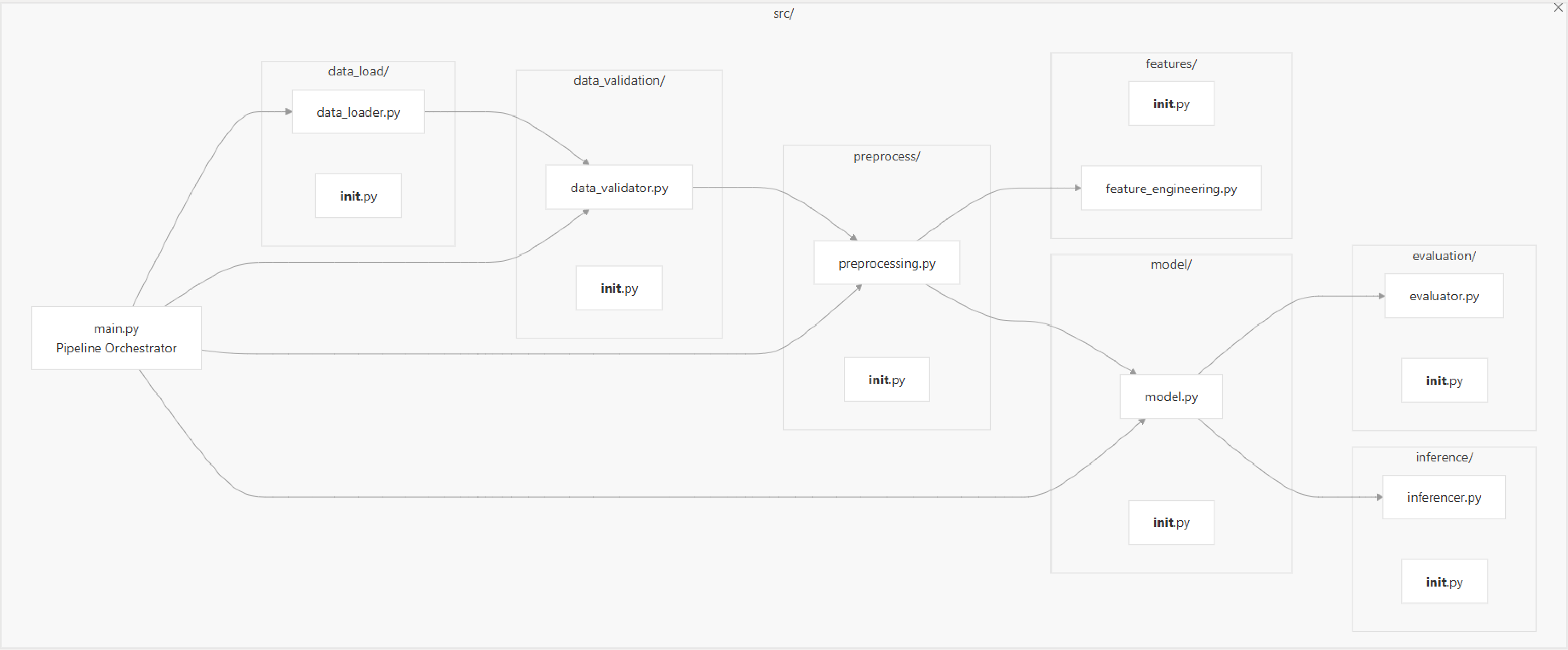
~15’ fast & furious project recap

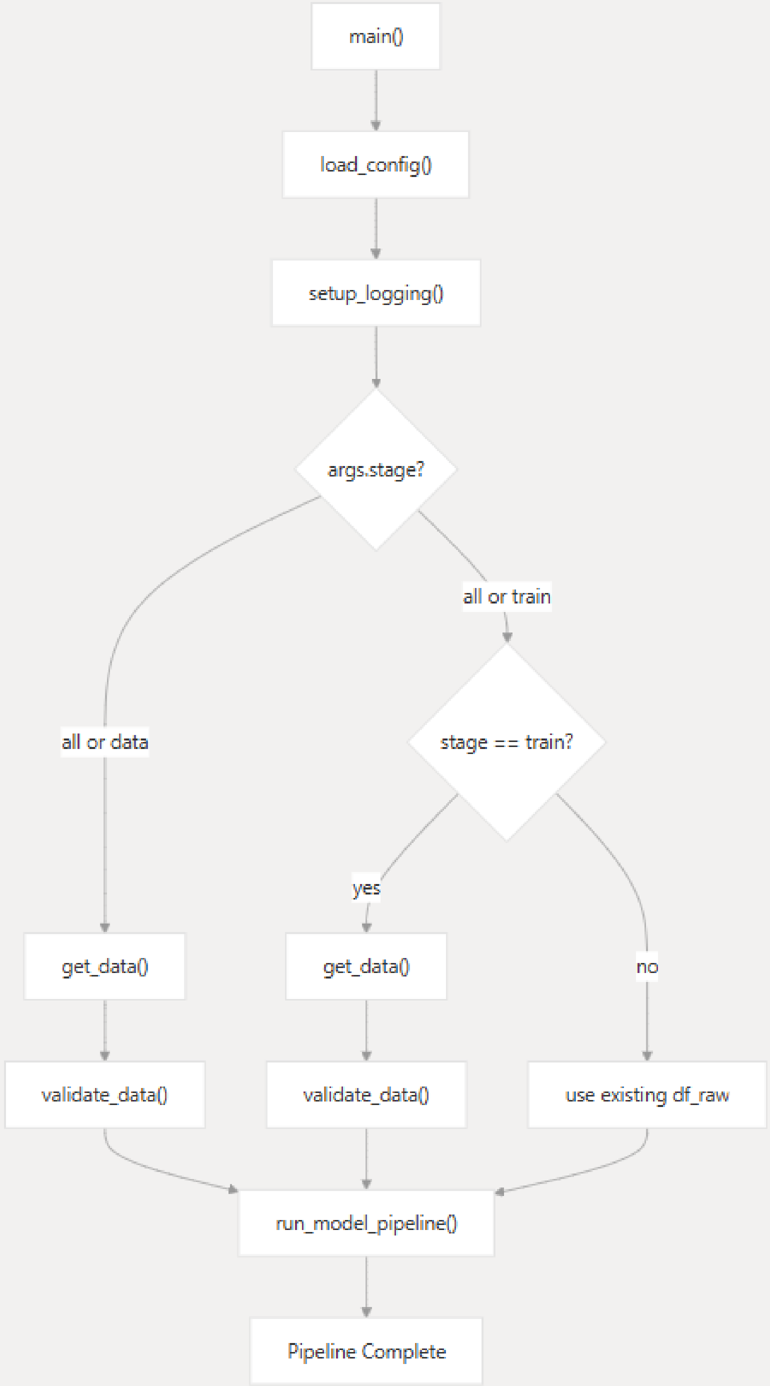
- High-level summary of the project e.g. README.md and deepwiki features
- Code-specific decisions and best practices

Structure code into modular packages to scale and accelerate great SW solutions development



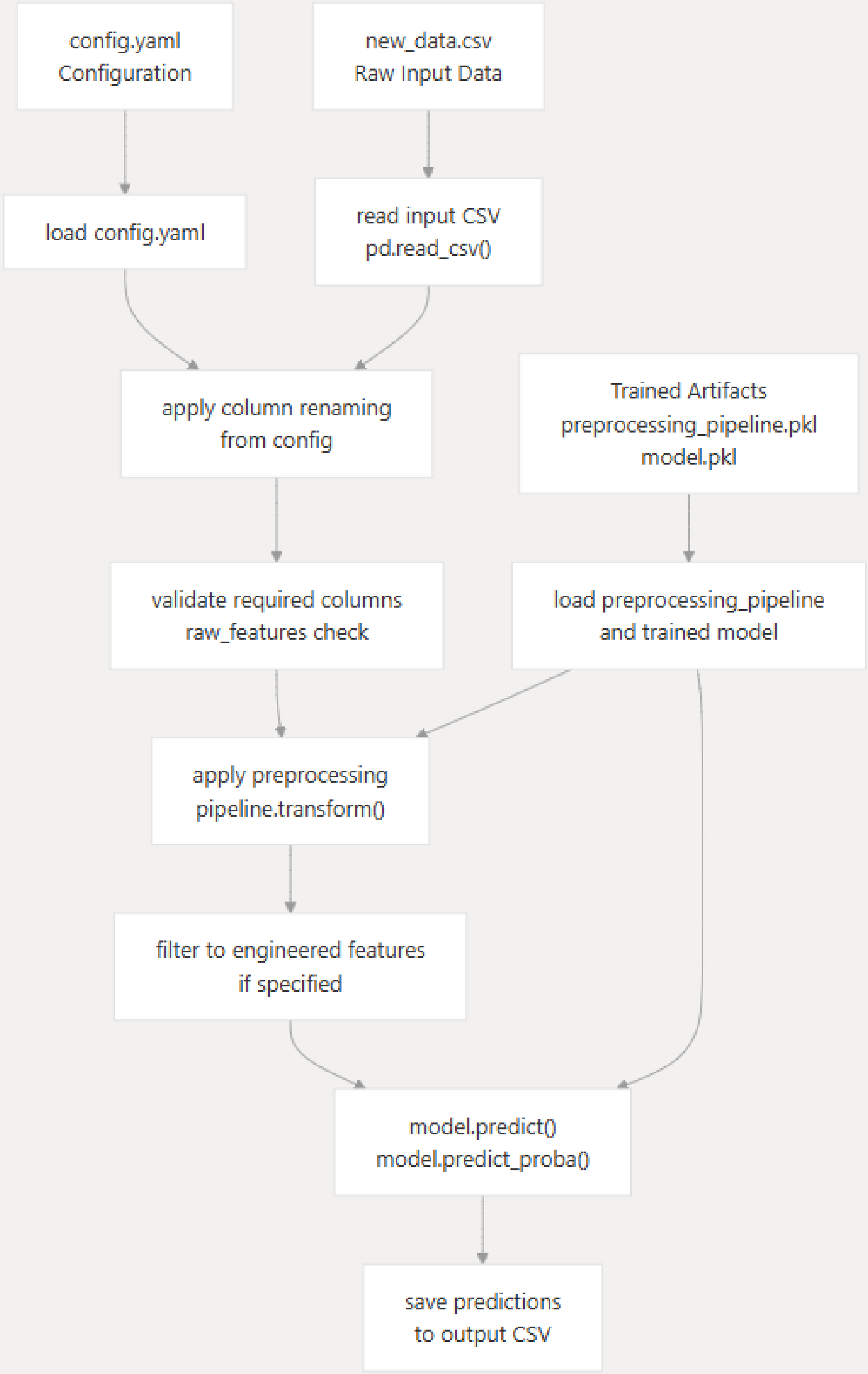
Source: deepwiki

Automate execution from a single-entry point to cut manual steps and ensure consistency end-to-end



Source: deepwiki

Separate inference from training to deploy faster, reuse components, and reduce operational risk



config.yaml
Configuration

new_data.csv
Raw Input Data

load config.yaml

read input CSV
pd.read_csv()

apply column renaming
from config

Trained Artifacts
preprocessing_pipeline.pkl
model.pkl

validate required columns
raw_features check

load preprocessing_pipeline
and trained model

apply preprocessing
pipeline.transform()

filter to engineered features
if specified

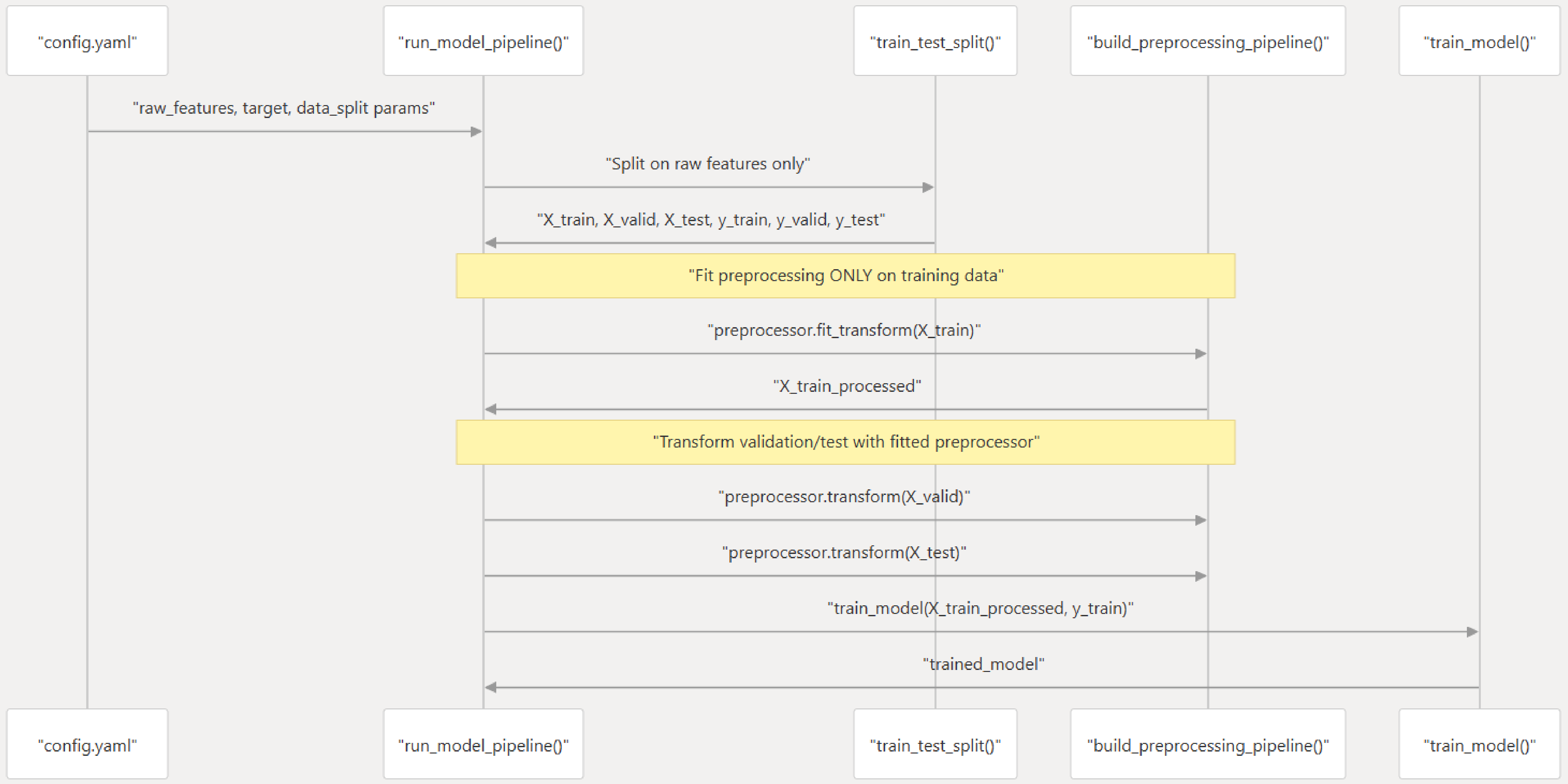
model.predict()
model.predict_proba()

save predictions
to output CSV

Source: deepwiki

Enforce strict data flow boundaries to prevent leakage and ensure valid model performance in production

Source: deepwiki



"config.yaml"

"run_model_pipeline()"

"train_test_split()"

"build_preprocessing_pipeline()"

"train_model()"

"raw_features, target, data_split params"

"Split on raw features only"

"X_train, X_valid, X_test, y_train, y_valid, y_test"

"Fit preprocessing ONLY on training data"

"preprocessor.fit_transform(X_train)"

"X_train_processed"

"Transform validation/test with fitted preprocessor"

"preprocessor.transform(X_valid)"

"preprocessor.transform(X_test)"

"train_model(X_train_processed, y_train)"

"trained_model"

"config.yaml"

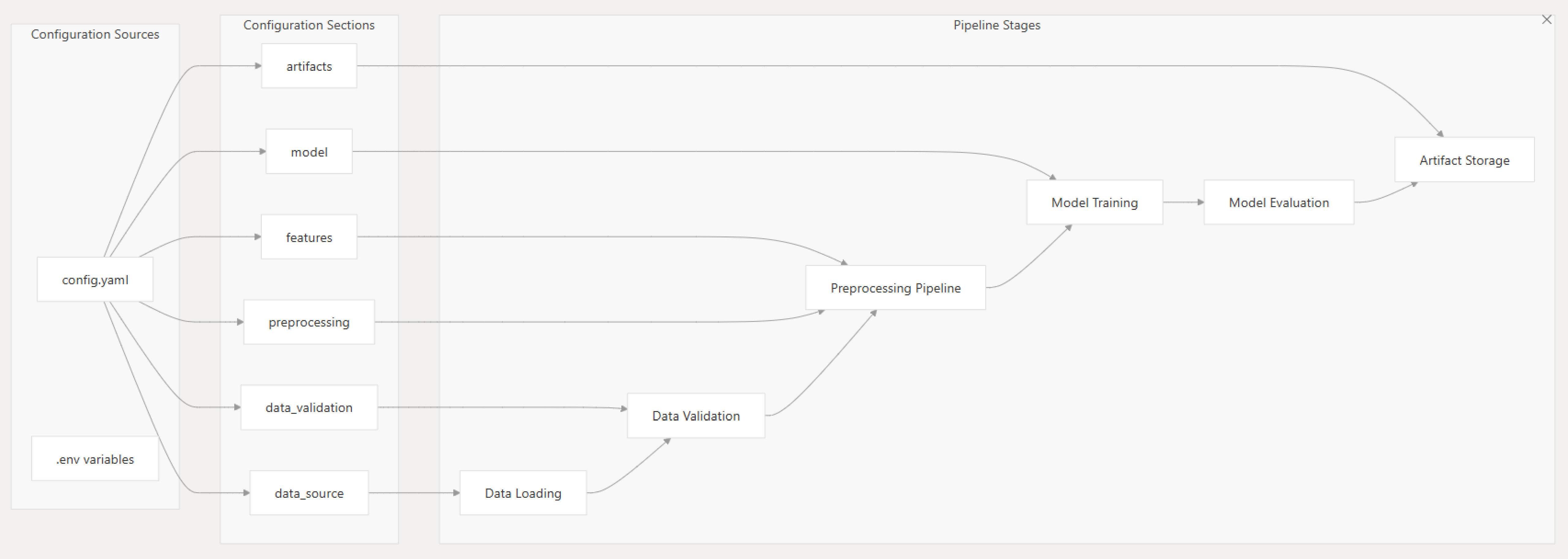
"run_model_pipeline()"

"train_test_split()"

"build_preprocessing_pipeline()"

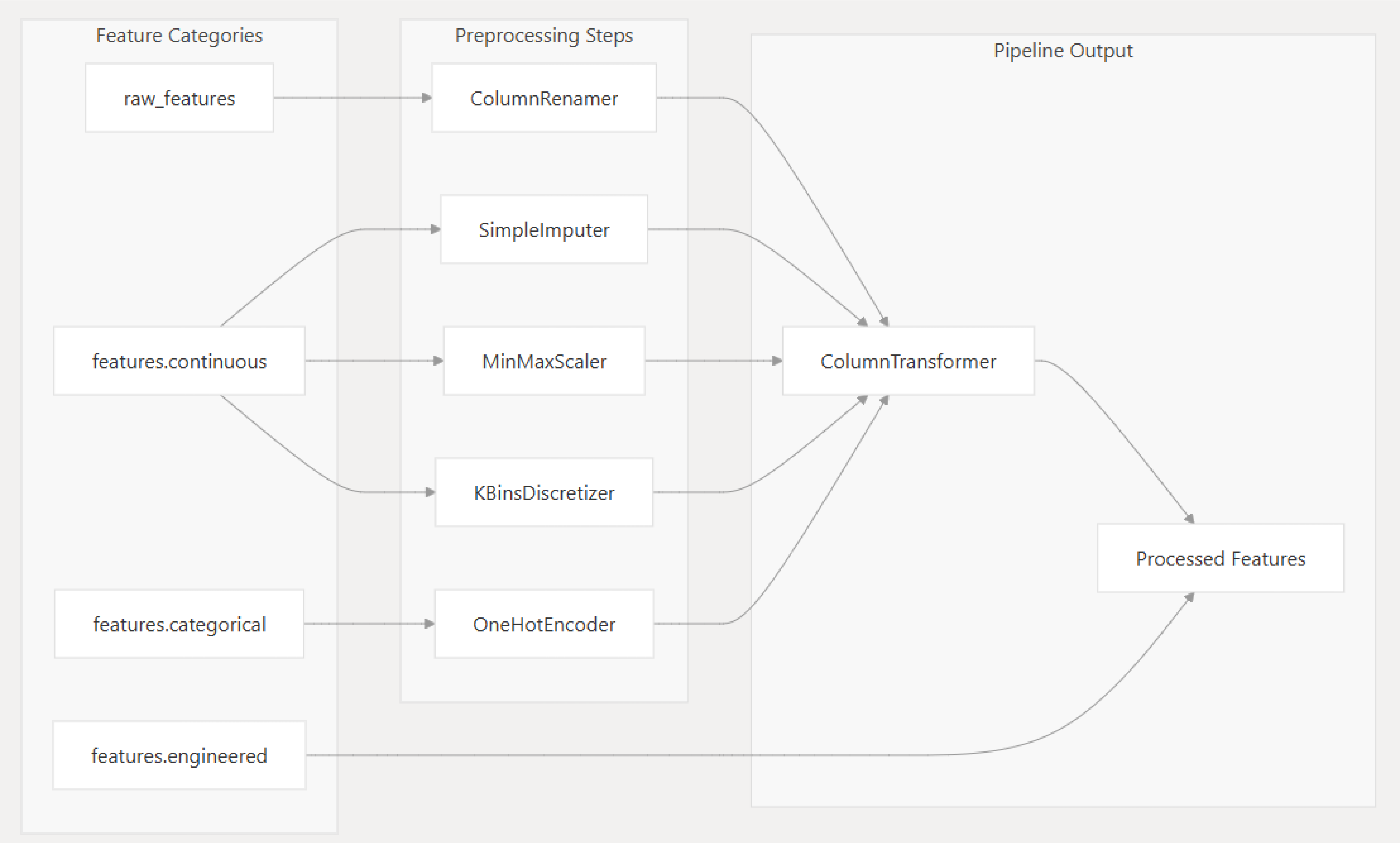
"train_model()"

Centralize config to reduce errors, speed up experiments, and adapt pipelines without touching the actual code



Source: deepwiki

Isolate feature eng. logic to improve reuse, trace, and model auditability across projects



Source: deepwiki

Some code level remarks and best practices

- *`Environment.yml`* is now frozen with current package versions
- *`README.md`* contains a comprehensive explanation of the project purpose and how to use it
- `Tests/` leverage pytest advanced feature to enhance your test suit dev e.g. @pytest.fixture, monkeypatch to prevent touching actual src's artifacts!
- `main.py` is greatly enhanced by adding parsers to execute different steps of our pipeline (specially "infer")
- `data_loader` enabling "standalone" runs enables you to each module's behaviour (enhanced by logs) *`(if __name__ == "__main__")`*
- `data_validation` is crafted as a checker for critical data entry points e.g. first ingestion AND inference
- `feature_eng` and `preprocessing` are easily managed thu SKlearn customed *`transformers`* - creating and integrating new features and ad hoc transformations
- For the `model` step, we **ONLY** fit the preprocessing pipeline to the train data, and transform validation and testing, and output respective artifacts `data splits, metrics.json, preprocessing_pipeline.pkl, model.pkl`
- `evaluator` saves a report for stakeholders' consumption and decisions making
- `inference` is called **ONLY** when serving new data. Via CLI as standalone, `main.py`, or a webapp