

Machine Learning 520

Advanced Machine Learning

Lesson 10: Building Machine Learning Applications

Today's Agenda

- End-to-End Machine Learning Process
- MLOPS
- Deploying Models to Production
- Recap & Recommendations



Learning Objectives

- Build and evaluate an end-to-end machine learning project

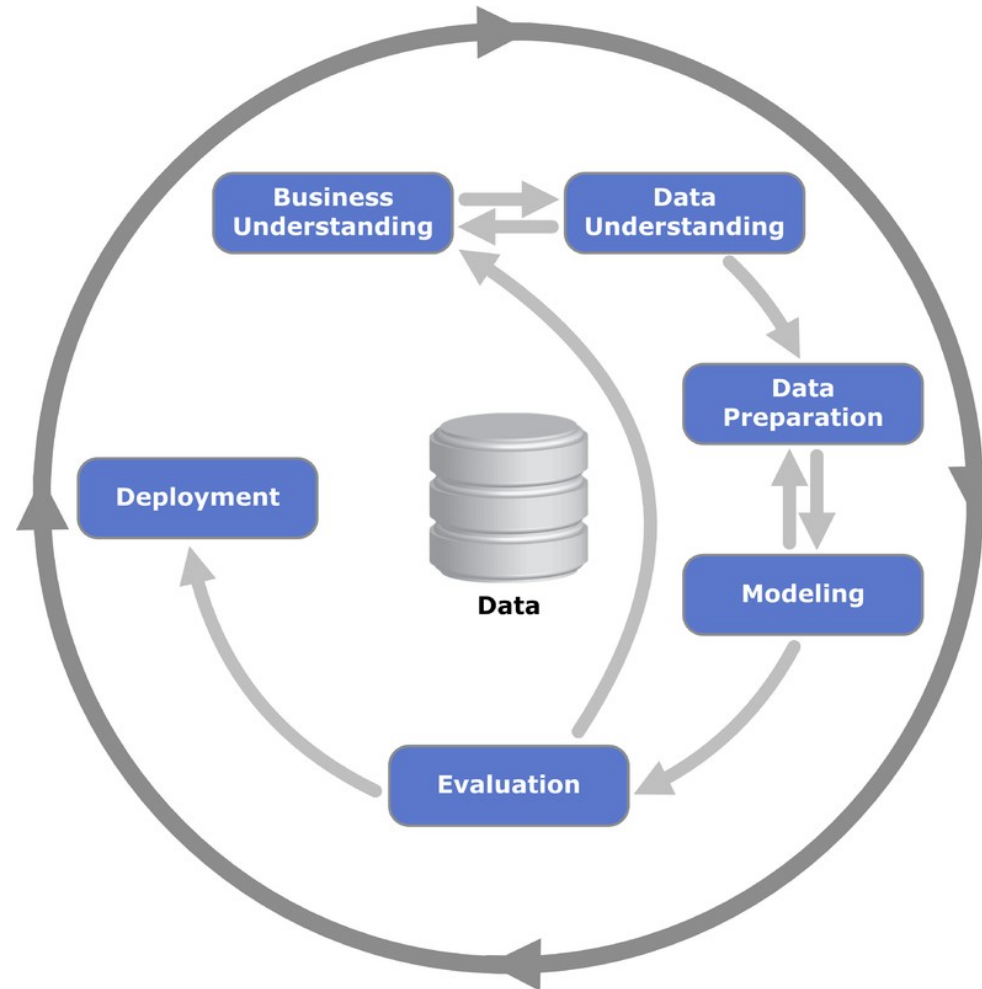


Machine Learning Everywhere

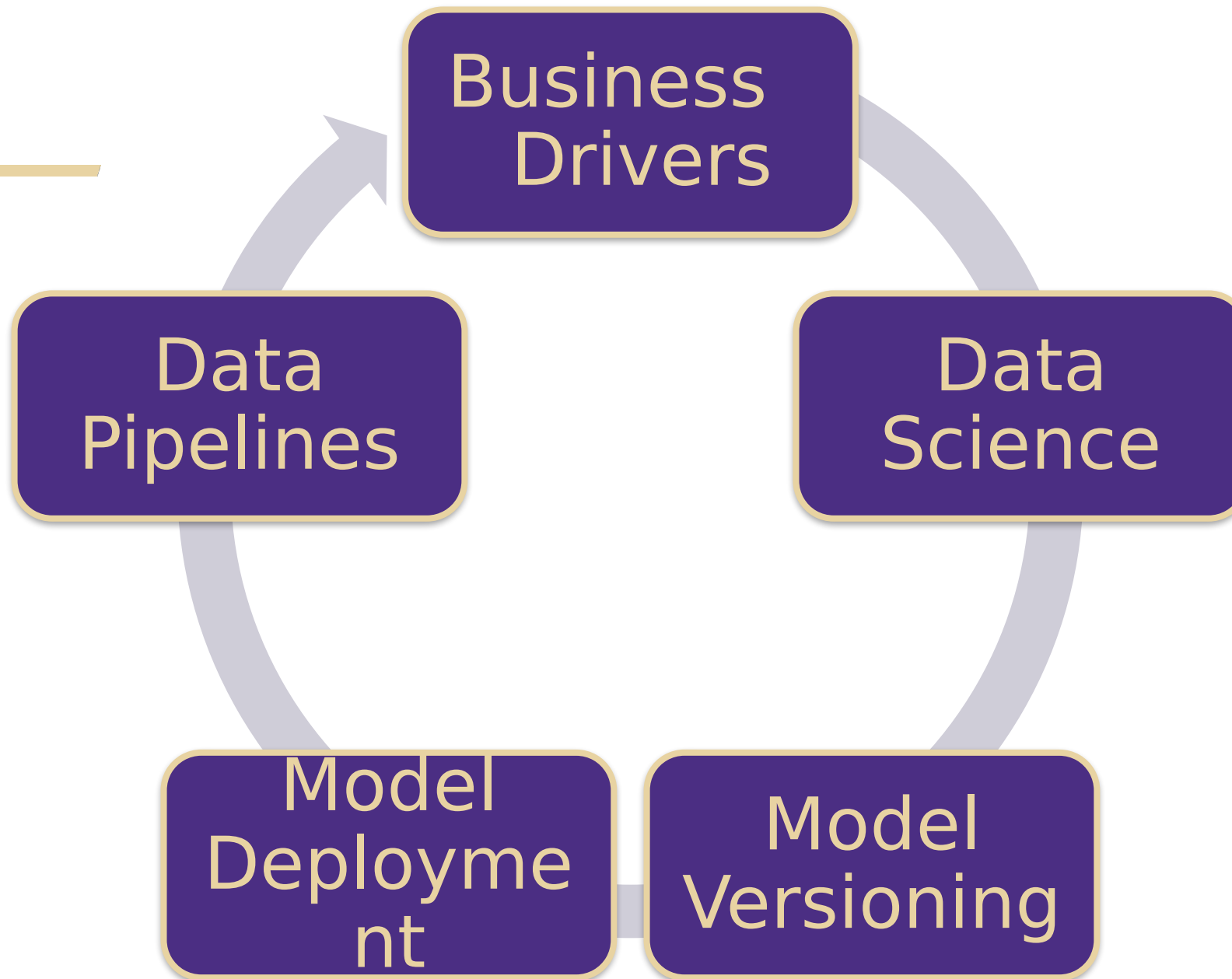
- Machine Learning is increasingly applied across various industries to solve problems
- Successful companies are moving more and more toward a data-driven culture, see [this HBR article](#)
- Many internal and external stakeholders can be involved in a Machine Learning project



CRISP-DM



ML Ops



Business Drivers

- Connect with stakeholders on low hanging fruits
- Use Design Thinking, 10 Types of Innovation, LEAN, Six Sigma, etc
- Direct and indirect **measures** and **uncertainty** around them
- Choosing key **metrics** and **benchmarks**
- Data collection, acquisition and preparation
- Stating simplifying assumptions, constraints, granularity?
- Are there **ethical considerations** that need to be addressed?
- Technical aspects of the methodology (machine learning)
- Technical aspects of the solution (operationalization)



Data Science

- Keep stakeholders updated
- Build prototypes
- Log experiments
- Iterate, iterate, iterate
- **Technology:** Python, R



Model Versioning

- Iteratively improve models
- Feature engineering
- Hyperparameters
- Track and archive models
- Track and archive data
- Make it reproducible
- **Technology:** Python, R



Model Deployment

- Add models to workflows
- Serve through APIs
- Monitor performance
- Automate deployment
- Automate rollback
- **Technology:** Spark, Docker, Kubernetes



Data Pipelines

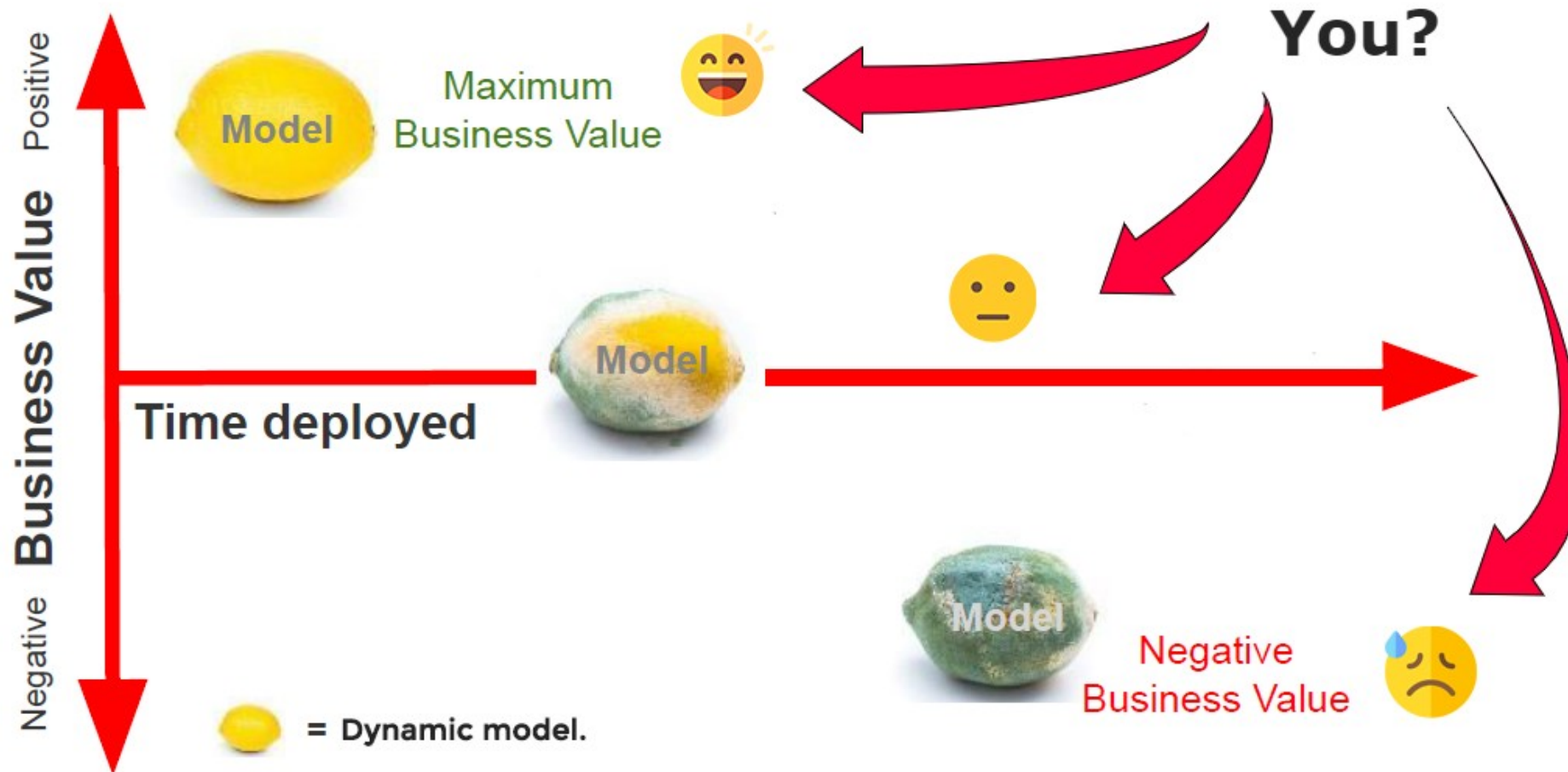
- Build robust pipelines
- Operate on real-time data
- Archive data
- **Technology:** Kafka, MQTT, Avro, Thrift



Why MLOps



Machine Learning models are dynamic and degrade over time after being deployed to production.



Emoji icons Source: www.flaticon.com

Source: <https://neptune.ai/blog/how-to-monitor-your-models-in-production-guide>



Why you need to monitor your Machine Learning models in production

- Machine learning models are dynamic and sensitive to changes in the real world, and their performance tends to degrade over time.
- Deploying a new model is akin to buying a new car: its value drops instantly, and its performance starts to decline as soon as it's put to use.
- Deployment should not be viewed as the final step, as there is ongoing work to be done to maintain and improve the model's performance.
- Regular reporting to business stakeholders is crucial to assess whether the deployed machine learning solution is effectively addressing the problem it was designed to solve.
- Model monitoring after deployment is essential to ensure that it continues to perform as expected.
- Updating the model to account for localized problems, such as geographic regions, is necessary for optimal performance.
- **The key takeaway is that the deployment of a machine learning model is just the beginning of the work involved in maintaining and improving its performance.**



Why we need to monitor ML models

Reason	Description
Detecting Drift	Monitoring helps detect drift in the model's performance, which is caused by changes in the input data distribution or other factors. Drift can lead to inaccurate predictions and reduced effectiveness of the model.
Ensuring Fairness	Monitoring helps ensure that the model is making fair and unbiased predictions, particularly with respect to sensitive attributes such as race or gender. It can also detect biases that emerge over time due to changes in the data.
Improving Performance	Monitoring helps identify areas where the model's performance can be improved, such as by fine-tuning hyperparameters or adjusting the model architecture. It can also help identify which features are most important for making accurate predictions.
Ensuring Compliance	Monitoring helps ensure that the model is complying with relevant laws, regulations, and ethical guidelines. For example, in some contexts, models must be explainable, meaning that they can be audited and understood by human experts.
Enabling Debugging	Monitoring helps diagnose and debug issues that may arise with the model, such as unexpected errors or poor performance on certain inputs. It can also help identify when the model is making incorrect or surprising predictions.
Demonstrating Value	Monitoring helps demonstrate the value of the machine learning solution to stakeholders, by providing evidence of its effectiveness and identifying areas for improvement. It can also help justify the investment in the model by showing how it is contributing to the organization's goals.

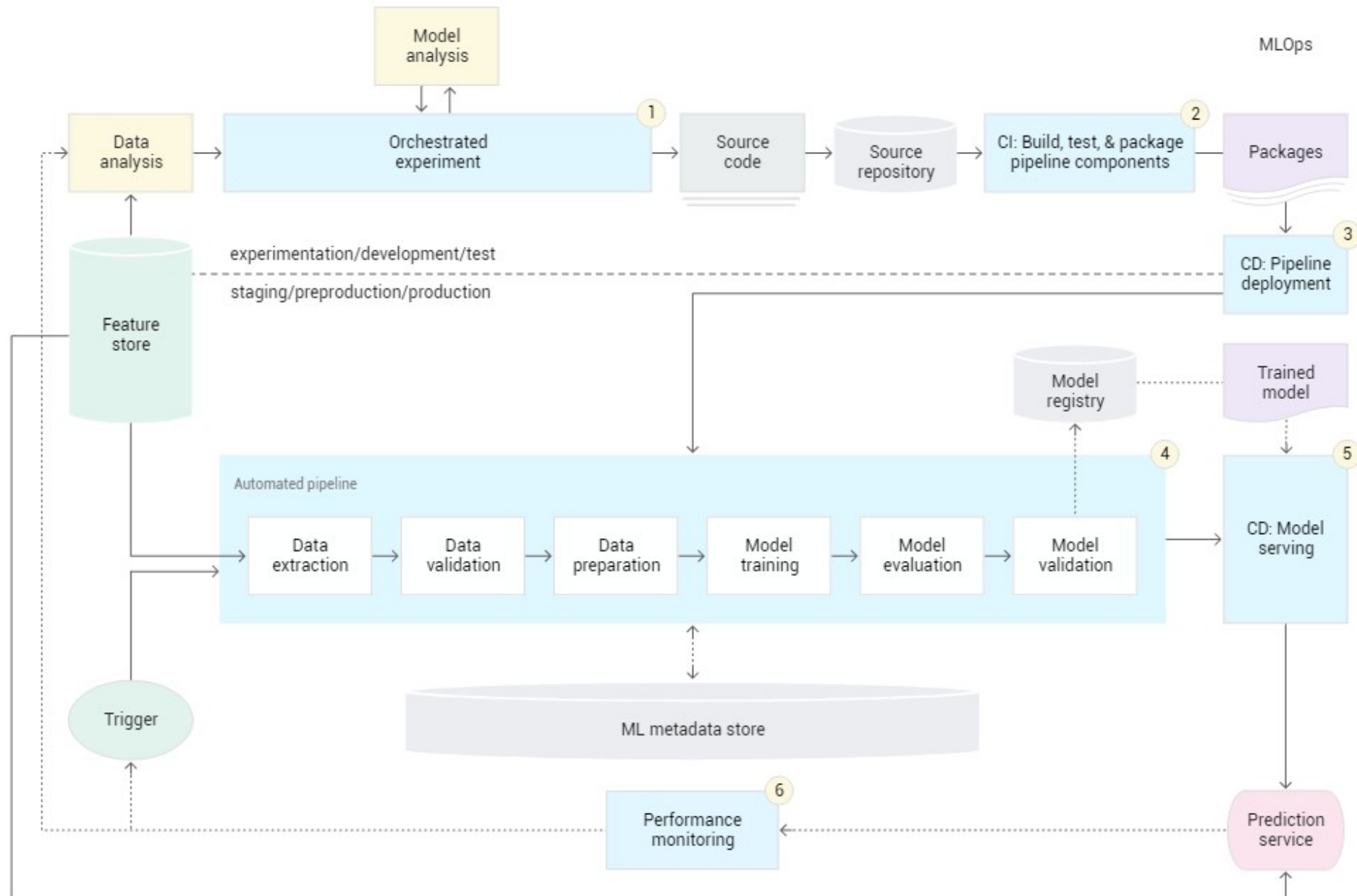




ML Ops



ML Ops Architecture

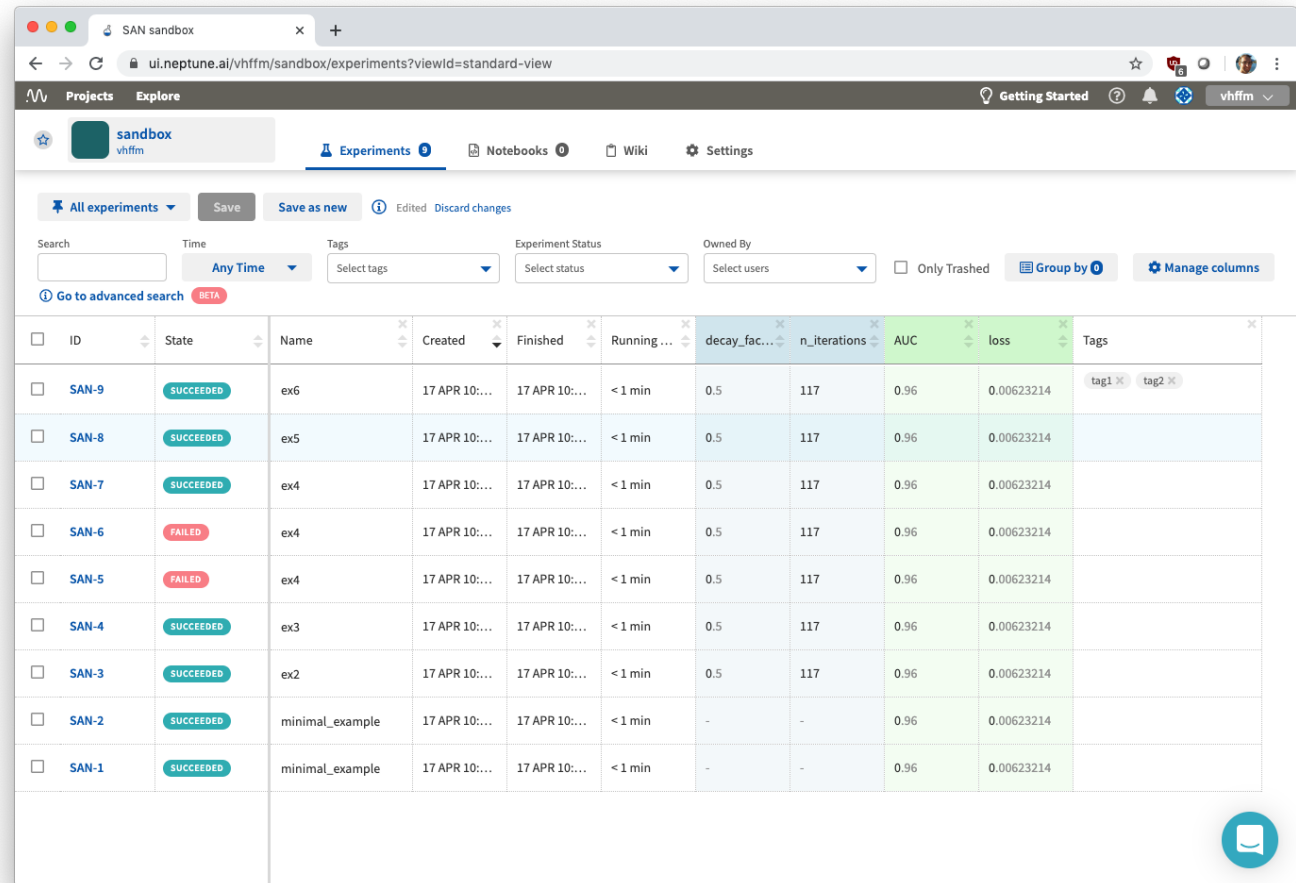


Neptune

Neptune is a tracker for experiments with focus on Python. It is a hosted service. Experiments are tracked by using library hooks to register (model) parameters, evaluation results, and upload artifacts (such as models, hashes of training data, or even code). The library can track hardware usage and experiment progress.

The results can be analyzed and compared on a website. There are also collaborative options. Neptune has integrations with Jupyter notebooks, various ML libraries, visualizers (HiFlow, TensorBoard), other trackers (MLFlow), and external offerings (Amazon Sagemaker).

They provide an API to query results from experiment. This can be used to feed CI/CD pipelines for model deployment.
<https://ui.neptune.ai/>



The screenshot shows the Neptune web interface for a user named 'vhffm'. The top navigation bar includes 'Projects', 'Explore', 'Getting Started', and a user profile. The main content area is titled 'sandbox' and contains a table of experiments. The table has columns for ID, State, Name, Created, Finished, Running, and various metrics like 'decay_fac...', 'n_iterations', 'AUC', and 'loss'. The experiments are listed in descending order of creation time. The first two experiments, SAN-9 and SAN-8, are in a 'SUCCEEDED' state, while SAN-6 and SAN-5 are in a 'FAILED' state. The remaining experiments (SAN-4, SAN-3, SAN-2, SAN-1) are also in a 'SUCCEEDED' state. The table is filtered by 'Any Time' and 'Select tags'. A 'Go to advanced search' button is visible below the filters. A 'Manage columns' button is located in the top right corner of the table area.

ID	State	Name	Created	Finished	Running	decay_fac...	n_iterations	AUC	loss	Tags
SAN-9	SUCCEEDED	ex6	17 APR 10:...	17 APR 10:...	< 1 min	0.5	117	0.96	0.00623214	tag1 tag2
SAN-8	SUCCEEDED	ex5	17 APR 10:...	17 APR 10:...	< 1 min	0.5	117	0.96	0.00623214	
SAN-7	SUCCEEDED	ex4	17 APR 10:...	17 APR 10:...	< 1 min	0.5	117	0.96	0.00623214	
SAN-6	FAILED	ex4	17 APR 10:...	17 APR 10:...	< 1 min	0.5	117	0.96	0.00623214	
SAN-5	FAILED	ex4	17 APR 10:...	17 APR 10:...	< 1 min	0.5	117	0.96	0.00623214	
SAN-4	SUCCEEDED	ex3	17 APR 10:...	17 APR 10:...	< 1 min	0.5	117	0.96	0.00623214	
SAN-3	SUCCEEDED	ex2	17 APR 10:...	17 APR 10:...	< 1 min	0.5	117	0.96	0.00623214	
SAN-2	SUCCEEDED	minimal_example	17 APR 10:...	17 APR 10:...	< 1 min	-	-	0.96	0.00623214	
SAN-1	SUCCEEDED	minimal_example	17 APR 10:...	17 APR 10:...	< 1 min	-	-	0.96	0.00623214	

```
$ conda create --name neptune python=3.6
$ conda activate neptune
$ conda install -c conda-forge neptune-client
$ cd /somewhere
```



MLFlow

MLFlow is an experiment tracker and generic model server. It can be self-hosted.

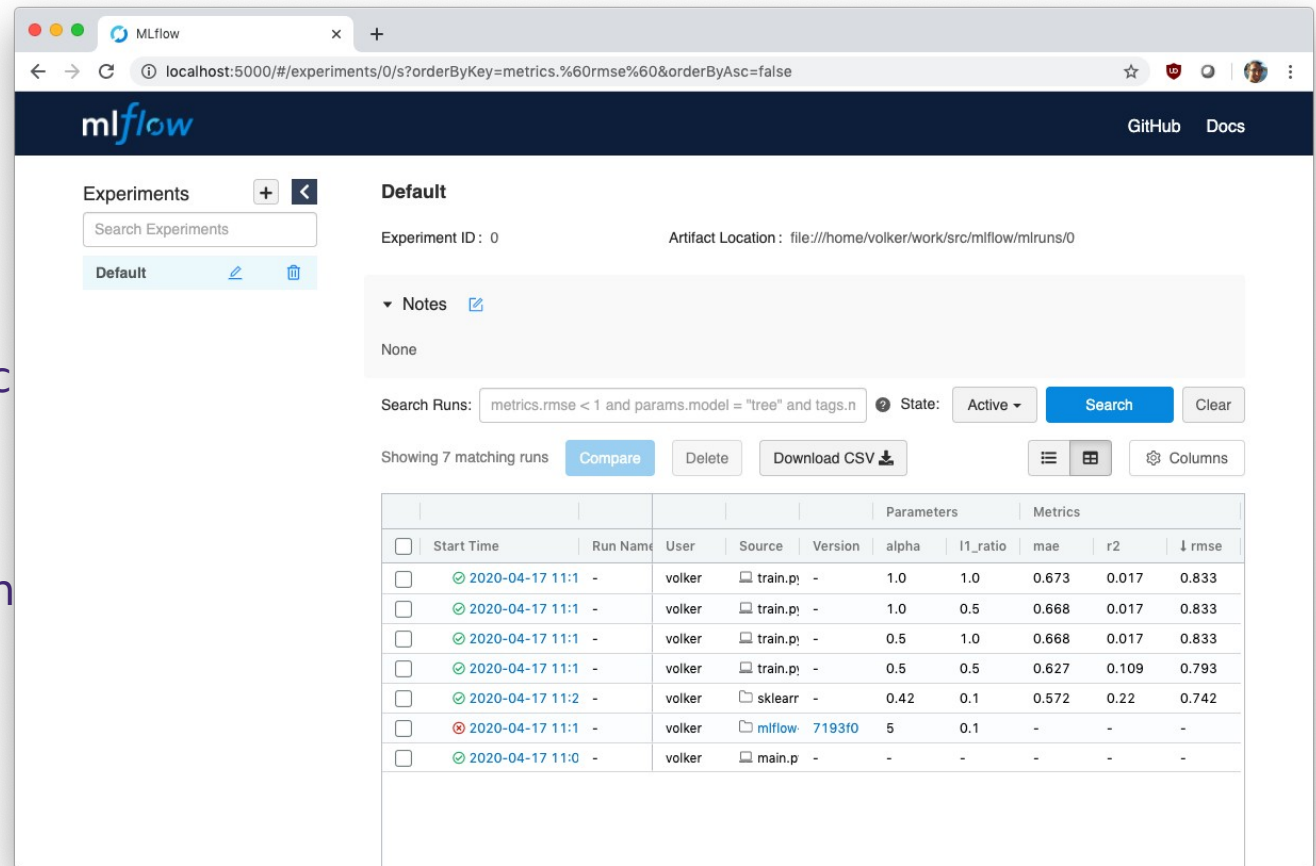
Experiments are tracked by using library hooks to register (model) parameters, evaluation results, and upload artifacts (such as models, hashes of training data, or even code). Artifacts can be logged to local, remote, or cloud storage (S3, GFS, etc).

Results can be analyzed through a web UI and CSV export is available. Models are packaged as a wrapper around the underlying format (Sklearn, XGBoost, Torch, etc). They can be pushed to Spark for batch inference or served through REST.

There are CLI, Python, R, Java, and REST APIs for further integration with CI/CD pipelines.

Models can be pushed to cloud services

(SageMaker, Azure ML, etc). <https://mlflow.org/docs/latest/quickstart.html>



```
$ conda create --name mlflow python=3.6
$ conda activate mlflow
$ conda install -c conda-forge mlflow
$ cd /somewhere
$ mlflow ui
--> http://localhost:5000
```



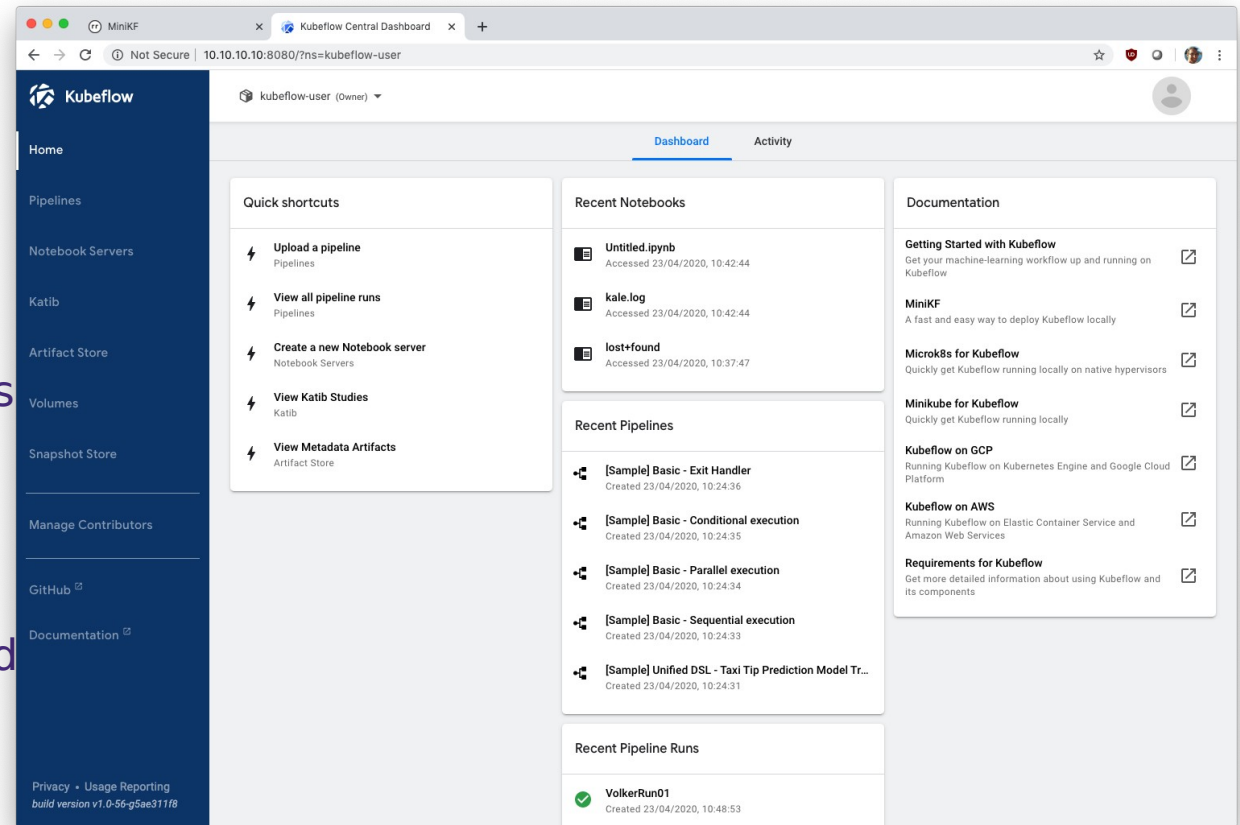
Kubeflow

Kubeflow is essentially a self-hosted version of the Google AI platform. It uses Kubernetes to abstract away infrastructure.

Kubeflow can deploy Jupyter notebooks, run pipelines for data processing and model training (scheduled, on-demand), organize runs, archive models and other artifacts, and expose models through endpoints. Pipelines are compute graphs and are described in Python with a DSL. Their components are wrapped as Docker images.

It integrates with GCP so it can elastically scale to the cloud compute and storage (e.g., distributed model training). It also integrates with offerings such as BigQuery or Dataproc.

The solution is heavy and complex but enables rapid scale-out. It is especially applicable if infrastructure is already managed through Kubernetes.



<https://www.kubeflow.org/docs/started/workstation/getting-started-minikf/>

```
$ cd /somewhere
$ vagrant init arrikto/minikf
$ vagrant up
```

---> <http://10.10.10.10>



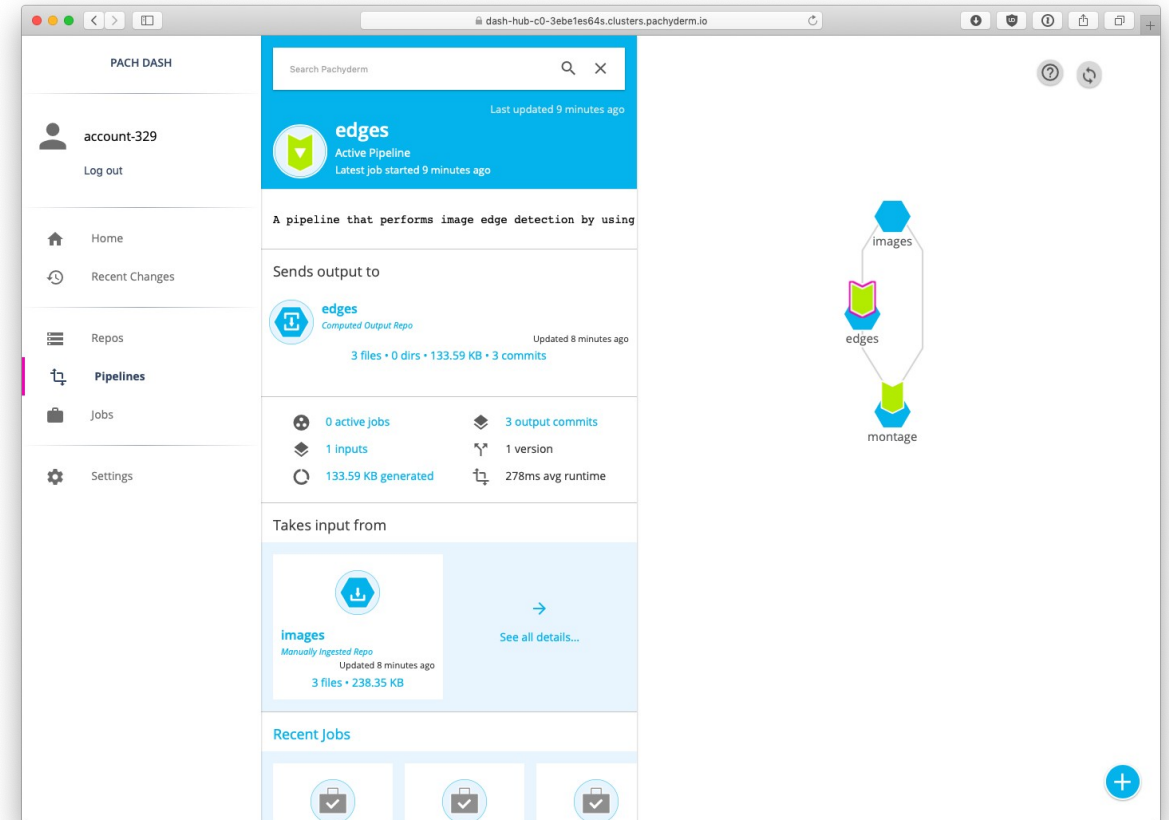
Pachyderm

Pachyderm is a versioning system and execution environment for data and processing pipelines. Hosted and self-hosted options exist.

At its core is data provenance. Data is committed to a repository and acted upon by processors in pipelines. The results (data and other artefacts like models) are committed back into a repository. By design, all use of data is traceable through pipelines. Pipelines are described in JSON and processors are packaged as Docker images. Pachyderm can integrate (but not deploy) Jupyter and can push/pull data from cloud stores (S3, etc).

Pachyderm is built on top of Kubernetes, so can easily scale-out and run in various clouds. Self-hosting comes with the usual Kubernetes complexity.

- https://docs.pachyderm.com/latest/pachub/pachub_getting_started/
- https://docs.pachyderm.com/latest/getting_started/beginner_tutorial/



```
$ cd /somewhere
$ download from (
https://github.com/pachyderm/pachyderm/releases
$ tar xfvz release_filename_linux_amd64.tar.gz
$ pachctl version --client-only
```



Common ML Ops Functionality

- 1. Model versioning and tracking**
- 2. Automated deployment**
- 3. Continuous integration and continuous deployment (CI/CD)**
- 4. Model monitoring and performance tracking**
- 5. Data management and versioning**
- 6. Model retraining and updating**
- 7. Model explainability and interpretability**
- 8. Collaboration and governance**
- 9. Automated testing and validation**
- 10. Error tracking and logging**



ML deployment options

- Cloud deployment
- Client deployment
- Hybrid deployment

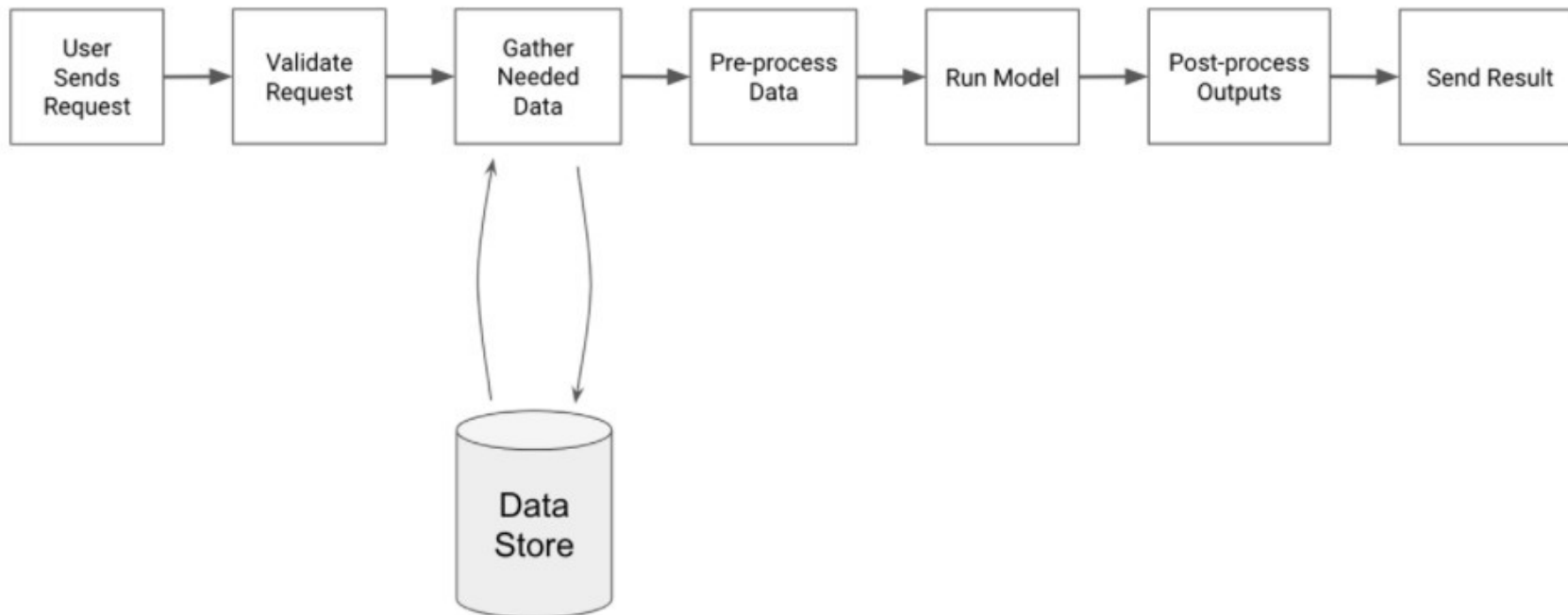
Cloud

deployment

- Set up a web server that can accept requests from clients, run them through an inference pipeline, and return the prediction results.
- Two common workflows:
 - Streaming API: accept prediction requests as they come and process them immediately.
 - Batch: process a large number of prediction requests all at once.

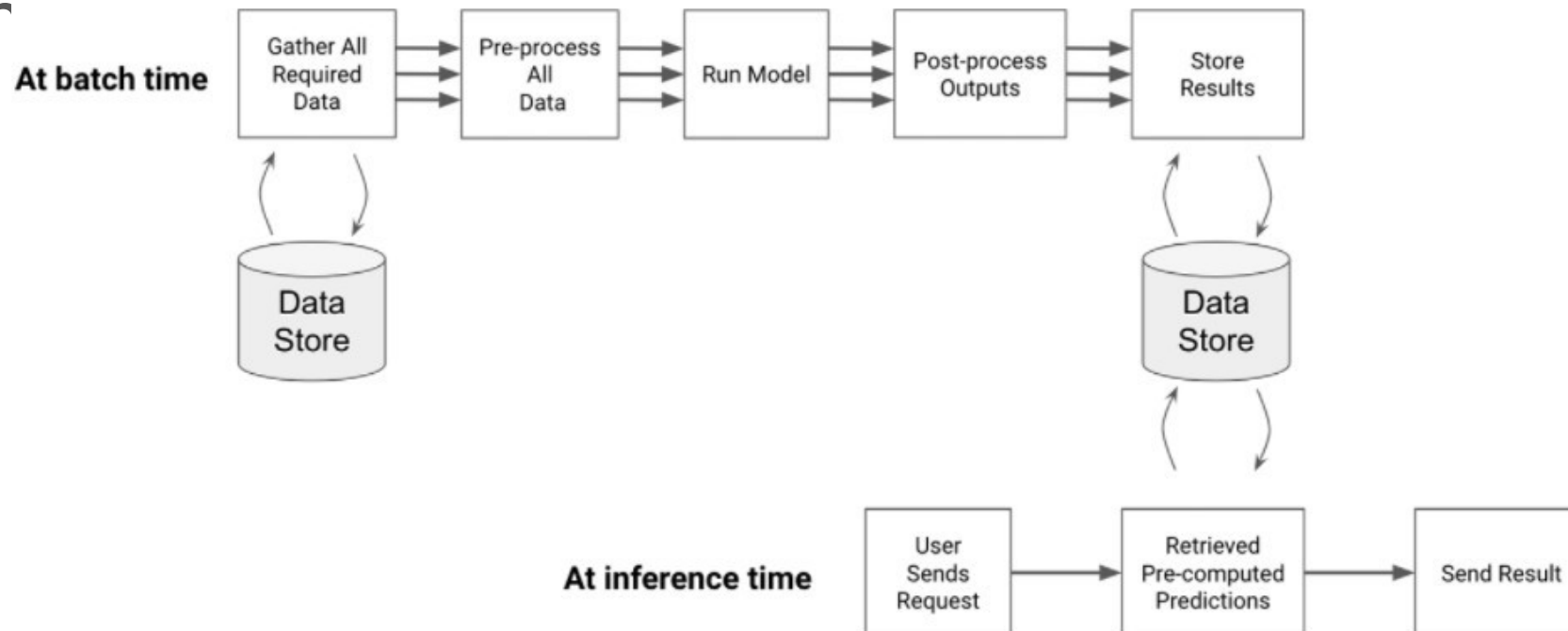
Cloud deployment – streaming API

- Streaming API workflow is good for scenarios that requires ML model's predictions to be available with low latency. Think about search engines.
- Com



Cloud deployment – batch

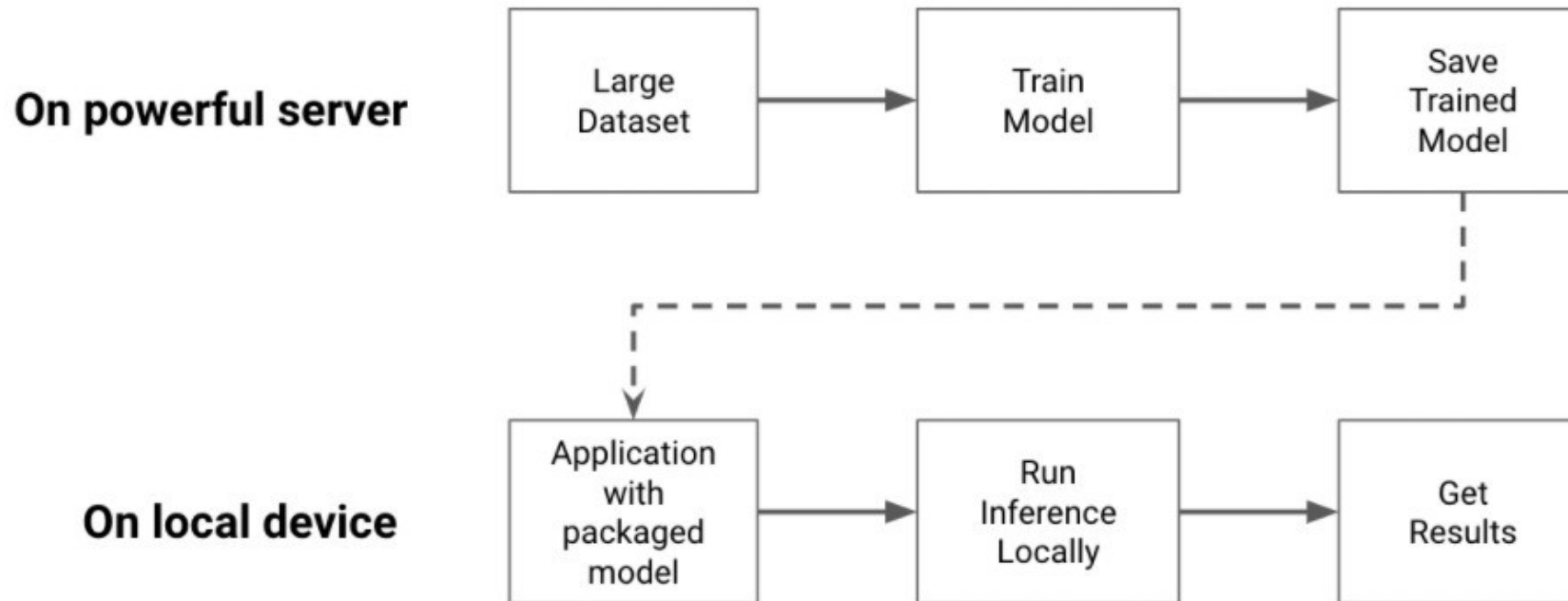
- Batch workflow is good for scenarios when we have access to model features before the model prediction is required. Think about house price prediction model.
- Com



Client

deployment

- Run all model predictions on the client, e.g. computers, tablets, smartphones, IoT devices.



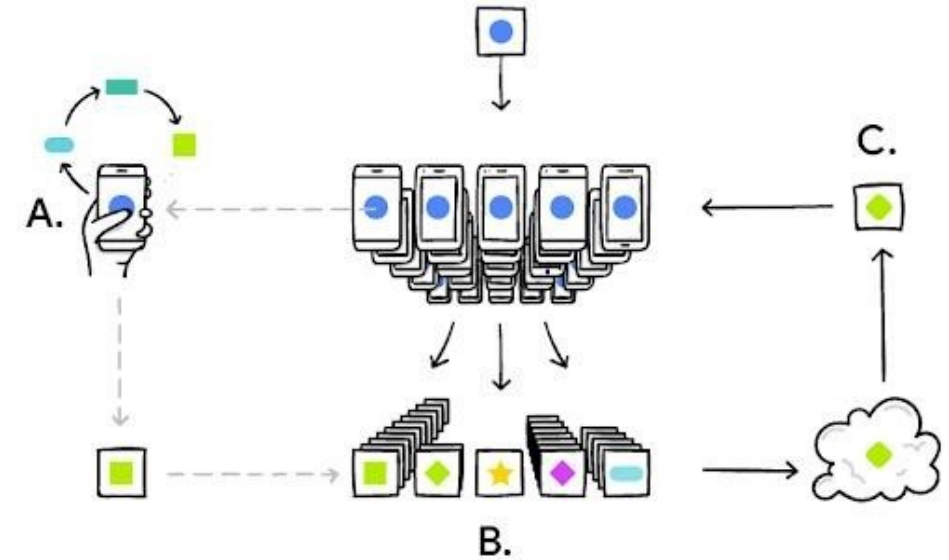
Client

deployment

- Benefits:
 - No data transmission back to the cloud.
 - Very low latency time of making a prediction.
 - Work even without internet access.
 - Privacy protection of users' sensitive data.
-

Hybrid deployment – federated learning

- A seamless integration between cloud and client deployment:
 - Each client trains its own model locally based on their user's data, and send aggregated updates to the cloud.
 - The cloud server improves its global model based on individual updates, and push the new model back to each client.
- An exciting direction that powers Google's mobile keyboard predictions:
<https://ai.googleblog.com/2017/04/fede>

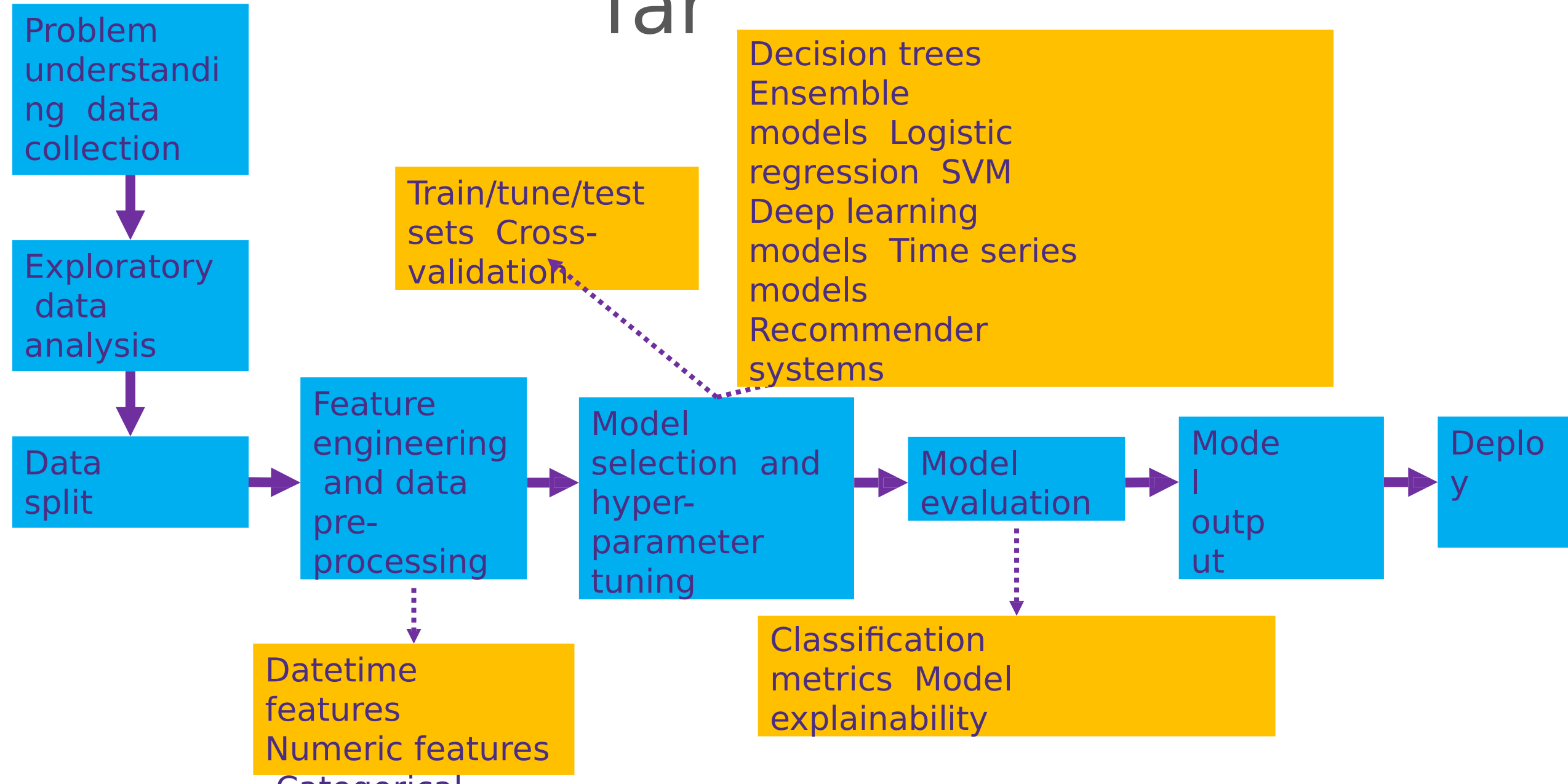




Recap

W

Recap: ML steps covered so far



General advices for ML practice

- Do you build an ML model for people, or for machines?
- 80/20 rule: grasp deep knowledge to solve 80% of all problems.
- Human judgement is influential.
 - When you have a problem at hand, always start thinking how a human domain expert would solve it manually.
 - What is the human error rate (especially for tasks in image, text, etc.)?
- Simpler is better.
 - Do you really need a deep learning model?

Tip #1: don't overlook the power of linear models

- They are computationally efficient, and can model non-linear relationships via feature transformations.
- Popularly used among production systems in industry (together with other techniques).
 - Facebook (ads click prediction):
<https://research.fb.com/publications/practical-lessons-from-predicting-clicks-on-ads-at-facebook/>
 - Google (recommender system at Google Play store): <https://arxiv.org/abs/1606.07702>

Tip #2: improve performance with more feature engineering

- Feature engineering, together with data pre-processing are more important than sophisticated ML models.
- In practice, start by trying “two extremes” as baselines:
 - “Performance floor”: fit a naïve linear model.
 - “Performance ceiling”: fit a complex model like ensemble learning.
- Perform more feature engineering to improve

Tip #3: model interpretability is important

- In the field of statistics, model interpretability is never a problem because it's required 🏖️
- In the field of machine learning, model interpretability is also required but often overlooked 🔍
 - Interpretability builds trust with stakeholders.
 - Interpretability speeds up model debugging.

Tip #4: error analysis is often overlooked

- An overall measure of accuracy often hides the details.
- Error analysis: on what population the model makes more errors?
- Example: face detection comparison across APIs,

conducted by [Microsoft](#) in August 2014

	Overall	Darker Female	Darker Male	Lighter Female	Lighter Male
Microsoft	99.52%	98.48%	99.67%	99.66%	100%
Face++	98.40%	95.90%	98.70%	99%	99.50%
IBM	95.59%	83.03%	99.37%	97.63%	99.74%
Kairos	93.40%	77.50%	98.70%	93.60%	100%
Amazon	91.34%	68.63%	98.74%	92.88%	100%

Ethics of Artificial Intelligence (AI)

- Microsoft:
<https://www.microsoft.com/en-us/ai/responsible-ai>
- Google: <https://ai.google/responsibilities/>
- IBM: <https://www.ibm.com/watson/ai-ethics/>
- PwC:
<https://www.pwc.com/gx/en/issues/data-and-analytics/artificial-intelligence/what-is-responsible-ai.html>

Farewell, and stay connected

- Please complete the end course survey on Canvas.
- I will submit your final grades one week after the final assignment due date.
- Thank you for all your patience and feedback!
- Farewell, and stay connected