Data Distribution Shifts

- **Covariate shift** (input distrib. changes while conditional proba of output | input is same – demographics changes, but proba income same).
- **Label shift** (output distrib. changes, but input distrib. same) – share of pos labels doubled
- **Concept drift** (input data remains the same, but the output changes - changes in real estate market caused by the COVID-19 pandemic).

Detecting Data Distribution Shifts

- **Accuracy**, F1 score, recall, and AUC-ROC - requires ground truth (may not always be available)
- **Statistical methods** (mean, median, variance, two sample method - if differences between two sets of data are statistically significant).
- **Time scale windows** for detecting shifts (analyze **stats across various time windows** to identify when data changes occurred)

Addressing Data Distribution Shifts

1. Train models with **massive datasets** (cover all potential data points)
2. **Adapt pre-trained models** to a target distribution without requiring new labels
3. **Retrain model** with labeled data from the new target distribution – complicated: retrain from scratch or fine-tune? Which data to use for retraining?

Monitoring and Observability

- Monitoring - tracking, measuring, and logging metrics to identify errors.
  - Monitoring accuracy-related metrics
  - Monitoring preds, features, raw inputs
  - Setting up Logs, Dashboards, Alerts
- Observability - system setup for easier visibility and issue diagnosis

Continual Learning

- <u>Continual learning</u>, <u>monitoring</u>, and <u>testing in production</u> are vital to maintaining an adaptable ML system.
- Purpose - **adapt to evolving data distrib.**, online learning algos to update model params continuously w/new data
- Stateful training - **incremental learning** on new data.
- Stateless retraining - **training from scratch** each time.
- **Data iteration** - training from scratch or retraining the same model.
- **Model iteration** - adding <u>new features</u> or modifying <u>model's architecture</u>.

Infrastructure and Tooling for **MLOps**

- **Storage and Compute:** Amazon S3, CPU or GPU, AWS EC2, etc.
- Public Cloud vs. Company-Owned Data Centers
- Multicloud Strategy - minimize dependence on a single cloud provider

Dev Env

- IDE

- **Versioning** of code, parameters, and data
- CI/CD, GIT, Weights & Biases.
- MLflow for tracking

- Need to standardizing dev environments.
- **Docker containers** simplify deploy in prod – Dockerfile re-creates model run env => builds Docker image,
- Complex applications - multiple containers (featurizing on CPU and training on GPU)
- **Kubernetes** creates a scalable network for containers.

Resource Management

- **Cron programs** use DAGs to schedule and jobs based on event triggers
- **Schedulers** are concerned w/when and how to run jobs
- **Orchestrators** (Kubernetes) procure resources, handle lower-level abstractions such as machines, clusters, and replication, and can provision more computers.

- **Data Science Workflow Management**: Airflow, Argo, Prefect, Kubeflow, and Metaflow.
- **ML Platform**: Model Deployment, Model Store, Feature Store
- Build Versus Buy Decision