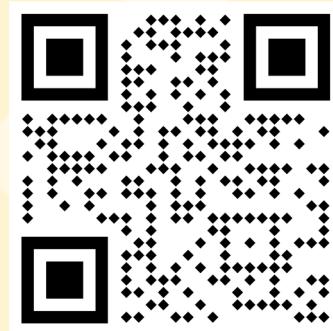


<https://github.com/pyladiesams/kedro-prod-ready-ds-pipelines-aug2025>





Data science that ships: production-ready pipelines with Kedro

Merel Theisen for PyLadies Amsterdam

26th of August 2025



CONFIDENTIAL AND PROPRIETARY
Any use of this material without specific permission of
McKinsey & Company is strictly prohibited



Agenda

1. About me
2. Common challenges for data practitioners
3. Intro to Kedro
4. Modularity
5. Separation of concerns
6. Maintainability
7. Q&A

About me

- Pronouns: she/her ⭐
- I'm from the Netherlands 🇳🇱
- I've been working as a Software Engineer for 9 years 💻
- Currently Principal Software Engineer at QuantumBlack ✨
- I've been the tech lead for Kedro for over 3 years 🚀



Contact

merel.theisen@quantumblack.com

DLF AI
& DATA

◆ kedro

01

Common challenges for data practitioners

Under-engineering refers to building with reduced complexity resulting in a less robust, efficient and capable product. It happens because of tight deadlines or because of a lack of expertise.

We under-engineer data science experiments & create code with a lot of technical debt

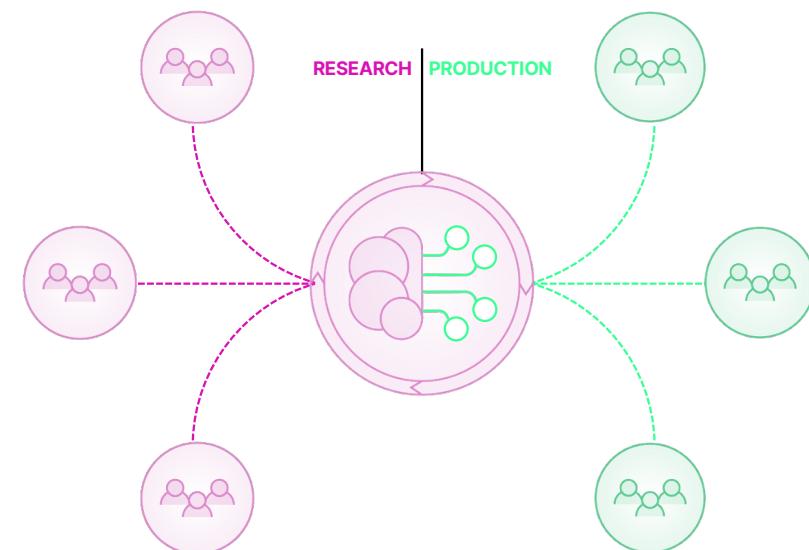
Technical debt is intentional or accidental decisions that make code difficult to understand, maintain, extend and fix. Much like a loan, you pay a higher cost later, because it decreases the team's agility as the project matures.

How do we *usually* pay off this technical debt?

Data scientists handover code to data & machine-learning engineers

“Part of the context is that I may fall slightly on the less technical side, so I have been mainly working out of notebooks and relying on data engineers to productionalise code.”

DATA SCIENTIST AT A MAJOR PHARMACEUTICAL COMPANY



Data Scientists

Anyone that gets to create data science experiments in the research phase

Data & ML Engineers

Anyone that must overhaul data science experiments produced in the research phase

The challenges of creating machine learning products

A workflow beyond notebooks still has challenges

“Data scientists have to learn so many tools to create high-quality code.”

“I have to think about Sphinx, flake8, isort, black, Cookiecutter Data Science, Docker, Python Logging, virtual environments, Pytest, configuration and more .”

“I spend a lot of time trying to understand a codebase that I didn’t write.”

“It’s tedious to always setup documentation and code quality tooling my project.”

“Everyone works in different ways.”

“It takes really long to put code in production and we have to rewrite and restructure large parts of it.”

“My code will not run on another person’s machine.”

“No one wants to use the framework I created.”

“We all have different levels of exposure to software engineering best-practice.”

“It’s an approach or process that’s understood to help build software that’s **superior in terms of **speed of execution**, **shipping with higher quality**, or building **more maintainable code**.”**

GERGELY OROSZ,
AUTHOR OF THE PRAGMATIC ENGINEER

02

Intro to Kedro



Kedro

An open-source Python toolbox that applies software engineering principles to data science code, making it easier to transition from prototype to production.

FOUNDED IN

2017

STATUS



DLF AI
& DATA

kedro

Benefits

Reduces the time spent rewriting data science experiments so that they are fit for production.

Encourage harmonious team collaboration and improve productivity.

Upskills all collaborators on how to apply software engineering principles to data science code.

+10,000
GITHUB STARS

+500,000
MONTHLY DOWNLOADS

+29,000,000
PIPELINE RUNS IN 2023

Enterprise adoption



Johnson & Johnson



Rabobank

Telkomsel

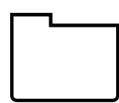


Developed by



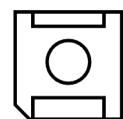
We built Kedro to reduce technical debt in data science experiments, making an easier transition from experimentation to production

What does Kedro give you?



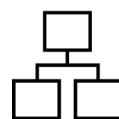
Project Template

Inspired by
Cookiecutter Data
Science



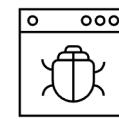
Data Catalog

Our core declarative
IO abstraction layer



Nodes + Pipelines

Constructs which
enable data-centric
workflows



Run status visualisation

Constructs to facilitate
debugging



Extensibility

Inherit, hook in or
plug-in to make Kedro
work for you

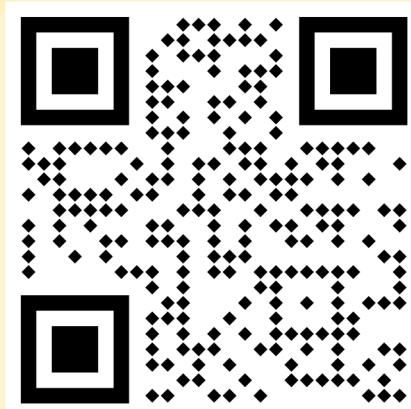




03

The workshop project

<https://github.com/pyladiesams/kedro-prod-ready-ds-pipelines-aug2025>



Example project

Scenario

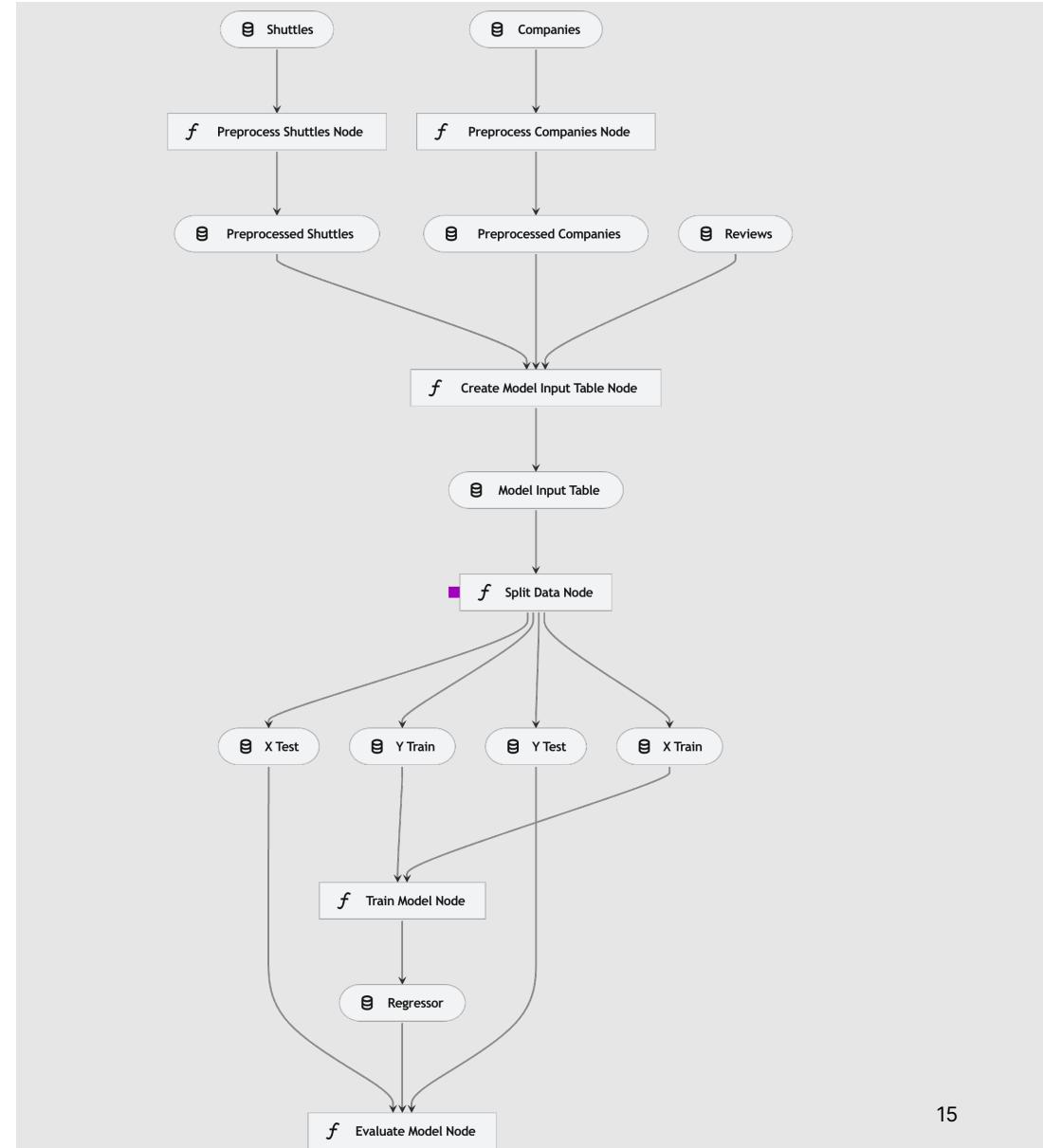
The year is 2160 and the space tourism industry is booming.

There are thousands of space shuttle companies taking tourists to the Moon and back.

You have been able to source amenities offered in each space shuttle, customer reviews and company information.

Goal

Build a model to estimate the price of a return trip.



04

Modularity

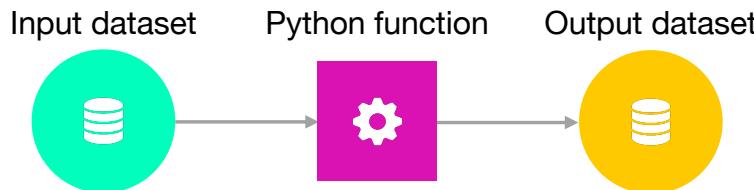
Modularity

- Breaking down a complex software system into **smaller, independent** modules.
- Each module is **responsible** for a specific function or feature and **operates independently**.
- Modules should be **loosely coupled**, meaning that they should not depend on each other too much.
- Enhanced **readability** and **understandability** of the system
- Allows for **reusability**
- Easier **collaboration**
- Makes it easier to **test** smaller chunks, which can be faster and help find bugs

Nodes & Pipelines

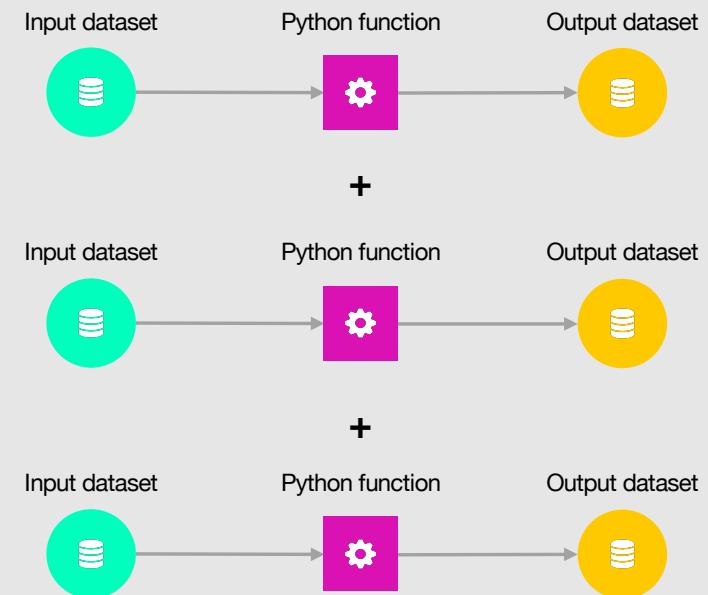
What is a node?

- A pure Python function that has an input and an output.
- Node definition supports multiple inputs for things like table joins and multiple outputs for things like producing a train/test split.



What is a pipeline?

- It is a directed acyclic graph.
- A collection of nodes with defined relationships and dependencies.



Example data science project

Before refactoring

```
import pandas as pd

# Load data
companies = pd.read_csv("../data/01_raw/companies.csv")
shuttles = pd.read_excel("../data/01_raw/shuttles.xlsx")
reviews = pd.read_csv("../data/01_raw/reviews.csv")

# Preprocess data
companies['iata_approved'] = companies['iata_approved'].astype(bool)
companies['company_rating'] = companies['company_rating'].str.replace('%', '').astype(float) / 100

shuttles["d_check_complete"] = shuttles["d_check_complete"].astype(bool)
shuttles["moon_clearance_complete"] = shuttles["moon_clearance_complete"].astype(bool)
shuttles["price"] = shuttles["price"].str.replace("$", "").str.replace(",","").astype(float)

# Create model input table
rated_shuttles = shuttles.merge(reviews, left_on="id", right_on="shuttle_id")
rated_shuttles = rated_shuttles.drop("id", axis=1)
model_input_table = rated_shuttles.merge(companies, left_on="company_id", right_on="id")
model_input_table = model_input_table.dropna()

from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

# Split data
features = ["engines", "passenger_capacity", "crew", "d_check_complete", "moon_clearance_complete", "iata_approved", "company_rating",
            "review_scores_rating"]
X = model_input_table[features]
y = model_input_table["price"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3)

# Train model
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Evaluate model
y_pred = regressor.predict(X_test)
score = r2_score(y_test, y_pred)
score
```

Example with modularity

Code separated into functions

```
# Function to load data
def load_data(companies_path, shuttles_path, reviews_path):
    companies = pd.read_csv(companies_path)
    shuttles = pd.read_excel(shuttles_path)
    reviews = pd.read_csv(reviews_path)
    return companies, shuttles, reviews

# Function to preprocess companies data
def preprocess_companies(companies):
    companies['iata_approved'] = companies['iata_approved'].astype(bool)
    companies['company_rating'] = companies['company_rating'].str.replace('%', '').astype(float) / 100
    return companies

# Function to preprocess shuttles data
def preprocess_shuttles(shuttles):
    shuttles["d_check_complete"] = shuttles["d_check_complete"].astype(bool)
    shuttles["moon_clearance_complete"] = shuttles["moon_clearance_complete"].astype(bool)
    shuttles["price"] = shuttles["price"].str.replace("$", "").str.replace(", ", "").astype(float)
    return shuttles

# Function to create the model input table
def create_model_input_table(shuttles, reviews, companies):
    rated_shuttles = shuttles.merge(reviews, left_on="id", right_on="shuttle_id")
    rated_shuttles = rated_shuttles.drop("id", axis=1)
    model_input_table = rated_shuttles.merge(companies, left_on="company_id", right_on="id")
    model_input_table = model_input_table.dropna()
    return model_input_table

# Function to split the data
def split_data(model_input_table, features, target, test_size=0.2, random_state=3):
    X = model_input_table[features]
    y = model_input_table[target]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=random_state)
    return X_train, X_test, y_train, y_test

# Function to train the model
def train_model(X_train, y_train):
    regressor = LinearRegression()
    regressor.fit(X_train, y_train)
    return regressor

# Function to evaluate the model
def evaluate_model(regressor, X_test, y_test):
    y_pred = regressor.predict(X_test)
    score = r2_score(y_test, y_pred)
    return score
```

Example with modularity

Main method that calls all functions

```
# Main function to orchestrate the workflow
def main():
    # Load data
    companies, shuttles, reviews = load_data("../data/01_raw/companies.csv",
                                              "../data/01_raw/shuttles.xlsx",
                                              "../data/01_raw/reviews.csv")

    # Preprocess data
    companies = preprocess_companies(companies)
    shuttles = preprocess_shuttles(shuttles)

    # Create model input table
    model_input_table = create_model_input_table(shuttles, reviews, companies)

    # Define features and target
    features = ["engines", "passenger_capacity", "crew", "d_check_complete", "moon_clearance_complete",
                "iata_approved", "company_rating", "review_scores_rating"]
    target = "price"

    # Split data
    X_train, X_test, y_train, y_test = split_data(model_input_table, features, target)

    # Train model
    regressor = train_model(X_train, y_train)

    # Evaluate model
    score = evaluate_model(regressor, X_test, y_test)

    print(f"Model R^2 score: {score}")

# Run the main function
if __name__ == "__main__":
    main()
```

Example with modularity in Kedro

Using nodes and pipeline to structure the functions

```
● ● ● data_processing/nodes.py
import pandas as pd

# Function to load data
def load_data(companies_path, shuttles_path, reviews_path):
    companies = pd.read_csv(companies_path)
    shuttles = pd.read_excel(shuttles_path)
    reviews = pd.read_csv(reviews_path)
    return companies, shuttles, reviews

# Function to preprocess companies data
def preprocess_companies(companies):
    companies['iata_approved'] = companies['iata_approved'].astype(bool)
    companies['company_rating'] = companies['company_rating'].str.replace('%', '').astype(float) / 100
    return companies

# Function to preprocess shuttles data
def preprocess_shuttles(shuttles):
    shuttles["d_check_complete"] = shuttles["d_check_"]
    shuttles["moon_clearance_complete"] = shuttles["m"
    shuttles["price"] = shuttles["price"].str.replace
    return shuttles

# Function to create the model input table
def create_model_input_table(shuttles, reviews, compa
    rated_shuttles = shuttles.merge(reviews, left_on=
    rated_shuttles = rated_shuttles.drop("id", axis=1)
    model_input_table = rated_shuttles.merge(companie
    model_input_table = model_input_table.dropna()
    return model_input_table

● ● ● data_processing/pipeline.py
from kedro.pipeline import Pipeline, node, pipeline

from .nodes import create_model_input_table, preprocess_companies, preprocess_shuttles,
load_data

def create_pipeline(**kwargs) -> Pipeline:
    return pipeline(
        [
            node(
                func=load_data,
                inputs=["companies_path", "shuttles_path", "reviews_path"],
                outputs=[ "companies", "shuttles", "reviews" ],
                name="load_data_node",
            ),
            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",
            ),
            node(
                func=create_model_input_table,
                inputs=[ "preprocessed_shuttles", "preprocessed_companies", "reviews" ],
                outputs="model_input_table",
                name="create_model_input_table_node",
            ),
        ]
    )
```

Example with modularity in Kedro

Using nodes and pipeline to structure the functions

```
data_science/nodes.py
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

# Function to split the data
def split_data(model_input_table, features, target, test_size=0.2, random_state=3):
    X = model_input_table[features]
    y = model_input_table[target]
    X_train, X_test, y_train, y_test =
        train_test_split(X, y, test_size=test_size, random_state=random_state)
    return X_train, X_test, y_train, y_test

# Function to train the model
def train_model(X_train, y_train):
    regressor = LinearRegression()
    regressor.fit(X_train, y_train)
    return regressor

# Function to evaluate the model
def evaluate_model(regressor, X_test, y_):
    y_pred = regressor.predict(X_test)
    score = r2_score(y_test, y_pred)
    return score
```

```
data_science/pipeline.py
```

```
from kedro.pipeline import Pipeline, node, pipeline

from .nodes import evaluate_model, split_data, train_model

def create_pipeline(**kwargs) -> Pipeline:
    return pipeline(
        [
            node(
                func=split_data,
                inputs=["model_input_table", "features", "target", "test_size", "random_state"],
                outputs=[ "X_train", "X_test", "y_train", "y_test" ],
                name="split_data_node",
            ),
            node(
                func=train_model,
                inputs=[ "X_train", "y_train" ],
                outputs="regressor",
                name="train_model_node",
            ),
            node(
                func=evaluate_model,
                inputs=[ "regressor", "X_test", "y_test" ],
                outputs=None,
                name="evaluate_model_node",
            ),
        ],
    )
```

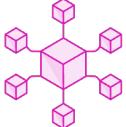
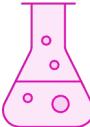
05

Separation of concerns

Separation of concerns

- Different parts of software system should be designed to address different concerns
 - E.g. separate user interaction layer, data access layer and business logic layer
- Separation of concerns helps to improve the readability, understandability, and maintainability of software systems.
- It also makes it easier to change and evolve the system over time.

Configuration

	Parameters		Hyperparameters		Experiment	What is configuration? <ul style="list-style-type: none">“Settings” for your machine-learning codeA way to define requirements for data, logging and parameters in different environmentsHelps keep credentials out of your code baseKeep all parameters in one place
	Location of Input Data		Credentials		Location of Output Results	What does configuration help you do? <ul style="list-style-type: none">Machine learning code that transitions from prototype to production with little effortMakes it possible to write generalisable and reusable analytics code that does not require significant modification to be used
Paths to files and directories, so the code knows where to find the data it needs. Also specify required columns or filters.	Connection information for databases and other data sources, such as server address and credentials.	Output settings, such as the directory for saving the results, file format, and level of verbosity.	 EXPERIMENT OR MODEL  DATA ACCESS			

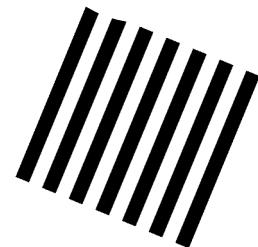
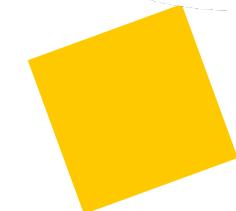
Data catalog

What is the Catalog?

- Manages the **loading and saving** of your data
- Available as a code or YAML API
- Versioning is available for file-based systems every time the pipeline runs

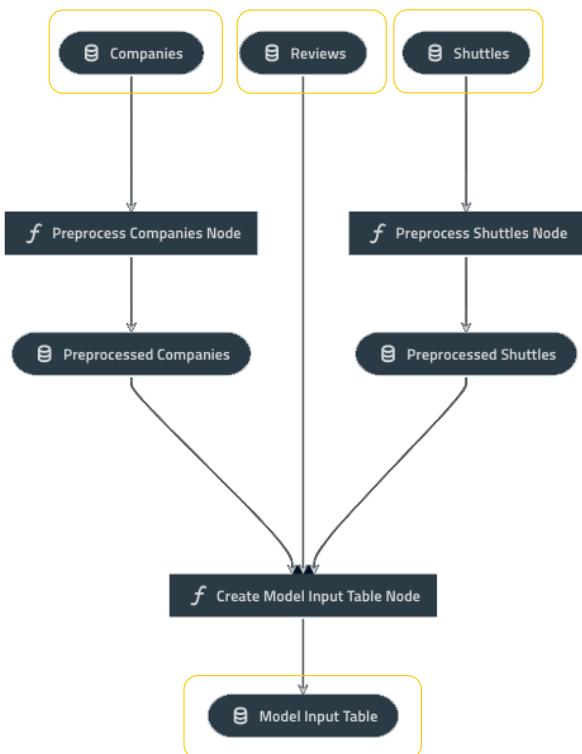
What does the Catalog help you do?

- Never write a single line of code that would read or write to a file, database or storage system
- Makes it possible to write **generalisable and reusable** analytics code that does not require significant modification to be used
- Access data without leaking credentials



Kedro connects datasets and nodes through pipelines

Nodes are normal Python functions and datasets are declared in YAML



```
# catalog.yml
```

```
companies
  type: pandas.CSVDataset
  filepath: data/01_raw/companies.csv

shuttles
  type: pandas.ExcelDataset
  filepath: data/01_raw/shuttles.xlsx

reviews
  type: pandas.CSVDataset
  filepath: data/01_raw/reviews.csv

model_input_table
  type: pandas.ParquetDataset
  filepath: s3://my_bucket/model_input_table.pq
  versioned: true
```

```
# pipelines.py
```

```
def create_pipeline(**kwargs):
    return pipeline([
        node(
            func=preprocess_companies,
            inputs="companies",
            outputs="preprocessed_companies",
        ),
        node(
            func=preprocess_shuttles,
            inputs="shuttles",
            outputs="preprocessed_shuttles",
        ),
        node(
            func=create_model_input_table,
            inputs=[
                "preprocessed_shuttles",
                "preprocessed_companies",
                "reviews"
            ],
            outputs="model_input_table",
        ),
    ])
```

Example data science project

Before refactoring

```
import pandas as pd

# Load data
companies = pd.read_csv("../data/01_raw/companies.csv")
shuttles = pd.read_excel("../data/01_raw/shuttles.xlsx")
reviews = pd.read_csv("../data/01_raw/reviews.csv")

# Preprocess data
companies['iata_approved'] = companies['iata_approved'].astype(bool)
companies['company_rating'] = companies['company_rating'].str.replace('%', '').astype(float) / 100

shuttles["d_check_complete"] = shuttles["d_check_complete"].astype(bool)
shuttles["moon_clearance_complete"] = shuttles["moon_clearance_complete"].astype(bool)
shuttles["price"] = shuttles["price"].str.replace("$", "").str.replace(", ", "").astype(float)

# Create model input table
rated_shuttles = shuttles.merge(reviews, left_on="id", right_on="shuttle_id")
rated_shuttles = rated_shuttles.drop("id", axis=1)
model_input_table = rated_shuttles.merge(companies, left_on="company_id", right_on="id")
model_input_table = model_input_table.dropna()

from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

# Split data
features = ["engines", "passenger_capacity", "crew", "d_check_complete", "moon_clearance_complete", "iata_approved", "company_rating",
            "review_scores_rating"]
X = model_input_table[features]
y = model_input_table["price"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3)

# Train model
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Evaluate model
y_pred = regressor.predict(X_test)
score = r2_score(y_test, y_pred)
score
```

Separate data I/O and configuration

Put I/O information and parameters in configuration

```
catalog.yml
```

```
companies:
  type: pandas.CSVDataset
  filepath: data/01_raw/companies.csv

reviews:
  type: pandas.CSVDataset
  filepath: data/01_raw/reviews.csv

shuttles:
  type: pandas.ExcelDataset
  filepath: data/01_raw/shuttles.xlsx
  load_args:
    engine: openpyxl
```

```
parameters.yml
```

```
model_options:
  test_size: 0.2
  random_state: 3
features:
  - engines
  - passenger_capacity
  - crew
  - d_check_complete
  - moon_clearance_complete
  - iata_approved
  - company_rating
  - review_scores_rating
```

Data I/O handled by the DataCatalog

No longer need code to read or save data

```
data_processing/nodes.py
import pandas as pd

# Function to load data
def load_data(companies_path, shuttles_path, reviews_path):
    companies = pd.read_csv(companies_path)
    shuttles = pd.read_excel(shuttles_path)
    reviews = pd.read_csv(reviews_path)
    return companies, shuttles, reviews

# Function to preprocess companies data
def preprocess_companies(companies):
    companies['iata_approved'] = companies['iata_approved'].astype(bool)
    companies['company_rating'] = companies['company_rating'].str.replace(' ', '')
    return companies

# Function to preprocess shuttles data
def preprocess_shuttles(shuttles):
    shuttles["d_check_complete"] = shuttles["d_check"].str.replace(' ', '')
    shuttles["moon_clearance_complete"] = shuttles["moon_clearance"].str.replace(' ', '')
    shuttles["price"] = shuttles["price"].str.replace(' ', '')
    return shuttles

# Function to create the model input table
def create_model_input_table(shuttles, reviews, companies):
    rated_shuttles = shuttles.merge(reviews, left_on='id', right_on='shuttle_id')
    rated_shuttles = rated_shuttles.drop("id", axis=1)
    model_input_table = rated_shuttles.merge(companies)
    model_input_table = model_input_table.dropna()
    return model_input_table

data_processing/pipeline.py
from kedro.pipeline import Pipeline, node, pipeline

from .nodes import create_model_input_table, preprocess_companies, preprocess_shuttles, load_data

def create_pipeline(**kwargs) -> Pipeline:
    return pipeline(
        [
            node(
                func=load_data,
                inputs=["companies_path", "shuttles_path", "reviews_path"],
                outputs=["companies", "shuttles", "reviews"],
                name="load_data_node",
            ),
            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",
            ),
            node(
                func=create_model_input_table,
                inputs=["preprocessed_shuttles", "preprocessed_companies", "reviews"],
                outputs="model_input_table",
                name="create_model_input_table_node",
            ),
        ],
    )
```

Data I/O handled by the DataCatalog

No longer need code to read or save data

```
import pandas as pd

# Function to load data
def load_data(companies_path, shuttles_path, reviews_path):
    companies = pd.read_csv(companies_path)
    shuttles = pd.read_excel(shuttles_path)
    reviews = pd.read_csv(reviews_path)
    return companies, shuttles, reviews

# Function to preprocess companies data
def preprocess_companies(companies):
    companies['iata_approved'] = companies['iata_approved'].astype(bool)
    companies['company_rating'] = companies['company'].str.replace(' ', '_')
    return companies

# Function to preprocess shuttles data
def preprocess_shuttles(shuttles):
    shuttles["d_check_complete"] = shuttles["d_check"]
    shuttles["moon_clearance_complete"] = shuttles["moon_clearance"]
    shuttles["price"] = shuttles["price"].str.replace('$', '')
    return shuttles

# Function to create the model input table
def create_model_input_table(shuttles, reviews, companies):
    rated_shuttles = shuttles.merge(reviews, left_on='id', right_on='shuttle_id')
    rated_shuttles = rated_shuttles.drop("id", axis=1)
    model_input_table = rated_shuttles.merge(companies, left_on='company', right_on='name')
    model_input_table = model_input_table.dropna()
    return model_input_table
```

```
from kedro.pipeline import Pipeline, node, pipeline

from .nodes import create_model_input_table, preprocess_companies, load_data

def create_pipeline(**kwargs) -> Pipeline:
    return pipeline(
        [
            node(
                func=load_data,
                inputs=["companies_path", "shuttles_path", "reviews_path"],
                outputs=["companies", "shuttles", "reviews"],
                name="load_data_node",
            ),
            node(
                func=create_model_input_table,
                inputs=["shuttles", "reviews", "companies"],
                outputs=["model_input_table"],
                name="create_model_input_table_node",
            ),
            node(
                func=preprocess_companies,
                inputs=["model_input_table"],
                outputs=["preprocessed_companies"],
                name="preprocess_companies_node",
            ),
            node(
                func=preprocess_shuttles,
                inputs=["model_input_table"],
                outputs=["preprocessed_shuttles"],
                name="preprocess_shuttles_node",
            ),
            node(
                func=create_model_input_table,
                inputs=["preprocessed_shuttles", "preprocessed_companies", "model_input_table"],
                outputs=["final_model_input_table"],
                name="final_create_model_input_table_node",
            ),
        ]
    )
```

```
from kedro.pipeline import Pipeline, node, pipeline

from .nodes import create_model_input_table, preprocess_companies, preprocess_shuttles, load_data

def create_pipeline(**kwargs) -> Pipeline:
    return pipeline(
        [
            node(
                func=load_data,
                inputs=["companies_path", "shuttles_path", "reviews_path"],
                outputs=["companies", "shuttles", "reviews"],
                name="load_data_node",
            ),
            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",
            ),
            node(
                func=create_model_input_table,
                inputs=["preprocessed_shuttles", "preprocessed_companies", "reviews"],
                outputs="model_input_table",
                name="create_model_input_table_node",
            ),
        ],
    )
```

“Settings” put into configuration

Use parameters from config

```
data_science/nodes.py

from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split

# Function to split the data
def split_data(model_input_table, features, target, test_size=0.2, random_state=3):
    X = model_input_table[features]
    y = model_input_table[target]
    X_train, X_test, y_train, y_test =
        train_test_split(X, y, test_size=test_size, random_state=random_state)
    return X_train, X_test, y_train, y_test

# Function to train the model
def train_model(X_train, y_train):
    regressor = LinearRegression()
    regressor.fit(X_train, y_train)
    return regressor

# Function to evaluate the model
def evaluate_model(regressor, X_test, y_:
    y_pred = regressor.predict(X_test)
    score = r2_score(y_test, y_pred)
    return score
```

```
data_science/pipeline.py

from kedro.pipeline import Pipeline, node, pipeline
from .nodes import evaluate_model, split_data, train_model

def create_pipeline(**kwargs) -> Pipeline:
    return pipeline(
        [
            node(
                func=split_data,
                inputs=["model_input_table", "features", "target", "test_size", "random_state"],
                outputs=["X_train", "X_test", "y_train", "y_test"],
                name="split_data_node",
            ),
            node(
                func=train_model,
                inputs=["X_train", "y_train"],
                outputs="regressor",
                name="train_model_node",
            ),
            node(
                func=evaluate_model,
                inputs=["regressor", "X_test", "y_test"],
                outputs=None,
                name="evaluate_model_node",
            ),
        ],
    )
```

“Settings” put into configuration

Use parameters from config

```
● ● ● data_science/nodes.py

# Function to split the data
def split_data(data: pd.DataFrame, parameters: Dict) -> Tuple:
    X = data[parameters["features"]]
    y = data["price"]
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=parameters["test_size"], random_state=parameters["random_state"]
    )
    return X_train, X_test, y_train, y_test

# Function to train the model
def train_model(X_train, y_train):
    regressor = LinearRegression()
    regressor.fit(X_train, y_train)
    return regressor

# Function to evaluate the model
def evaluate_model(regressor, X_test, y_test):
    y_pred = regressor.predict(X_test)
    score = r2_score(y_test, y_pred)
    return score
```

```
● ● ● data_science/pipeline.py

from kedro.pipeline import Pipeline, node, pipeline

from .nodes import evaluate_model, split_data, train_model

def create_pipeline(**kwargs) -> Pipeline:
    return pipeline(
        [
            node(
                func=split_data,
                inputs=["model_input_table", "params:model_options"],
                outputs=["X_train", "X_test", "y_train", "y_test"],
                name="split_data_node",
            ),
            node(
                func=train_model,
                inputs=["X_train", "y_train"],
                outputs="regressor",
                name="train_model_node",
            ),
            node(
                func=evaluate_model,
                inputs=["regressor", "X_test", "y_test"],
                outputs=None,
                name="evaluate_model_node",
            ),
        ],
    )
```



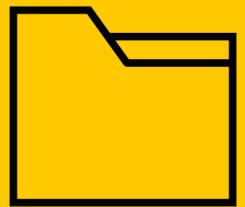
06

Maintainability

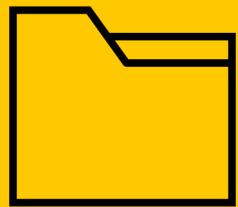
Maintainability

- A measure of how easy it is to **understand**, **modify**, and **improve** software.
- **Testability**: verifies that your code meets different requirements (functional, business, performance, reliability, etc..)
- **Readability**: linted, formatted and documented code
- **Findability**: structure and organisation of code

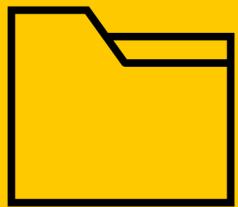
Project template



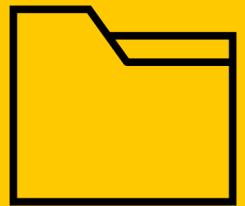
Configuration



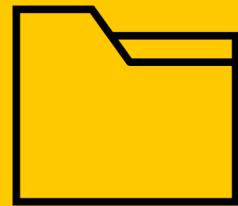
Notebooks



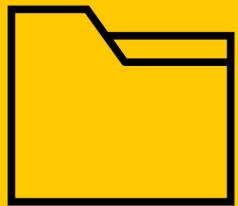
Data



Source code



Tests



Project Documentation

What is the project template?

- Standard and customisable [project structure](#)
- Built-in support for Python logging, Pytest for unit tests and Sphinx for documentation

What does the project template help you do?

- Establish [standardisation](#) across your organisation
- You spend less time digging around in previous projects for useful code
- Makes it easier for collaborators to work with you

Project creation options

Include linting, testing, logging and documentation setup

```
kedro new

Project Tools
=====
These optional tools can help you apply software engineering best practices.
To skip this step in future use --tools
To find out more: https://docs.kedro.org/en/stable/starters/new\_project\_tools.html

Tools
1) Lint: Basic linting with Ruff
2) Test: Basic testing with pytest
3) Log: Additional, environment-specific logging options
4) Docs: A Sphinx documentation setup
5) Data Folder: A folder structure for data management
6) PySpark: Configuration for working with PySpark
7) Kedro-Viz: Kedro's native visualisation tool

Which tools would you like to include in your project? [1-7/1,3/all/none]:
```

Example Kedro project structure

Including test setup & examples

The screenshot shows a code editor interface with two main panes. The left pane displays a project structure under a 'Project' heading. The right pane shows a Python test script named 'test_pipeline.py'.

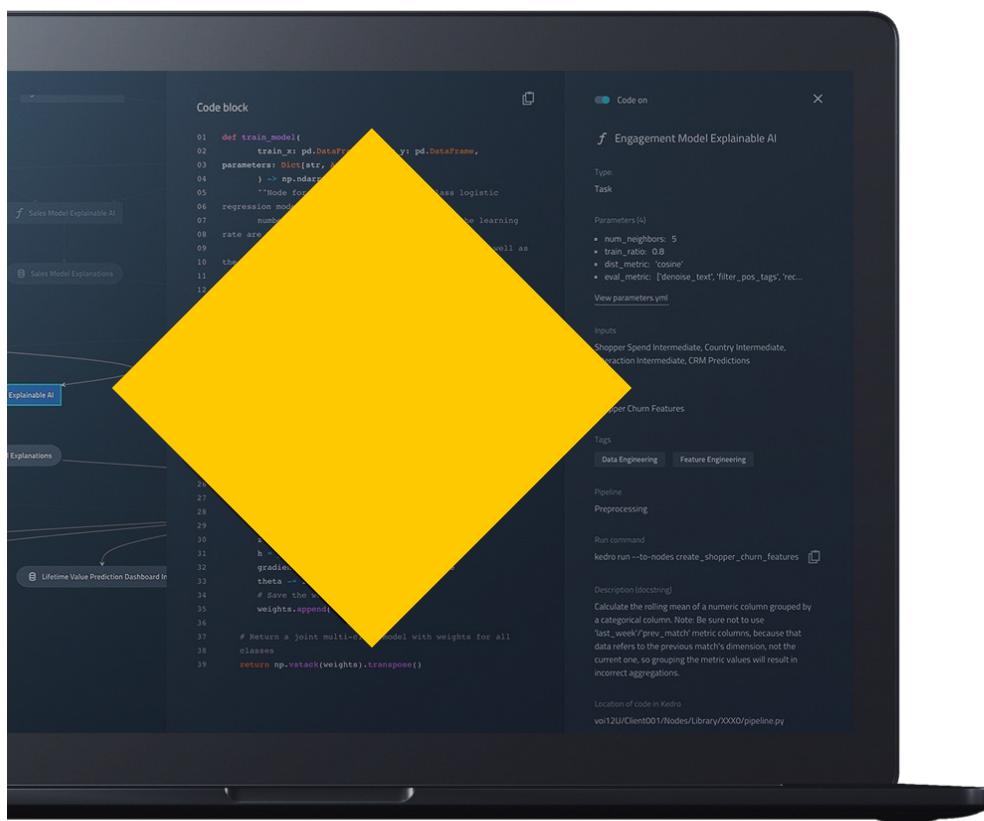
Project Structure:

- kickstart (~/Projects/Testing/kickstart)
 - .env
 - .viz
 - conf
 - base
 - catalog.yml
 - parameters.yml
 - parameters_data_processing.yml
 - parameters_data_science.yml
 - local
 - README.md
- data
 - 01_raw
 - 02_intermediate
 - 03_primary
 - 04_feature
 - 05_model_input
 - 06_models
 - 07_model_output
 - 08_reporting
- notebooks
- src
 - kickstart
 - pipelines
 - data_processing
 - __init__.py
 - nodes.py
 - pipeline.py
 - data_science
 - __init__.py
 - nodes.py
 - pipeline.py
 - __init__.py
 - main.py
 - pipeline_registry.py
 - settings.py
 - tests
 - pipelines
 - data_science
 - __init__.py
 - __init__.py
 - test_pipeline.py

Summary

- Software Engineering principles can elevate the quality and usability of your data science projects, even when experimental
- Kedro makes it easy to embed modularity, separation of concerns and maintainability
- It's possible to establish and embed best practices without any tools

Kedro Resources

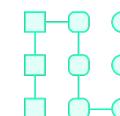


There are many resources to get started using Kedro; here is a list of resources you can access.



Convert existing work into a Kedro project

We support workflows that take you from Jupyter notebooks and glue code scripts into a Kedro project.
blog.kedro.org



View demo

Interact with pipeline visualisation and experiment tracking with Kedro-Viz.
demo.kedro.org



Learn Kedro

Walk through our spaceflights tutorial and get up to speed on beginner-to-intermediate functionality.
docs.kedro.org



Join the community

Join a friendly and growing community of Kedroids waiting for your questions.
slack.kedro.org

Q&A



Thank you!