

Twitter Thread by Goku Mohandas



Goku Mohandas

@GokuMohandas



All the @madewithml machine learning fundamentals & MLOps lessons are released!

- ■ Project-based
- ■ Intuition & application (code)
- ■ 26K+ GitHub ■■
- ♥■ 30K+ community
- ■ 47 lessons, 100% open-source

<https://t.co/XlhD3wl1DA>

■ Thread on details & lesson highlights ■

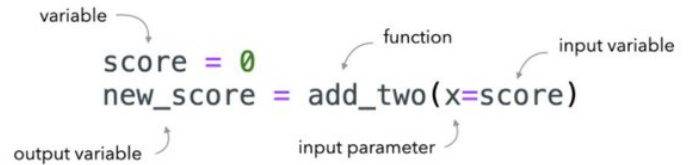
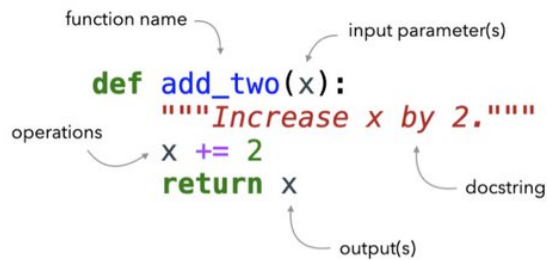
Who is this course for?

- ■ Software engineers / Data scientists looking to learn how to responsibly create ML systems.
- ■ College grads looking to learn the practical skills they'll need for the industry.
- ■ Product Managers who want to develop a technical foundation.

We start with lessons on the fundamentals of ML through intuitive explanations, clean code and visualizations.

■ Foundations

- Python (variables, functions, classes, decorators)
- NumPy (numerical analysis)
- Pandas (data analysis)
- PyTorch (operations, gradients)



```
1 # Define the function
2 def add_two(x):
3     """Increase x by 2."""
4     x += 2
5     return x
```

```
1 # Use the function
2 score = 0
3 new_score = add_two(x=score)
4 print(new_score)
```

2

Then we dive into implementing basic ML algorithms 1■ from scratch then 2■ in PyTorch. Starting from simple models → complex models.

■ Modeling

- Linear Regression
- Logistic Regression
- Neural Networks
- Data Quality (■■ very important)
- Utilities (for loading and training)

Step 4: Calculate the gradient of loss $J(\theta)$ w.r.t to the model weights.

$$J(\theta) = \frac{1}{N} \sum_i (y_i - \hat{y}_i)^2 = \frac{1}{N} \sum_i (y_i - X_i W)^2$$

$$\rightarrow \frac{\partial J}{\partial W} = -\frac{2}{N} \sum_i (y_i - X_i W) X_i = -\frac{2}{N} \sum_i (y_i - \hat{y}_i) X_i$$

$$\rightarrow \frac{\partial J}{\partial b} = -\frac{2}{N} \sum_i (y_i - X_i W) 1 = -\frac{2}{N} \sum_i (y_i - \hat{y}_i) 1$$

```
1 # Backpropagation
2 dW = -(2/N) * np.sum((y_train - y_pred) * X_train)
3 db = -(2/N) * np.sum((y_train - y_pred) * 1)
```

We wrap up the fundamentals by implementing deep learning algorithms in PyTorch.

- Deep Learning
 - CNNs
 - Embeddings
 - RNNs
 - Transformers

■ We motivate the need for specific architectures and additional complexity as we implement each method.

output channels =
filters =
50

input channels =
vocab size

*
(conv)

num_filters = 50

=

$$W_1 = \frac{W_2 - F + 2P}{S} + 1 = \frac{8 - 3 + 2(0)}{1} + 1 = 6$$

$$H_1 = \frac{H_2 - F + 2P}{S} + 1 = \frac{1 - 1 + 2(0)}{1} + 1 = 1$$

$$D_2 = D_1$$

Variable	Description
W	width of each input = 8
H	height of each input = 1
D	depth (# of channels)
F	filter size = 3
P	padding = 0
S	stride = 1

The first MLOps lessons are on the Product development and iteration cycle.

- Product
 - Identify the core objective.
 - Design a solution with constraints.
 - Evaluation strategies that avoid bias.
 - Iterate via feedback and motivate adding complexity.



Product

✗ Model fitter

naively maps a set of inputs to outputs.

obsesses on methods (SOTA, single metric, etc.)

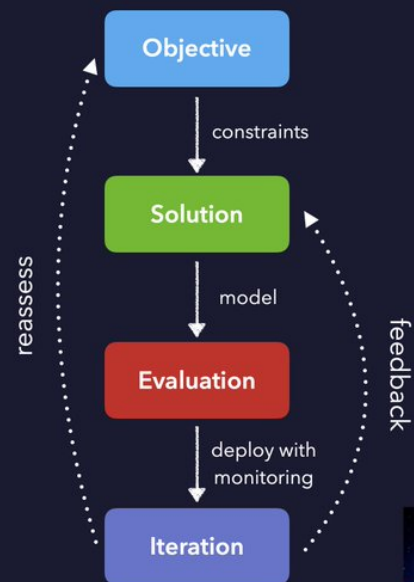
fitting methods are ephemeral.

✓ Problem solver

knows which set of inputs and outputs are worth mapping.

focuses on application (objective, constraints, evaluation, etc.)

foundational mental models are enduring.



Goku Mohandas

Next we dive into exploring and transforming our data.

■ Data

- Labeling (data worth modeling, active learning)
- Preprocessing (prepare + transform)
- Exploration (answering questions)
- Splitting (multi-label classification)
- Augmentation (nlpaug, transformation functions)

```
1 # Load tags
2 url = "https://raw.githubusercontent.com/GokuMohandas/MadeWithML/main/datasets"
3 tags = json.loads(urlopen(url).read())
4 tags_dict = {}
5 for item in tags:
6     key = item.pop("tag")
7     tags_dict[key] = item
8 print (f"{len(tags_dict)} tags")
```

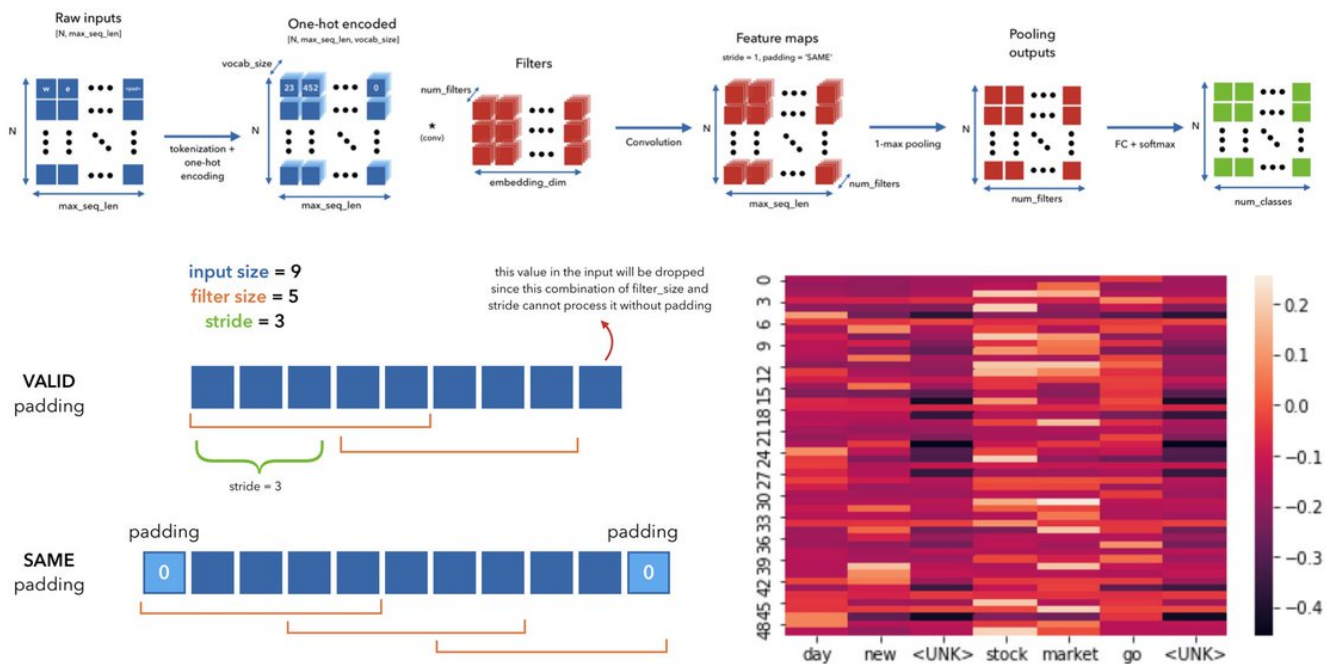
```
1 from snorkel.labeling import labeling_function
2
3 @labeling_function()
4 def contains_tensorflow(text):
5     condition = any(tag in text.lower() for tag in ("tensorflow", "tf"))
6     return "tensorflow" if condition else None
```

■ Modeling

- Baselines (simple → complex)
- Evaluation (overall, slices, generated)

- Experiment tracking (tracking, viewing and loading)
- Optimization (sampling + pruning)

■ These aren't just tutorial code snippets. We implement everything with clean and tested code.



Next, we move our work from notebooks to scripts.

■ Scripting

- Organization
- Packaging (setup + virtualenv)
- Documentation (auto)
- Logging (logger, handler, formatter)
- Styling (black, isort, flake8)
- Makefile

■ All of this makes for a very calm developing experience.

```
def pad_sequences(
    sequences: np.ndarray, max_seq_len: int = 0
) -> np.ndarray:
    """Zero pad sequences to a specified `max_seq_len`
    or to the length of the largest sequence in `sequences`.

    Usage:

    ```python
 # Pad inputs
 seq = np.array([[1, 2, 3], [1, 2]], dtype=object)
 padded_seq = pad_sequences(seq, max_seq_len=5)
 print (padded_seq)

 [[1. 2. 3. 0. 0.]
 [1. 2. 0. 0. 0.]]

 Note:
 Input `sequences` must be 2D.

 Args:
 sequences (np.ndarray): 2D array of data...
 max_seq_len (int, optional): Length to pad...

 Raises:
 ValueError: Input sequences are not two-dimensional.

 Returns:
 An array with the zero padded sequences.

 """
```

pad\_sequences(sequences, max\_seq\_len=0)

Zero pad sequences to a specified `max\_seq\_len` or to the length of the largest sequence in `sequences`.

Usage:

```
Pad inputs
seq = np.array([[1, 2, 3], [1, 2]], dtype=object)
padded_seq = pad_sequences(sequences=seq, max_seq_len=
print (padded_seq)
```

```
[[1. 2. 3. 0. 0.]
 [1. 2. 0. 0. 0.]]
```

Note

Input `sequences` must be 2D. Check out this [implementation](#) for a generalized approach.

Parameters:

Name	Type	Description	Default
sequences	ndarray	2D array of data to be padded.	required
max_seq_len	int	Length to pad sequences to. Defaults to 0.	0

Exceptions:

Now we're ready to wrap our application via various interfaces.

## ■ Interfaces

- Command-line (CLI)
- RESTful API with FastAPI (design, schemas, validation)

■ These interfaces allow us to quickly execute both internal (training, testing, etc.) and external (inference) tasks.

POST

/predict Predict

Predict tags for a list of texts using the best run.

Parameters

No parameters

Cancel

Request body required

application/json

```
{
 "texts": [
 {
 "text": "Transfer learning with transformers for self-supervised learning."
 },
 {
 "text": "Generative adversarial networks in both PyTorch and TensorFlow."
 }
]
}
```

Responses

Code	Details
200	<div>Response body</div> <pre>{   "message": "OK",   "method": "POST",   "status-code": 200,   "timestamp": "2021-03-30T11:21:42.012890",   "url": "http://localhost:5000/predict",   "data": {     "predictions": [       {         "input_text": "Transfer learning with transformers fo",         "preprocessed_text": "transfer learning transformers",         "predicted_tags": [           "attention",           "language-modeling",           "natural-language-processing",           "self-supervised-learning",           "transfer-learning",           "transformers"         ]       },       {         "input_text": "Generative adversarial",         "preprocessed_text": "Generative adversarial",         "predicted_tags": [           "attention",           "language-modeling",           "natural-language-processing",           "self-supervised-learning",           "transfer-learning",           "transformers"         ]       }     ]   } }</pre> <div>Download</div> <div>Response headers</div> <pre>content-length: 688 content-type: application/json date: Tue, 30 Mar 2021 18:21:34 GMT server: uvicorn</pre>

Throughout development, we've been testing not only our code but also our data and models.

## ■ Testing

- Test types, coverage, best practices


- Pytest fixtures, markers, parametrize
- Test data w/ Great Expectations
- Test models via slicing functions
- Behavioral tests

```
(venv) → applied-ml git:(main) x pytest --cov tagifai --cov app --cov-report html
===== test session starts =====
platform darwin -- Python 3.7.10, pytest-6.0.2, py-1.10.0, pluggy-0.13.1
rootdir: /Users/goku/Documents/madewithml/applied-ml, configfile: pyproject.toml, testpaths: tests
plugins: cov-2.10.1
collected 68 items

tests/app/test_api.py
tests/app/test_cli.py
tests/tagifai/test_config.py .
tests/tagifai/test_data.py
tests/tagifai/test_eval.py .
tests/tagifai/test_models.py ...
tests/tagifai/test_train.py .
tests/tagifai/test_utils.py

----- coverage: platform darwin, python 3.7.10-final-0 -----
Coverage HTML written to dir htmlcov

===== 68 passed, 10 warnings in 202.52s (0:03:22) =====
(venv) → applied-ml git:(main) x
```

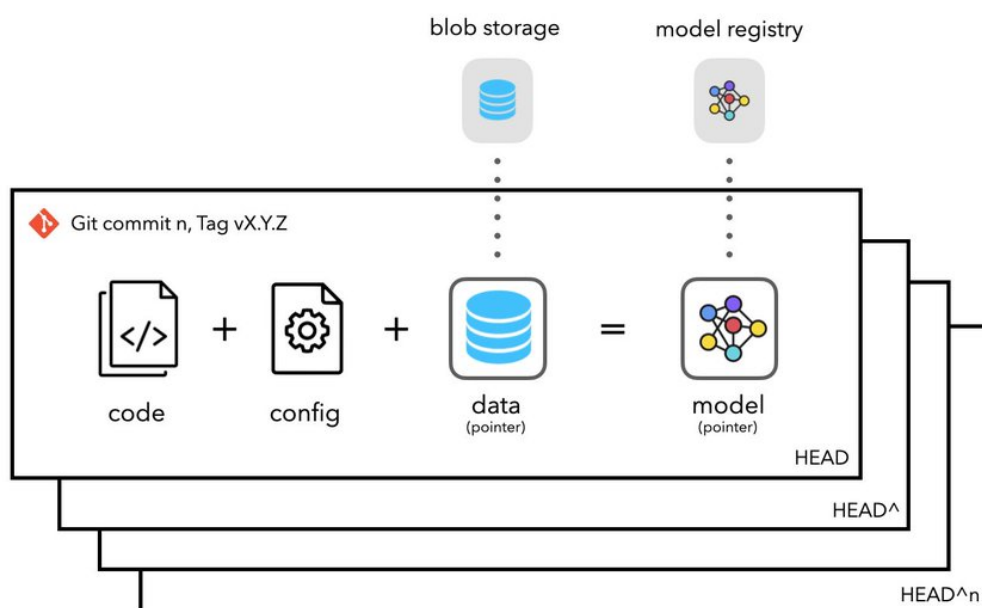


[ 11%]
[ 22%]
[ 23%]
[ 86%]
[ 88%]
[ 92%]
[ 94%]
[100%]

We want to ensure that our work is entirely reproducible by anyone.

## ■■ Reproducibility

- Git basics via workflows (dev, inspect, merge, etc.)
- Pre-commit hooks (+ custom local)
- Versioning code + config + data = models via DVC
- Containerization via Docker



Next, we want to be able to showcase our work and enable interaction via [@streamlit](#).

#### ■ Dashboard:

- Data: annotation, EDA, preprocessing
- Performance: overall, slices, regressions
- Inference: intermediate & final outputs
- Inspection: labeling (FP), weaknesses (FN)

### Annotation

We want to determine what the minimum tag frequency is so that we have enough samples per tag for training.



#### Most common tags:

('natural-language-processing', 424)

('computer-vision', 388)

('pytorch', 258)

('tensorflow', 213)

('transformers', 196)

#### Tags that just made the cut:

('streamlit', 27)

('exploratory-data-analysis', 27)

('graph-clustering', 27)

('graph-embedding', 26)

('semi-supervised-learning', 25)

#### Tags that just missed the cut:

('text-classification', 24)

('linear-regression', 24)

('graph-convolutional-networks', 23)

('named-entity-recognition', 23)

('classification', 22)

### Preprocessing

Input text

Generation via Autoencoders!

filters

[!'"#\$%&()\*+,-./:;<=>?@\[\]^\_`{|}~]

☒ lower

☐ stem

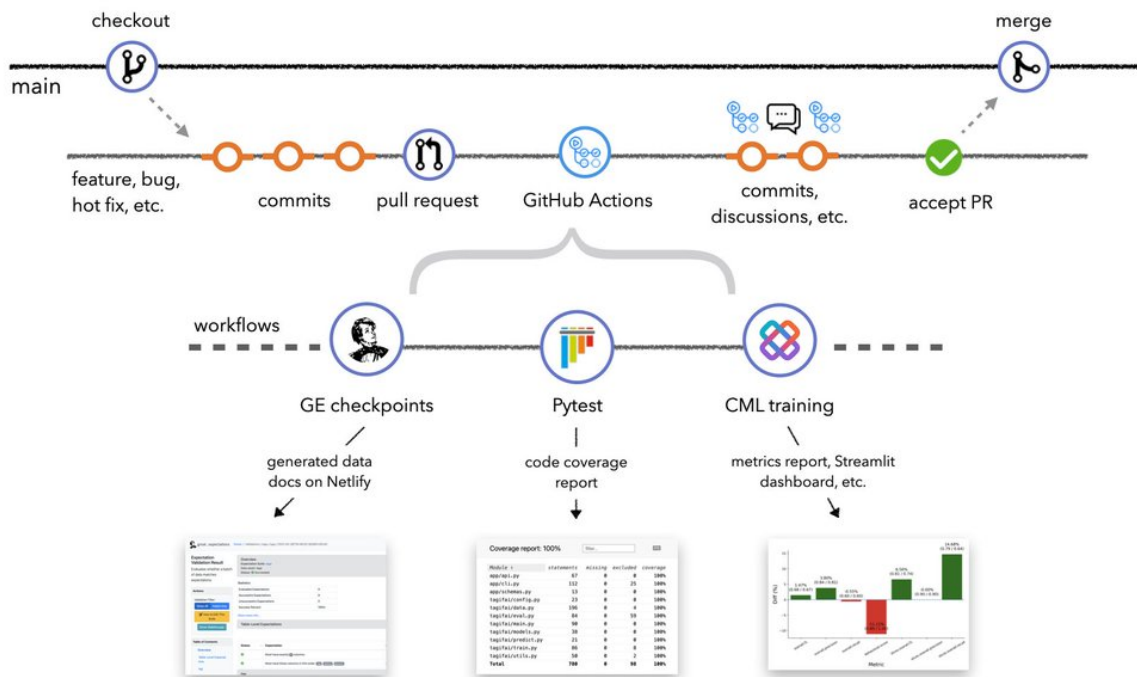
Preprocessed text:

generation via autoencoders

Then, we wrap all of the CI/CD workflows we've created with [@GitHub Actions](#):

#### ■ CI/CD workflows

- Workflow components (events, runners, jobs)
- Testing Actions locally using Act
- Best practices (ex. caching)
- ML Actions (Great Expectations checkpoints, DVC CML)

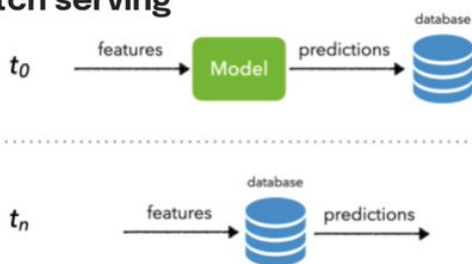


Next, we explore the infra needed to deploy & serve ML applications.

#### ■ Infrastructure:

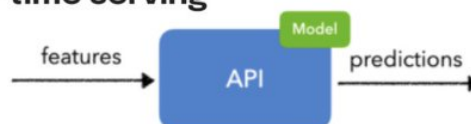
- Serving (batch, real-time)
- Processing (batch, stream)
- Learning (offline, online)
- Testing (AB, canary, shadow)
- Optimization (prune, quantize, distill)
- Methods (K8s, serverless)

#### Batch serving



- ☒ generate and cache predictions for very fast inference for users.
- ☒ the model doesn't need to be spun up as it's own service since it's never used in real-time.
- ☒ predictions can become stale if user develops new interests that aren't captured by the old data that the current predictions are based on.
- ☒ input feature space must be finite because we need to generate all the predictions before they're needed for real-time.

#### Real-time serving



- ☒ can yield more up-to-date predictions which may can yield a more meaningful user experience, etc.
- ☒ requires managed microservices to handle request traffic.
- ☒ requires real-time monitoring since input space is unbounded, which could yield erroneous predictions.

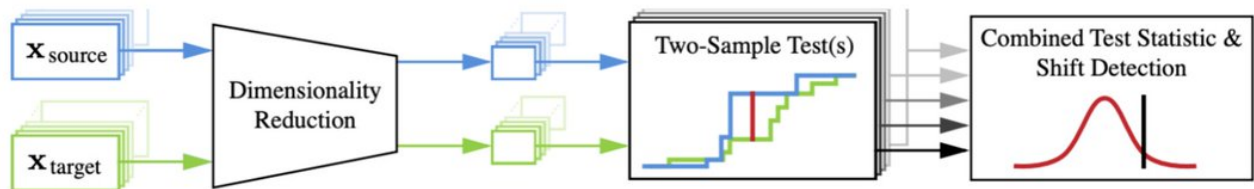
#### Note

Besides wrapping our model(s) as separate, scalable microservices, we can also have a purpose-built model server to host our models. Model servers, such as [MLFlow](#) or [RedisAI](#), provide a common interface to interact with models for inspection, inference, etc. In fact, modules like [RedisAI](#) can even offer added benefits such as data locality for super fast inference.

We ensure the health of our ML system with appropriate monitoring.

## ■ Monitoring:

- identifying drift (data, target, concept)
- measuring drift on uni/multivariate data via
- reducers (PCA, UAE)
- detectors (chi<sup>2</sup>, KS, MMD)
- solutions (not always retraining)



Detecting drift as outlined in [Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift](#)

```
1 from functools import partial
2 from alibi_detect.cd.pytorch import preprocess_drift
```

```
1 # Untrained autoencoder (UAE) reducer
2 enc_dim = 32
3 reducer = nn.Sequential(
4 embeddings_layer,
5 nn.AdaptiveAvgPool2d((1, embedding_dim)),
6 nn.Flatten(),
7 nn.Linear(embedding_dim, 256),
8 nn.ReLU(),
9 nn.Linear(256, enc_dim)
10).to(device).eval()
```

```
1 # Preprocessing with the reducer
2 preprocess_fn = partial(preprocess_drift, model=reducer, batch_si:
```

```
1 from alibi_detect.cd import MMDDrift
```

```
1 # Initialize drift detector
2 embeddings_mmd_drift_detector = MMDDrift(reference, backend="pytorch", p_val:
```

```
1 # No drift
2 no_drift = get_data_tensor(texts=df.text[-200:].to_list())
3 embeddings_mmd_drift_detector.predict(no_drift)
```

```
{'data': {'is_drift': 0,
'distance': 0.0006961822509765625,
'p_val': 0.2800000011920929,
'threshold': 0.01,
'distance_threshold': 0.008359015},
'meta': {'name': 'MMDDriftTorch',
'detector_type': 'offline',
'data_type': None,
'backend': 'pytorch'}}
```

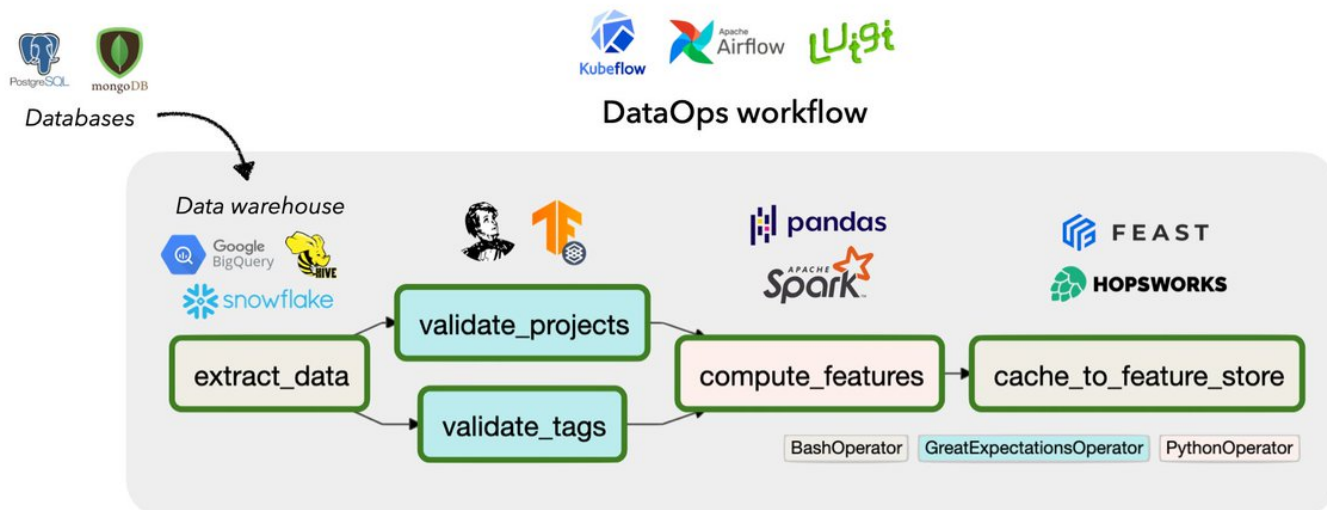
Finally, we connect our DataOps & MLOps workflows in our ML systems.

## ■■ Workflow orchestration w/ [@ApacheAirflow](#)

- DAGs
- Scheduler
- Tasks
- Operators
- Runs

## ■ Feature stores w/ [@feast\\_dev](#)

- data ingestion
- feature definitions
- historical/online features



Over the past 7 years, I've worked on ML and product at [@Apple](#), health tech startups and ran my own venture in the rideshare space. I've worked with brilliant developers and managers and learned how to responsibly develop and iterate on ML systems across various industries.

I currently work closely with early-stage & mid-sized companies in helping them deliver value with ML while diving into the best & bespoke practices of this rapidly evolving space. I want to share that knowledge with the rest of the world so we can accelerate overall progress.

ML is not a separate industry, instead, it's a powerful way of thinking about data. The foundations we've laid out will continue to hold but the methods and avenues of application will evolve. So these lessons are by no means "complete" and we'll continue to keep them up-to-date.

Even more exciting content coming later this year, so stay tuned!

- ■ Among top MLOps repos on GitHub: <https://t.co/gsYawqTq6U>
- ■■ A highly recommended resource used by industry: <https://t.co/pcMd8jZfo5>
- ♥■ 30K+ community members: <https://t.co/CswgmDZhCq>