Kedro

Cheat sheet - 1/2

Basics

Create and activate conda env (this is optional)

```
conda create --name my_kedro_env python==3.8.1
conda activate my_kedro_env
```

Install kedro (the latest version in pip)...

```
pip install kedro
```

... or install kedro (specific version as needed)

```
pip install kedro==0.17.5
```

Verify kedro installation

```
kedro info
```

Create new kedro project in cwd

```
kedro new
```

Create new kedro project in cwd with example project code

```
kedro new --starter=pandas-iris
```

Create new kedro project in cwd with pyspark configuration

```
kedro new --starter=pyspark
```

Initialise git repo in your new kedro project

```
cd path_to_your_project && git init
```

Install all project dependencies

```
kedro install
```

Run the kedro pipeline(s)

```
kedro run
```

Package your project, build .whl and .egg files

```
kedro package
```

Visualize your pipelines with kedro viz

```
pip install kedro-viz && kedro viz
```

Constructs

Kedro is broken up into a few major building blocks:

Node
Simply a wrapper for a Python function that defines optional inputs/outputs (data from the Data Catalog).

A Kedro Node must have at least 1 input or output.

Pipeline
Organises the dependencies and execution order of nodes.

Data Catalog
A registry of all the data sources that a project can utilise.
It handles loading and saving of data for you.
Define once here and reuse everywhere.

Runner
An object that runs the pipeline. You can choose from:
Sequential – executes pipeline nodes one-by-one
Parallel – enables concurrency via multiprocessing
Threaded – uses threading for concurrent execution

Parameters
One or many yml files which are used to store re-usable project configuration such as model tuning params or splits. Remove all hardcoding from your code by defining once here.

Folder structure

```
my_project          # Kedro Project Parent dir
├── conf              # Project configuration files
├── data              # Local project data*
├── docs              # Documentation
├── logs              # Output logs*
├── notebooks         # Jupyter notebooks
├── README.md         # Project README
├── setup.cfg         # Config for `pytest` and `isort`
└── src               # Project source code

* = Not committed to version control.
```

conf/
- base – project-specific settings to share with others.
- local – settings that should not be shared, e.g., sensitive credentials

data/
- raw – raw unprocessed data should live here, unchanged
- intermediate – optional data transformations applied to raw
- primary – cleaned, transformed and wrangled data
- feature – primary data grouped, and with features added
- model_input – specific feature data for a given analysis
- models – stored, serialised pre-trained model(s)
- model_output – results generated by the model(s)
- reporting – used to drive dashboards/views

Handling dependencies

There are two places dependencies are defined (both inside src/), as per pip compile best practices:

requirements.in – define your project dependencies
requirements.txt – generated file of pinned dependencies

For a new project, without a *requirements.in* file:

Add your dependencies to *requirements.txt* and generate *requirements.in*. After this, update only *requirements.in*:

```
kedro build-reqs
```

For a project, with a *requirements.in* file:
Add your dependencies to *requirements.in* only, then:

```
kedro build-reqs
```

To install all project dependencies from *requirements.txt:*

```
kedro install
```

Executing nodes & pipelines

Execute the default pipeline, sequentially

```
kedro run
```

Execute the default pipeline, with multiprocessing. Note: doesn't work with Spark data sets - use ThreadRunner.

```
kedro run --runner=ParallelRunner
```

Execute the default pipeline, with multithreading

```
kedro run --runner=ThreadRunner
```

Execute the pipeline asynchronously

```
kedro run --async
```

View all pipelines available

```
kedro pipeline list
```

Run a pipeline by name

```
kedro run --pipeline=my_pipeline
```

Run tagged nodes/pipeline

```
kedro run --tag=my_tag
```

Run a node by name

```
kedro run --node=my_node_name
```

Kedro

Cheat sheet - 2/2

Jupyter and ipython

Run all of these commands from Kedro project root.

Use Kedro within ipython

```
kedro ipython
```

Use Kedro within jupyter notebook

```
kedro jupyter notebook
```

Use Kedro within jupyter lab

```
kedro jupyter lab
```

From here, you have access to Kedro's 'context' and 'session' objects.

Load or save item from the Data Catalog

```
df = context.catalog.load("my_data")
context.catalog.save("catalog_item_name", my_df)
```

Access project parameters from parameters.yml

```
parameters = context.params # type: Dict
parameters["test_train_split"]
```

Execute the Pipeline in a notebook cell - see run options

```
session.run()
session.run(pipeline_name='my_pipeline')
```

Reload kedro variables (e.g. if you modify the catalog)

```
%reload_kedro
```

Load kedro variables into a standard ipython session

```
%load_ext kedro.extras.extensions.ipython
%reload_kedro <path_to_project_root>
```

...or

```
%load_ext kedro.extras.extensions.ipython
%init_kedro <path_to_project_root>
%reload_kedro
```

Data catalog code

Data Catalog code lives inside one or many catalog.yml files.

```
companies:
  type: pandas.ExcelDataSet
  filepath: data/01_raw/companies.xlsx

reviews:
  type: pandas.CSVDataSet
  filepath: data/01_raw/reviews.csv
```

Unique name of the catalog entry

Location on disk, inside the data folder. Note the layer (01_raw)

Type of data. Here we're using a Pandas CSV dataset. There are many out of the box data formats supported including Excel, S3, Azure, Spark, SQL Databases.

Node & pipeline code

Python node code typically lives within: `src/<your_project>/pipelines/<your_pipeline>/nodes.py`
Pipeline definition code, as below, lives within: `src/<your_project>/pipelines/<your_pipeline>/pipeline.py`

```
from kedro.pipeline import Pipeline, node
from .nodes import preprocess_companies, preprocess_shuttles

def create_pipeline(**kwargs):
    return Pipeline(
        [
            node(
                func=preprocess_companies,
                inputs=["companies"],
                outputs="preprocessed_companies",
                name="preprocess_companies_node",
            ),
            node(
                func=preprocess_shuttles,
                inputs=["shuttles"],
                outputs="preprocessed_shuttles",
                name="preprocess_shuttles_node",
            ),
        ]
    )
```

Import your node functions.

Define a pipeline. A pipeline consists of 1 or more nodes. You can add tags if you wish

Define a node. A node consists of a function to call, inputs, outputs, a name and optional tags. Must have 1 input or output

Define which function to call. A function is simply a python function

Define optional inputs. Inputs link to data catalog items, parameters or memory datasets. Can be a single item, a list or a dict.

Define optional outputs. As per inputs above. If you simply enter a string that doesn't exist in params or catalog, it will be saved to an in-memory dataset instead.

Be sure to update the project's pipeline in `src/<your_project>/pipeline_registry.py` so kedro knows to run your pipeline

```
from typing import Dict
from kedro.pipeline import Pipeline
from kedro_tutorial.pipelines import data_processing as dp

def register_pipelines() -> Dict[str, Pipeline]:
    data_processing_pipeline = dp.create_pipeline()

    return {
        "__default__": data_processing_pipeline,
        "dp": data_processing_pipeline,
    }
```

Set a default pipeline. This is the pipeline which will execute when you type: `kedro run`

Define a pipeline. The key of the dict here is used when you call a pipeline, e.g., `kedro run --pipeline=dp`

Created by Augment Partners

Learn more at https://kedro.readthedocs.io/