interpreters. Helium online registry lists all interpreters provided by both Zeppelin community and 3rd party. Interpreters include languages like Scala (for Spark), Python, R, SQL (JDBC), Groovy as well as various system integration like Spark, Cassandra, Pig, Flink, HBase and so on.

Multi-user Support

Jupyter: Native Jupyter does not support multi-user. However Jupyter Hub lets you create a multi-user hub which spawns, manages, and proxies multiple instances of the single-user Juptyer notebook server. Jupyter Hub can be used to serve notebooks to a relatively small group of users, although users are in different sessions so they cannot collaborate on the same notebook instance.

IBM's notebook offering is adding multi-user support for Jupyter Notebook (see section below).

Zeppelin: Zeppelin natively supports multi-user environment. Zeppelin can authenticate user using LDAP, active directory or any method that Apache Shiro supports.

Users can set owner, writer, reader on each notebook for access control. Interpreter also supports access control to restrict the access to backend system.

Notebook session is being shared in real-time so any change is propagated to other users who see the same notebook.

Interpreter session is also configurable, from sharing an interpreter session with all notebooks and users, to isolated interpreter sessions for each individual notebooks and only certain users

Support and community

Jupyter: Mature project with active community and good support. An inaugural conference called JupyterCon was held Aug 2017. As an example of engagement level, http://stackoverflow.com/search?q=jupyter returns 7,415 results as of 6/1/17. Born in 2014 out of the IPython notebook project, which was originally started in 2001 by Fernando Perez. Jupyter is an open source software project carrying a modified BSD license. It is governed by a steering council of approximately 15 members from academia and commercial companies, who are the most active project contributors.

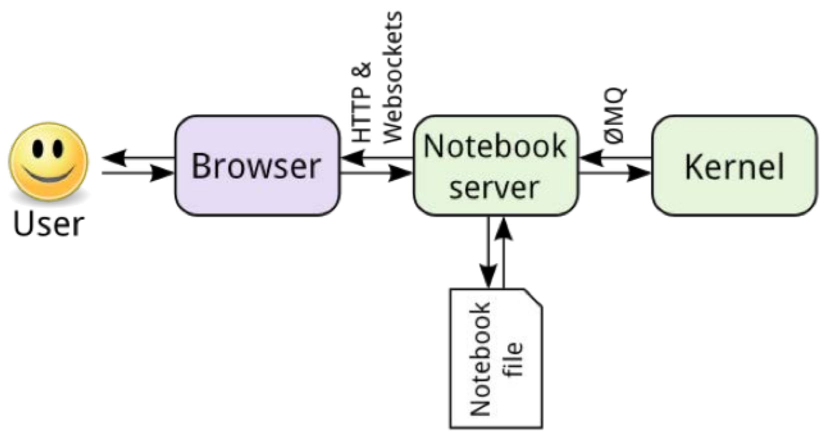
Zeppelin: Zeppelin is a top level project in Apache Software Foundation since May 2016. It has 9 members on the Project Management Committee, 14 committers and more than 200 code contributors world-wide as of 5/15/17. The project was originally started by Moon soo Lee at ZEPL in 2013. The project entered the Apache Incubator in Dec 2014.

As an example of engagement level, https://stackoverflow.com/search?q=zeppelin returns 1,280 results as of 6/1/17.

.

Architecture

Jupyter: How Jupyter notebooks work describes the operation of the notebook server as a place to store code, output and Markdown notes as a JSON file. The notebook server sends code to connected kernels to execute and renders in the browser.

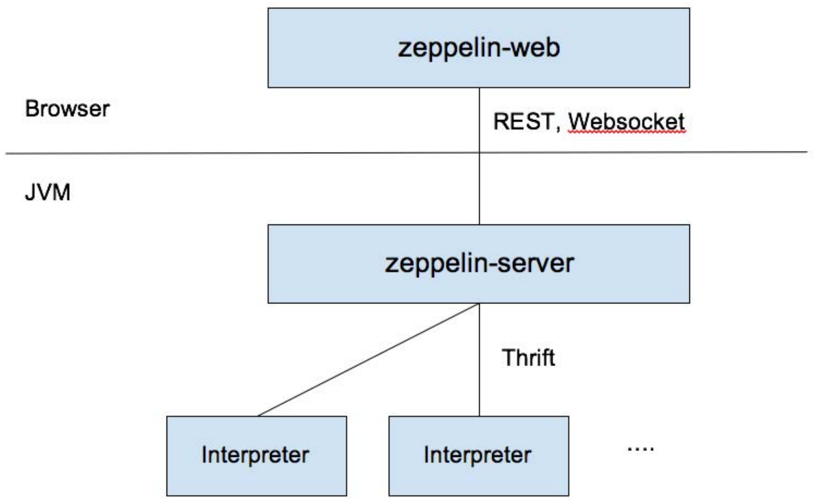


High-level Jupyter Architecture

Zeppelin: zeppelin-server process provides API endpoints and manages interpreters.

Each interpreter runs as a separate process and communicate with zeppelin-server via thrift protocol.

zeppelin-web runs on a browser and provides notebook user interface that communicates with zeppelin-server via REST API and WebSocket API.



High-level Zeppelin Architecture

Big Data Ecosystem

Jupyter: Jupyter notebooks can be connected to a variety of big data execution engines and frameworks: Spark, massively parallel processing (MPP) databases, Hadoop, etc. This allows the compute to occur on larger data sets that reside outside of the notebook client. In this way, only the result set is returned to the client machine running Jupyter for display or subsequent operations..

Zeppelin: Zeppelin integrates with various technologies that handle large scale of data sets: Apache Spark, Apache Flink, Apache Cassandra, etc. Integration not only includes code execution and display result but also display progress bar of job, canceling the job, link to cluster master (e.g. Spark master UI) for convenience. Also, Apache Hive is the only technology that can attach to Zeppelin that ODPi officially specifies.

Security and access control

Jupyter: The Jupyter security model assumes that code executed in the notebook is trusted, like any other Python program. HTML and Javascript in Markdown cells is never trusted; untrusted HTML is sanitized, and untrusted Javascript is never executed.

The Jupyter 4.25 release introduced token-based authentication that is on by default. The token is provided to the notebook server either in the authorization header, URL parameter, or password field. Alternatively, a notebook password can be set up.

IBM's notebook offering is adding the access control to Jupyter notebooks (see section below).

Zeppelin: Zeppelin supports authentication, notebook access control, and interpreter access control natively. Browser connection can be secured with SSL.

Data science readiness

Jupyter: Jupyter notebooks are widely used by data scientists since they provide an effective means do quick exploration tasks. They can also document research and make results reproducible. In addition, the web application can also be used as a presentation tool by mixing code, plots, graphics, HTML, Markdown cells, etc. Notebooks can be shared but cannot be worked on simultaneously by more than one user. If a user wants to overcome this limitation, Jupyter Lab is a good alternative.

The most popular languages used by data scientists today are Python, R and SQL. Popular Python-based data science libraries like Pandas and scikit-learn are readily accessible within Jupyter notebooks, as well as R using the R kernel, and SQL using SQL magic.

Zeppelin: In addition to conventional feature for data science notebook, data scientist can turn a notebook into a report for business users with few clicks. Business users can interact with notebooks using a dynamic form, built-in visualization and table pivot.

Data scientist can also schedule notebooks using a built-in cron-like scheduler in few clicks. In this way, a notebook can become part of the data processing pipeline.

Conclusion

As you can see from this guideline, there are many aspects we should consider when picking the “right” Data Science Notebook to enhance one’s analytics experience. While this guideline tries to cover as many useful aspect as possible, it is not our goal to announce the “winner” of the Data Science Notebook.

Readers are encouraged to use this guideline, which is a collaborative efforts of many experienced data professionals, as a starting point to explore the best alternative fitting the unique requirements. More importantly, one can even contribute development effort to improve the corresponding Notebook for a missing yet useful features.

if you have any feedback, please feel free to send us feedback at odpi-sig-bi@lists.odpi.org

Appendix 1:
Using Notebooks within containers

Containers are getting very popular for a clean application/ dependency packaging, as well as for doing a quick test drive of features. Instead of downloading and setting-up dependencies & changing the deployment environment, one can just pull a pre-built image and give it a spin on a single-node. Therefore, it is only logical to briefly mention this experience for the two popular notebooks.

Disclaimers:

1. *This isn't an exhaustive (or official) list of images available on Docker Hub; one can search for more pre-built images or customize an existing image to suit one's needs*
2. *Environment: Mac OS X 10.12.5, VirtualBox 5.1.22, docker-machine 0.7.0*

Zeppelin

Quite a few options here; following are the **top 3 images** on Docker Hub (stars/ pulls):

https://hub.docker.com/r/dylanmei/zeppelin/ (~1+GB compressed image, 2G on install)	Zeppelin with Spark 2.1 client and many commonly used interpreters. *A good image to test drive your spark examples in standalone mode.*
https://hub.docker.com/r/xemuliam/zeppelin/ (~800GB compressed; 1G on install)	Base Zeppelin standalone (interpreters have to be installed). **A good base image to extend from.**
https://hub.docker.com/r/epahomov/docker-zeppelin (3+GB compressed)	Zeppelin with Spark 2.1, Scala, R - with a bunch of libraries. *Opinionated; probably suited for a multi-language, all-Spark experience*

Once the images are pulled, starting a Zeppelin container is simple with a single command. Zeppelin also supports remote notebook repos on AWS, Azure and Git. To enable this, the user can configure the container after startup and save it instead of destroy to maintain the state for next use.

Jupyter

Again, many options are available, however, the official set of notebook variants are the most popular: https://github.com/jupyter/docker-stacks

datascience-notebook (2+GB compressed)	Jupyter, Python, R and a bunch of commonly used packages & libraries *A good image to test drive your data science examples in standalone mode.*
all-spark-notebook (2+GB compressed)	Jupyter Notebook with Python, Scala (Apache Toree), R, Spark, even Mesos Stack *Opinionated; probably suited for a multi-language, all-Spark experience*

Once the images are pulled, starting a Jupyter container is simple with a single command. Unlike Zeppelin, Jupyter saves notebooks with nice names in a predictable fashion.

It is worth mentioning a couple of more projects, which provide embedded Jupyter notebooks:

1. tensorflow: A lot of image variations (tags) depending on data science user requirements
2. gcr.io/cloud-datalab/datalab: This is oriented towards (Google's) cloud computations, but also have a nice 'local' mode that can run on user's local machine

Bonus: Spark-Notebook

For Spark-only Data science users, there's another variant of notebook available: http://spark-notebook.io/

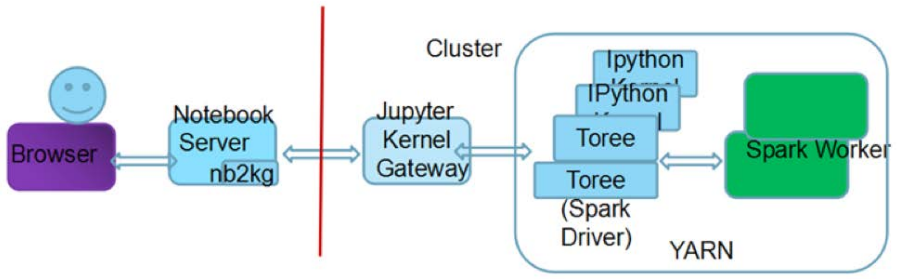
It includes interactive features to add dashboard capabilities to notebooks. Docker image is available here: https://hub.docker.com/r/andypetrella/spark-notebook/

Appendix 2: Jupyter Notebook Service on IBM Open Platform

IBM Open Data Platform is providing a Notebook service to the data scientists on the Hadoop IBM Open Platform (IOP).

The Jupyter Notebook has been gaining a lot of popularity after it evolved from the earlier known IPython Notebook project. The IPython Notebook has been around since 2001 which is one of the longest lifetime Notebooks. The original Jupyter Notebook is more intended for a single user case and very limited security. In order to gain more scalability, have multiple end users in a large organization to leverage and share the computation cycles of a cluster and make the Notebook service secured for the enterprise usage, we, IBM Open Platform(IOP) is providing a solution to power up the original Jupyter Notebook with our IOP Hadoop/Spark Cluster.

Below is the architectural diagram that explains the solution:



The end users use their browsers to connect to their Jupyter Notebook servers which reside outside of the IOP Hadoop/Spark cluster. Inside of the IOP cluster, we offer a gateway, a.k.a, the Jupyter Kernel Gateway(JKG) to handle multiple requests from different Jupyter Servers. Jupyter Kernel Gateway spawns the Jupyter Kernels to launch Spark jobs. Currently, we offer two Jupyter Kernels to support the Python and Scala language. The Python Kernel is based off the default IPython kernel that comes from the Jupyter Kernel Gateway and the Scala Kernel is an Open Sourced Jupyter Kernel that is developed by IBM named, Toree. The Jupyter Kernels, either IPython or Toree serves as the Spark Driver which launches Spark job on the IOP Cluster. The Spark jobs are running under the Yarn container which is the Spark execution model on the IOP cluster.

With our Jupyter Notebook solution, the end users installs their own Jupyter Server outside of the IOP cluster and connects to the Jupyter Kernel Gateway which sits inside of IOP Cluster behind the Knox security Gateway. The Jupyter Server is free to be obtained from the Open Sourced Jupyter project. The IBM DSX team has an enterprise offering of the Jupyter Server based off the Open Sourced version.   Multiple end users can have different Jupyter Servers of their own all connected to the same Jupyter Kernel Gateway on IOP. The user authentication is granted by the IOP Knox security gateway. The end user's notebook is kept on their own Jupyter Notebook servers.

In IOP 4.25, we offer the first release of this solution. We still have some limitations that we would like to develop in phases for the future releases:

Multiple end users can have their own user's credentials when connecting to Jupyter Notebook service on IOP Cluster. Their user credentials are checked by Knox. However, inside of the cluster, it is a shared cluster account which runs and manages all the Spark jobs created by each Notebook. From the IOP cluster's perspective, it is a single Notebook user who runs every one's Notebook jobs. This means that all Spark jobs launched by the Notebook service are all under this single cluster user's account. If the end users need to check the logs of the Spark job, all of them need share the same Notebook user's cluster account. In the future release, we will implement impersonation for the Notebook service user account on the IOP cluster in order to provide user account isolation.

The shared cluster Notebook user account is an ordinary user account on the cluster, but not a service account / super user for the cluster. This also means that the user will need to renew the Kerberos ticket manually or using a piece of code if the a Notebook runs more than the default Kerberos ticket active time window. Let's call this Time Window A. On the other hand, we also expose the tuning parameter for any Notebook that has not been active for a Time Window B to be killed and release the cluster resources. In most cases, Time Window A should be large than Time Window B. Therefore, the Kerberos ticket should not to be expected to be renewed often by the user.

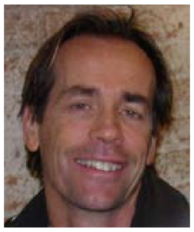
In summary, we offer multiple users to all be connected to the IOP cluster. We power up Jupyter Notebooks with large scale computation power. We provide user authentication via our Knox security gateway. Inside of the cluster we offer the Kerberization when the Spark job to access the Kerberized cluster.

Acknowledgment

Authors ***(in last name alphabetical order)***

Cupid Chan
Cupid Chan is a seasoned professional who is well-established in the data industry. His journey started out as one of the key players in building a world-class BI platform. Aside from holding various technical accreditations, his credential extends into business qualifications such as PMP and Lean Six Sigma. Cupid has been a consultant for years providing solutions to various Fortune 500 companies as well as the Public Sector.

Moon Soo Lee
Moon soo Lee is a creator for Apache Zeppelin and a Co-Founder, CTO at ZEPL. For past few years he has been working on bootstrapping Zeppelin project and its community. His recent focus is growing Zeppelin community and build healthy business around it.



Frank McQuillan
Frank McQuillan is Director of Product Management at Pivotal and based in Palo Alto, CA. He is responsible for data science tools at Pivotal, with a focus on distributed systems for reasoning over very large data sets. He is also a committer on the Apache MADlib project.

Tanping Wang
Tanping Wang is the chief architect for IBM Open Platform (IOP). Tanping has been contributing to Hadoop open source since its early years. Tanping architects and manages the infrastructure for IBM Open Platform based on Hadoop cluster. She is now focusing on providing Data Science Experience including Notebook powered by Hadoop to the end users.

BI & Data Science SIG Member (in last name alphabetical order)

- Cupid Chan, *4C Decision (SIG Champion)*
- Jared Dean, *SAS*
- Raj Desai, *IBM*
- Alan Gates, *Hortonworks*
- Nitin Lamba, *Ampool*
- Moon Soo Lee, *ZEPL*
- Frank McQuillan, *Pivotal*
- Ganesh Raju, *Linaro*
- Bikas Saha, *Hortonworks*
- Roman Shaposhnik, *The Linux Foundation*
- Tim Thorpe, *IBM*
- Tanping Wang, *IBM*