Project Report

MLOps Tool Analysis - Kedro

IDS 594 | CRN 45280 | Fall 2022

Team 13

Anupriya Rastogi	(arasto6@uic.edu)
Deepak Singhal	(dsingh39@uic.edu)
Rajaram Ramesh	(rrames8@uic.edu)

Project Goal

The primary focus of this project is on the deployment component because its main purpose is to demonstrate a knowledge of the basic concepts involved in deploying a model. In other words, given the project's scope, the ML model itself is less important and of lesser significance because it just serves as a placeholder for a deployment demonstration. Therefore, for this project, we decided to better understand Kedro as a key ML deployment tool and to show a simple example of its implementation.

Kedro

Kedro is a Python-based tool for orchestrating workflows. Back in 2019, QuantumBlack, a McKinsey AI firm launched Kedro. They describe Kedro as a code library that one can use to build data and machine learning pipelines which are considered the building blocks of any analytics and ML projects these days. To make the ML processes simpler and more precise, it would help develop reproducible, maintainable, and modular workflows. Kedro combines software engineering principles like modularity, separation of concerns, and versioning into a machine learning environment. Building a data pipeline using Kedro makes it simpler to create production-ready code, automate the "hard lifting," and reduce the amount of time needed for such work. Data cleansing, processing, and compilation are regular tasks that data scientists must complete on a regular basis even if they may not find these tasks to be their favorite activities. In January 2022, McKinsey/QuantumBlack donated Kedro to the Linux Foundation, where it would be hosted with the LF AI & Data organization.

Pros and Cons of using Kedro from a Business Point of View

Advantages/Features of using Kedro in projects are:

1. A **standard and easy-to-use project template**, allowing your collaborators to spend less time understanding how you’ve set up your analytics project
2. **Data abstraction**, managing how you load and save data so that you don’t have to worry about the reproducibility of your code in different environments
3. **Configuration management**, helping you keep credentials out of your code base

4. Promotes test-driven development and industry standard code quality, **decreasing operational risks for businesses**
5. **Modularity**, allowing you to break large chunks of code into smaller self-contained and understandable logical units
6. **Pipeline visualization** making it easy to see how your data pipeline is constructed
7. **Seamless packaging**, allowing to ship the projects to production using Docker, for example
8. **Versioning** for your datasets and machine learning models whenever your pipeline runs
9. Kedro's structure is especially useful for organizing complex projects that are difficult with other tools.
10. Its plugins can provide a wide range of operations to provide what Kedro does not make natively.
11. **Strong community**, given that community developments are so frequent, there is always bug fixing and release of new plugins.
12. **Reusability**, meaning node and pipeline architecture avoids code repetition as it allows functions reuse and thus clean coding.
13. It is possible to use different environments, such as development and production to separate the data and model sources.

Limitations and Disadvantages of using Kedro:
1. Given that Kedro is well structured, it may not work effectively if the project is too simple

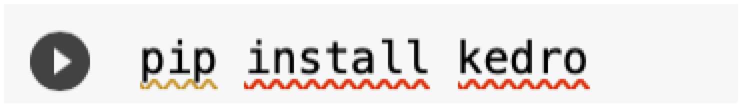
Cost Benefit Analysis

Kedro is a well-known open-source tool that helps in standardizing ML processes. While there is no cost for using Kedro, there could be other indirect costs associated with implementing a complete model involving other components needed to make the model and processes function effectively.

The Experience and Code Artifacts

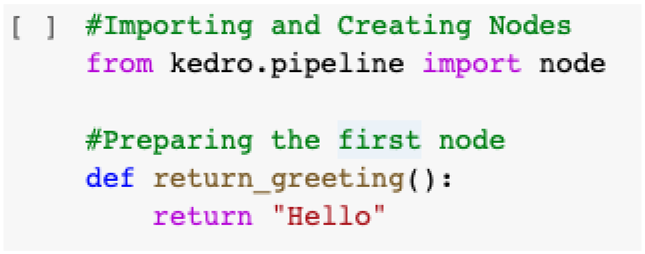
We had no trouble running the sample Kedro code in our Google Colab and Jupyter notebook. We noticed that the runtime is significantly shorter and easy, which could vary on a case by case basis. Kedro offers numerous intriguing features, and it excels at handling complex tasks. It separates the models and data sources, and it works well with many other tools. It can be easily configured to work with the majority of today's Machine Learning projects, including all accepted industry best practices. The code that we run in this project is a simple implementation code by Kedro showing the different elements of it.

The first step is to install the Kedro library to be able to use the features/methods of Kedro.



```
pip install kedro
```

The second step is to create nodes,

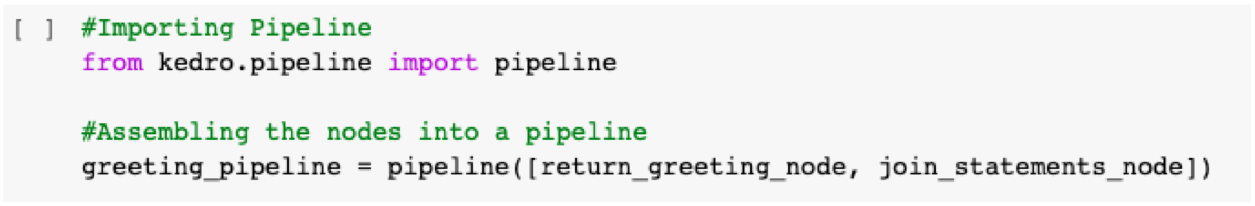


```
#Importing and Creating Nodes
from kedro.pipeline import node

#Preparing the first node
def return_greeting():
    return "Hello"
```

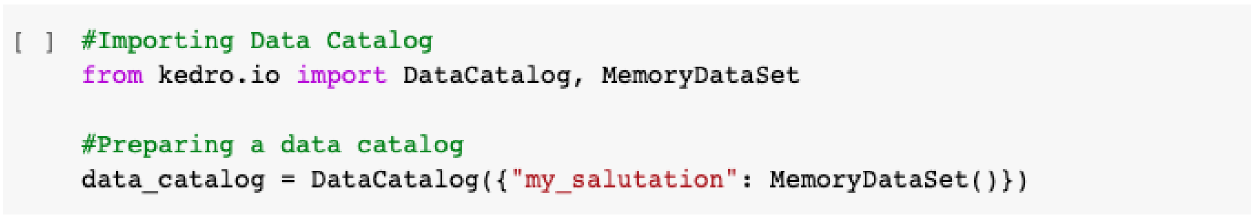
As stated in the Kedro guide, "Node is a Kedro concept. It is a wrapper for a Python function that names the inputs and outputs of that function. It is the building block of a pipeline. Nodes can be linked when the output of one node is the input of another."

Then, the next step would be to create a pipeline where the nodes are mentioned and finally create a Data catalog.



```
#Importing Pipeline
from kedro.pipeline import pipeline

#Assembling the nodes into a pipeline
greeting_pipeline = pipeline([return_greeting_node, join_statements_node])
```

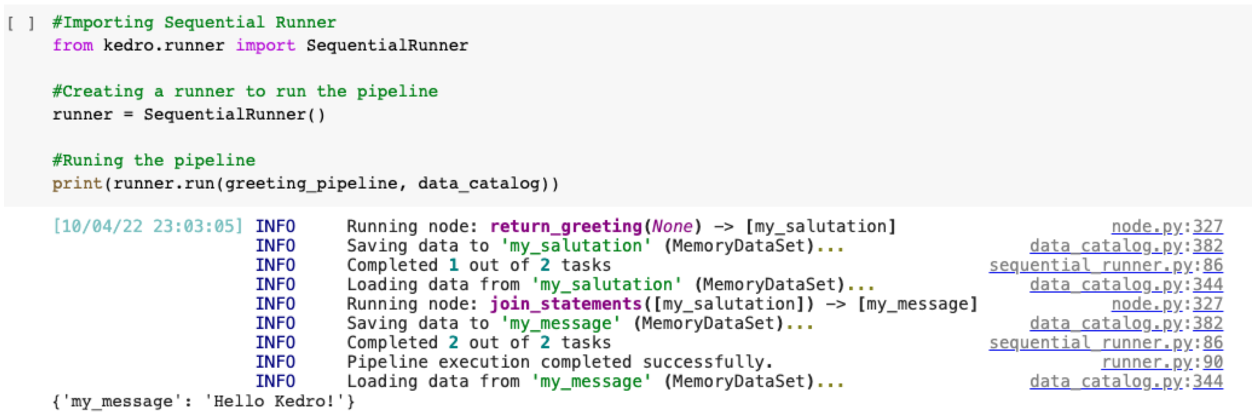


```
#Importing Data Catalog
from kedro.io import DataCatalog, MemoryDataSet

#Preparing a data catalog
data_catalog = DataCatalog({"my_salutation": MemoryDataSet()})
```

A pipeline, as stated in the Kedro guide, "organizes the dependencies and execution order of a collection of nodes and connects inputs and outputs while keeping your code modular. The pipeline determines the node execution order by resolving dependencies and does *not* necessarily run the nodes in the order in which they are passed in." A Data Catalog, as stated in the Kedro guide, "is the registry of all data sources that the project can use. It maps the names of node inputs and outputs as keys in a dataSet, which is a Kedro class that can be specialized for different types of data storage. Kedro uses a MemoryDataSet for data that is simply stored in-memory."

The final step is importing the Sequential Runner, which is an object that runs the pipeline, and printing the results which can be shown in the image as Hello Kedro!



```
[ ] #Importing Sequential Runner
    from kedro.runner import SequentialRunner

    #Creating a runner to run the pipeline
    runner = SequentialRunner()

    #Runing the pipeline
    print(runner.run(greeting_pipeline, data_catalog))
```

```
    [10/04/22 23:03:05] INFO     Running node: return_greeting(None) -> [my_salutation]           node.py:327
                        INFO     Saving data to 'my_salutation' (MemoryDataSet)...         data_catalog.py:382
                        INFO     Completed 1 out of 2 tasks                            sequential_runner.py:86
                        INFO     Loading data from 'my_salutation' (MemoryDataSet)...      data_catalog.py:344
                        INFO     Running node: join_statements([my_salutation]) -> [my_message]   node.py:327
                        INFO     Saving data to 'my_message' (MemoryDataSet)...            data_catalog.py:382
                        INFO     Completed 2 out of 2 tasks                            sequential_runner.py:86
                        INFO     Pipeline execution completed successfully.                      runner.py:90
                        INFO     Loading data from 'my_message' (MemoryDataSet)...         data_catalog.py:344
    {'my_message': 'Hello Kedro!'}
```

Kedro follows the order in which the nodes are executed. The below steps show how Kedro runs the pipeline:

- Kedro first executes *return_greeting_node*. This runs *return_greeting*, which takes no input but outputs the string "Hello".
- The output string is stored in the *MemoryDataSet* named *my_salutation*.
- Kedro then executes the second node, *join_statements_node*. This loads the *my_salutation* dataset and injects it into the *join_statements* function.
- The function joins the input salutation with "Kedro!" to form the output string *"Hello Kedro!"*
- The output of the pipeline is returned in a dictionary with key *my_message*.

Project Plan (Gantt Chart)

				Weeks				
	Task	Member(s) Assigned	Planned End Date	1	2	3	4	5
1	Understand Project Requirements	All	09/08/22	🟦				
2	Research and Understand the Kedro Tool	All	09/15/22		🟦			
3	Test Example Codes and Get Hands-On working with Kedro	Anupriya	09/22/22			🟦		
		Rajaram	09/22/22			🟦		
4	Work on Sections needed for Final Report	Deepak	09/29/22			🟦	🟦	
5	Finalize Code and Project Report for Submission	All	10/05/22					🟦

References

1. https://kedro.readthedocs.io/en/stable/
2. https://towardsdatascience.com/kedro-a-python-framework-for-reproducible-data-science-project-4d44977d4f04
3. https://medium.com/quantumblack/introducing-kedro-the-open-source-library-for-production-ready-machine-learning-code-d1c6d26ce2cf
4. https://towardsdatascience.com/learn-you-some-kedro-be67d4fc0ce7
5. https://kedro.readthedocs.io/en/stable/get_started/hello_kedro.html
6. https://towardsdatascience.com/kedro-prepare-to-pimp-your-pipeline-f8f68c263466
7. https://neptune.ai/blog/data-science-pipelines-with-kedro