


Source: KDnuggets
TheSequence

<p>TheSequence</p> <p>(Core ML concepts + groundbreaking research papers and frameworks + AI news and trends) x 5 minutes, 3 times a week =...</p> <p>thesequence.substack.com</p>	
--	---

Notebooks are the data scientist best friend and can also be a nightmare to work with. For someone accustomed to work with modern integrated develop environments (IDEs), working with notebooks feels like going back decades. Furthermore, modern notebook environments is mostly constrained to Python programs and lack first-class support for other programming languages. A few days ago, [Netflix open sourced Polynote](#), a new notebook environment that addresses some of those challenges.

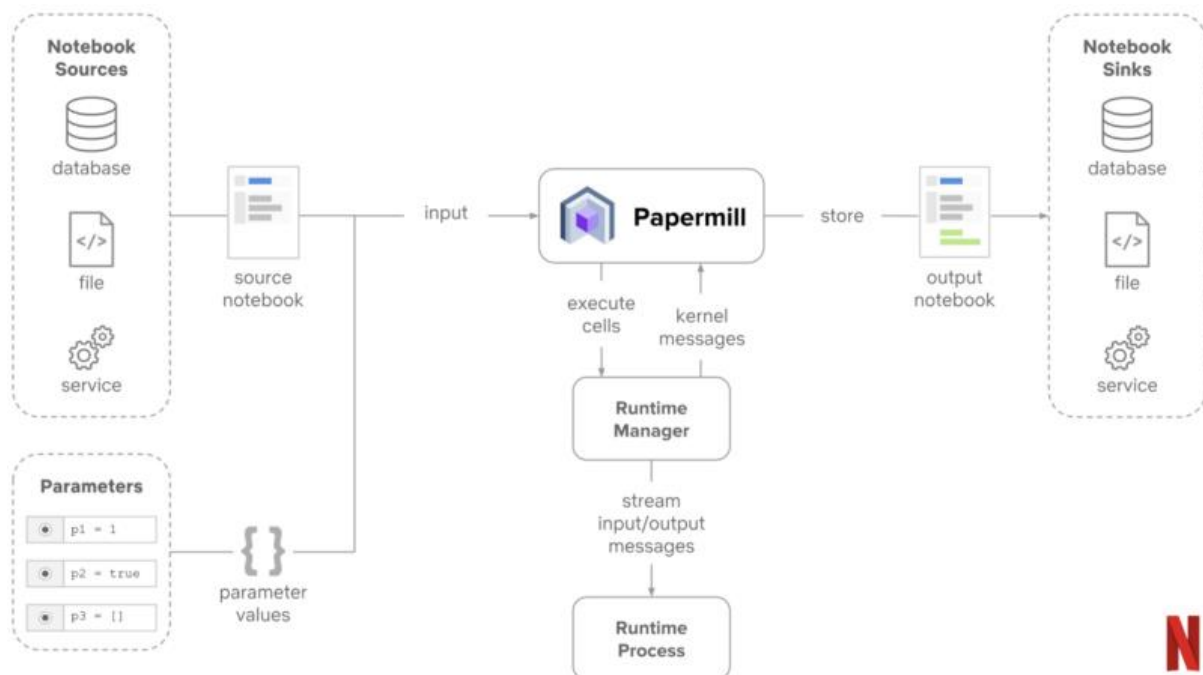
Polynote was born out of the necessity to accelerate data science experimentation at Netflix. Over the years, Netflix has built [a world-class machine learning platform](#) mostly based on JVM languages like Scala. The support for those languages in mainstream notebook technologies such as Jupyter is fundamentally basic so they needed a better solutions. Polynote was initiated by that basic requirement but incorporated the lessons learned building one of the most ambitious notebook-based experimentation platforms in the data science world.

Inside Netflix' Notebook Drive Architecture

Over the last few years, Netflix has transformed its use of data science notebooks from an experimentation artifact to a key component of the lifecycle of machine learning solutions. Initially, Netflix adopted Jupyter Notebooks like a data exploration and analysis tools. However, the engineering team quickly realized that Jupyter offered tangible advantages in terms of runtime abstraction, extensibility, interpretability of the code and debugging that could have a major impact in data science workloads if used correctly. In order to expand the use of Jupyter as a data science runtime, the Netflix team needed to solve a few major challenges:

- **The Code-Output Mismatch:** Notebooks are frequently changed and, many times, the output you are seeing in the environment does not correspond to the current code.
- **The Server Requirement:** Notebooks typically require a Notebook server runtime to run which represents an architecture challenge when adopted at scale.
- **Scheduling:** Most data science models need to be executed on a periodic basis but the tools for scheduling Notebooks are still fairly limited.
- **Parametrizing:** Notebooks are fairly static code-environments and the processes for passing input parameters are far from trivial.
- **Integration Testing:** Notebooks are isolated code- environments which notoriously difficult to integrate with other Notebooks. As a result, tasks like integration testing become a nightmare when using Notebooks.

To address those requirements, Netflix built a very ambitious architecture that enable the operationalization of Jupyter notebooks. The initial implementation included technologies such as [Papermill](#) which enables the parametrization of notebooks.



Source: <https://polynote.org/>

While the initial notebook architecture at Netflix was certainly ambitious, it was also constrained Python programs. Now it was time to expand.

Entering Polynote

Polynote is a multi-language notebook experimentation environment. In addition to Python, the current release supports languages such as SQL, Vega (visualizations) and, of course, Scala. The platform is also integrated with data science infrastructures such as Apache Spark. At its core, Polynote includes the following capabilities:

- Improved Editing Experience:** Polynote tries to enable an editing experience closer to modern IDEs.
- Multi-Language Support:** Polynote introduces first-class support for Scala and other languages used in data science environments.
- Data Visualization Improvements:** Polynote integrates native data visualizations into notebooks' dataset without the need of adding a lot of code.
- Configuration and Dependency Management:** Languages like Scala require complex package dependencies in its programs. Polynote saves the package dependency configuration within the notebook itself addressing some of the common challenges in this area experienced by JVM developers.
- Reproducibility:** The combination of code, data and execution results into a single document makes notebooks powerful, but also difficult to reproduce. Polynote includes reproducibility as a first-class capability of the framework.

Improved Editing Experience

Polynote includes common features in IDEs such as code auto-completion or syntax error highlighting which improves the experience for data scientists and researchers building Notebooks. More of the editing capabilities are powered by the [Monaco](#) editor which powers the experience of Visual Studio Code.

```

gs.map
    map[B, That](f: A => B)
    toMap[T, U](ev: A <:< T)
    flatMap[B, That](f: A => B)
    reverseMap[B, That](f:
    max[B](cmp: Ordering[B])
    maxBy[B](f: A => B)(cmp:

```

Source: <https://polynote.org/>

Multi-Language Support

Polynote does not only provide support for multiple languages but it also allows those languages to be combined in a single program. In Polynote, every cell can be based on a different language. When a cell is run, the kernel provides the available typed input values to the cell's language interpreter. In turn, the interpreter provides the resulting typed output values back to the kernel. This allows cells in Polynote notebooks to operate within the same context. The example below shows a Python library, to compute an isotonic regression of a dataset generated with Scala.

► Configuration & dependencies

scikit learn

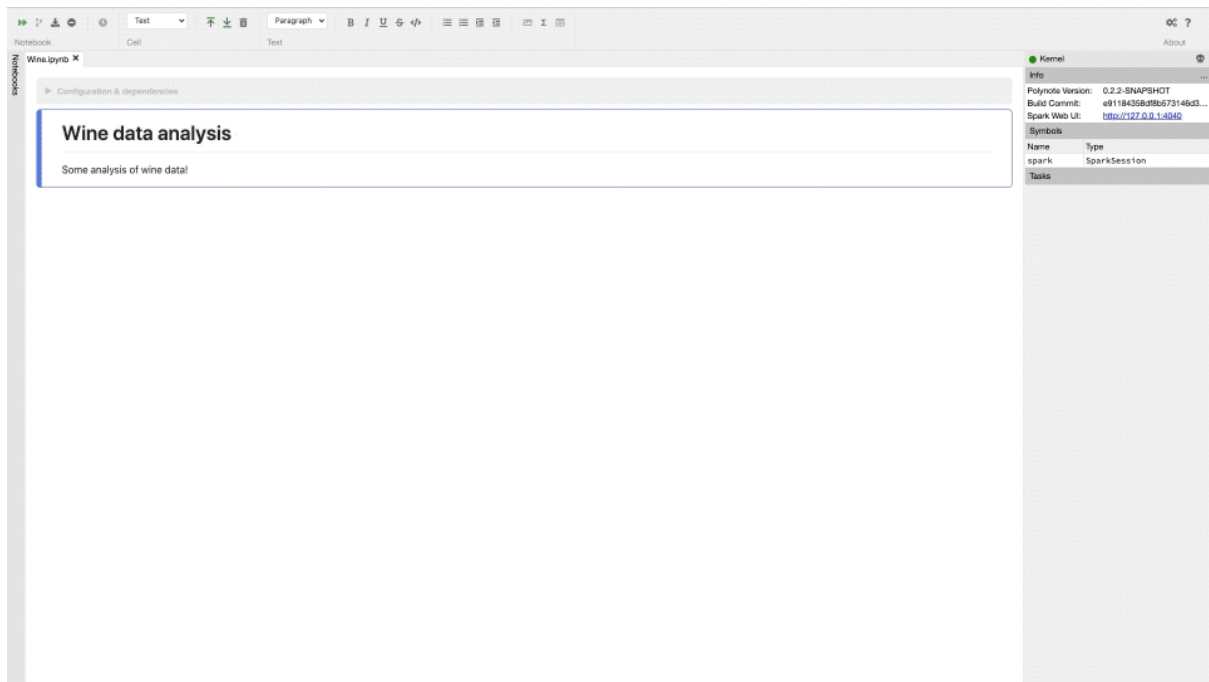
```
In(1): Python {} Ran on 10/16/2019, 3:35:55 PM PDT took 1s

1 # Copied from
2 # https://scikit-learn.org/stable/auto\_examples/plot\_isotonic\_regression.html#sphx-glr-auto-examples-p
3 #
4 # Author: Nelle Varoquaux <nelle.varoquaux@gmail.com>
5 # Alexandre Gramfort <alexandre.gramfort@inria.fr>
6 # License: BSD
7
8 import numpy as np
9 import matplotlib.pyplot as plt
10 from matplotlib.collections import LineCollection
11
12 from sklearn.linear_model import LinearRegression
13 from sklearn.isotonic import IsotonicRegression
14 from sklearn.utils import check_random_state
15
16 # Here's where the example generates the random numbers used in the experiment.
17 # We will generate these numbers in Scala, below.
18 # n = 100
19 # x = np.arange(n)
20 # rs = check_random_state(0)
21 # y = rs.randint(-50, 50, size=(n,)) + 50. * np.log1p(np.arange(n))
22
23 Out:
24
25
26 In(2): Scala {} Ran on 10/16/2019, 3:35:56 PM PDT took 1s
27
28 1 // Generate numbers for the regression models
```

Source: <https://polynote.org/>

Data Visualization Improvements

Data visualizations are a common component of most notebook environment. However, Polynote takes the visualization value proposition to another level by including it as a native component of the platform which does not require developers to write any code in order to visually explore a dataset.



Source: <https://polynote.org/>

Configuration and Dependency Management

Most of the time, data scientists working on notebooks can enjoy the efficiency of Python's package management model to handle the dependencies of a program. However, in JVM-languages like Scala dependency management can become a total night mare. Polynote addresses that challenge by storing the configuration and dependency information directly in the notebook itself, rather than relying on external files. Additionally, Polynote provides a user-friendly Configuration section where users can set dependencies for each notebook.

example.ipynb x foo.ipynb x

▼ Configuration & dependencies

Dependencies

You can provide Scala / JVM dependencies using Maven coordinates , e.g. `org.myorg:package-name_2.11:1.0.1`, or URLs like `s3://path/to/my.jar`
You can also specify pip packages, e.g. `requests`, or with a version like `urllib3==1.25.3`

scala/jvm	org.vegas-viz:vegas_2.11:0.3.11	⊖
pip	matplotlib	⊖
pip	seaborn	⊖
pip	plotly	⊖
scala/jvm	com.mycompany.supersecretproject:supersecret:1.2.3	⊖ ⊕

Resolvers

Specify any custom Ivy, Maven, or Pip repositories here.

Ivy	http://my.super.cool.company.repo/super-secret-releases	Artifact pattern (blank for default)	Metadata pattern (blank for default)	⊖ ⊕
-----	---	--------------------------------------	--------------------------------------	-----

Exclusions

[Scala only]: Specify organization:module coordinates for your exclusions, i.e. `org.myorg:package-name_2.11`

com.google.guava:guava	⊖ ⊕
------------------------	-----

Spark Config

Set Spark configuration for this notebook here. Please note that it is possible that your environment may override some of these settings at runtime :(

spark.driver.memory	20g	⊖
spark.executor.memory	1million	⊖
spark.otherConfig	magicValueThatMakesThingsWork	⊖ ⊕

Save & Restart Cancel

In(1): Scala {} ☰

Ran on 10/16/2019, 2:20:31 PM PDT took 7s

1 val x = 1

Source: <https://polynote.org/>

Reproducibility

With Polynote, Netflix a new code interpretation block instead of relying on a [REPL](#) model like a traditional notebook. One of the key capabilities of the new interpretation model is that it removes hidden states which allows data scientists to copy cells within a notebook without introducing any state from the previous position.

```
In(1): Python {} ≡
1 x = 1

In(2): Python {} ≡
1 print(x)

In(3): Python {} ≡
1 x = 2

In(4): Python {} ≡
1 print(x)
```

Source: <https://polynote.org/>