Catch Me If You Can

Keeping up with Machine Learning in Production 🏃‍♀️

Shreya Shankar, UC Berkeley 🐻
November 2021

risel ab
UC Berkeley

Agenda

- My story
- Intro to machine learning (ML) pipeline development
- The case for ML observability
- Concrete patterns and problems in ML observability
- **Today's focus: continuous integration (CI) for ML pipelines**
  - Building and debugging a pipeline with `mltrace`
  - Future work for `mltrace`

My Story 📕

About me 👩🏾‍💻

- Grew up in Texas 🤠
- BS & MS from Stanford Computer Science 🌲
- Adversarial ML research at Google Brain 🧠
- First ML engineer at an applied ML startup 🦄
  - Worked with terabytes of time series data
  - Built infrastructure for large-scale ML and data analytics
- PhD student at UC Berkeley 🐻

Evolution of my Interests 🧬

- In school and research, I trained *many* models and cared about *robustness*
  - Fairness
  - Generalizability to unknown distributions
  - Security
- In industry, I wanted to train *few* models but do *lots* of inference
- What happens beyond the validation or test sets?

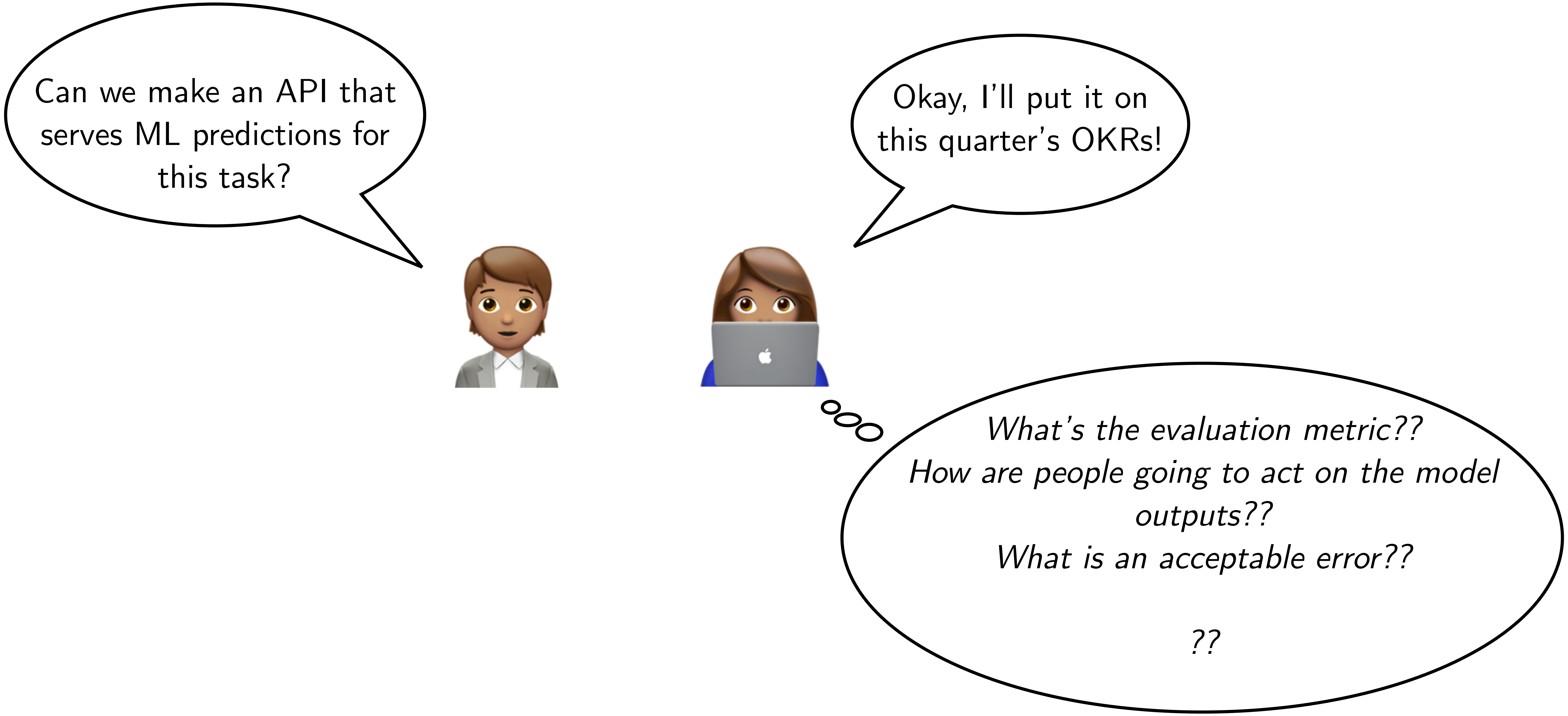
The Depressing Truth about ML IRL 💔

- Most data science projects don't make it to production
- Data in the "real world" is not clean and balanced like canonical datasets (e.g., ImageNet) ⚖️
- Data in the "real world" is always changing 🌎
- Things break in prod
  - So...we need ops 🕵🏽

Intro to ML Pipeline Development* 👷

*at a non-FAANG company
(Maybe even at a FAANG company)

Step 1: “Business Planning”



Step 2: Model Development (Offline) 📴

Garbage In; Magic Out? 🪄

1. Get the data 📋
2. Do some exploratory data analysis (EDA) 📊 📈
3. Figure out a prediction task and evaluation metric ✅
4. Train and evaluate models 🧮
5. Deploy the best model ?? 🤠

Step 3: Deploy to Production 🚀

Set and Forget? 🤭

1. Build a pipeline of components (a DAG) to serve predictions 🔧
2. Schedule the DAG 📆
3. Set and forget? **This is not how production software works!** 🤔

Production ML

An on-call engineer's biggest nightmare 😱

- Upstream data owned by someone else is corrupted?
- Model needs to be retrained, but model developer is on leave and no one else knows how the bespoke ML works?
- Feature generation code has a bug?
- Training assumptions don't hold in practice (e.g., data arrives in the system after some *lag*)?
- And more...

Production ML

An on-call engineer's biggest nightmare 😱

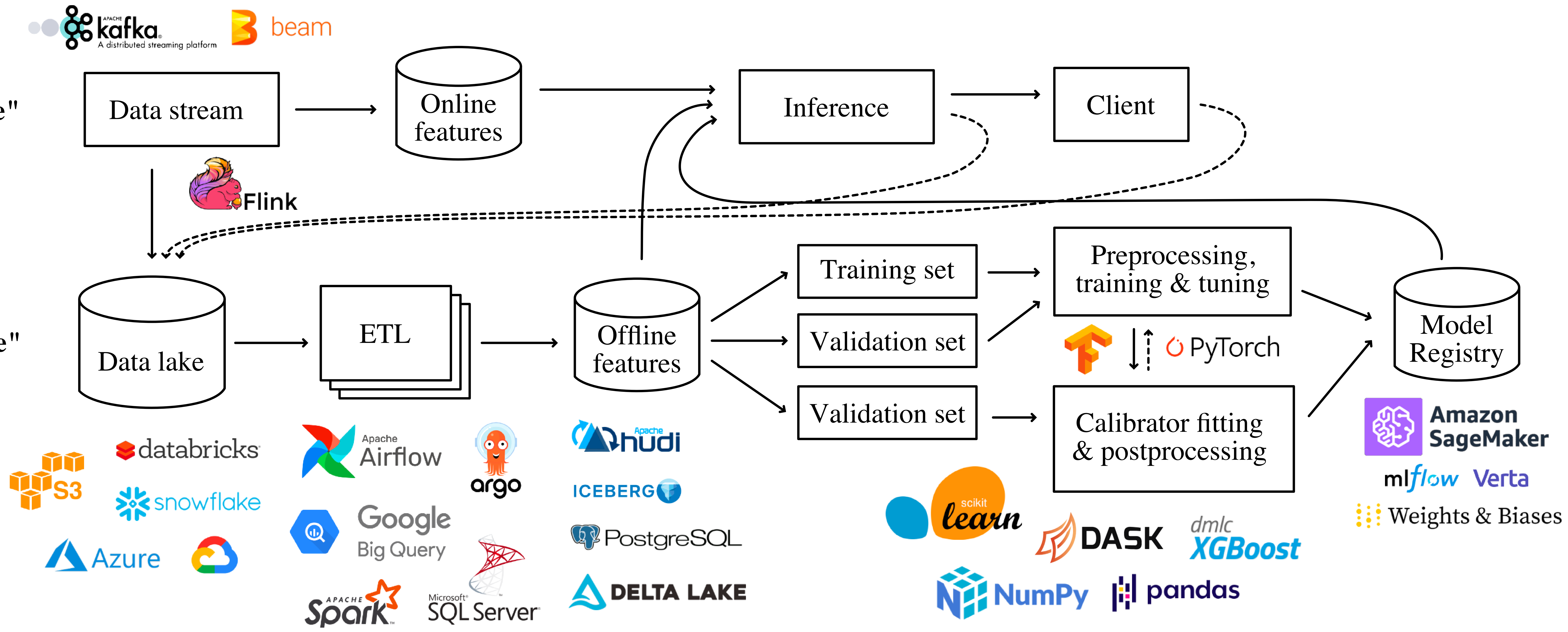


Figure 1: High-level architecture of a generic end-to-end machine learning pipeline. Logos represent a sample of tools used to construct components of the pipeline, illustrating heterogeneity in the tool stack. *Shankar et al. 2021*

Life as an ML Engineer 🌩️

- Chasing bugs that could lie anywhere across the pipeline 🤯
- Managing all the metadata around keeping models in production (e.g., feature stores, model registries, data quality tests)
- Stitching together tens? hundreds? thousands? of MLOps "tools" to make their lives easier, even though a hodgepodge of tools threatens sustainability 🤔

Life as ~~an MLE~~ a Site Reliability Engineer ☔

- Chasing bugs that could lie anywhere across the pipeline 🤯
- Managing all the metadata around keeping ~~models~~ software in production (e.g., ~~feature~~ metric stores, ~~model~~ registries, ~~data quality~~ tests)
- Stitching together tens? hundreds? thousands? of ~~ML~~DevOps "tools" to make their lives easier, even though a hodgepodge of tools threatens sustainability 🤔

The Case for ML Observability 🔍

Why Observability?

- Observability: end-to-end visibility into a pipeline or system
- Systems running in production over long periods of time will always have bugs
  - Goal: minimize downtime
  - To achieve the goal:
    - Help engineers identify bugs
    - Help engineers react to bugs

Why not use existing DevOps tools?

- Use Github Actions/TravisCI/other continuous integration (CI) tools that runs on pull request?
  - Traditional software changes on code change ✅
  - ML pipeline behavior changes on code *and* data change ❌
- Use Github/Prometheus/ELK/other continuous deployment (CD) tools?
  - Traditional software can be monitored simply with few metrics ✅
  - ML SLAs are loosely defined, span many metrics, and may not have real-time feedback ❌

Applying Observability to ML

- Need to support a wide variety of skill sets
  - Engineers, data scientists, etc.
- Logging 🪵
- Monitoring 🏥
- Querying 🕵️‍♀️

Concrete Patterns and Problems in ML Observability

Four ML Debugging Patterns 🔬

1. Component run-level queries

- "What were the inputs to yesterday's data cleaning job?"
- "Why did this specific training run fail?"

Four ML Debugging Patterns

2. Component history queries

- “Has the distribution of inputs to the inference component changed in the last month?”
- “Can we compare inference outputs over the last 3 months across different subgroups?”

Four ML Debugging Patterns 🔬

3. Cross-component queries

- "Is there a code discrepancy between the online inference component and offline evaluation component?"
- "Can we assert the same data quality tests in the online components as the ones in the offline components?"

Four ML Debugging Patterns 🔬

4. Cross-component history queries

- "Can we get end-to-end traces for these predictions?"
- "How does the trace for user A's prediction differ from user B's prediction?"

Concrete Problem #1 🪵

How do we efficiently and automatically log the right metadata at runtime for users?

- Logs (*noun*, /lôgs,lägs/): what practitioners search post-hoc to answer "🪡-in-a-haystack" queries)
- Log I/O and dependencies
- Run *reusable* computation (e.g., tests for drift) on I/O
- Versioning and storage can be expensive

Concrete Problem #2 🏥

How do we empower users to monitor ML pipelines?

- What are the crucial ML-related "SLAs" to monitor?
  - Are we really going to monitor KL divergences for thousands of features...
  - Direct feedback (labels) may not be immediately available
- Need to materialize large states (i.e., historical component I/O) at runtime for users to compute metrics on
  - This could be expensive
- A database junkie's favorite phrase: "constraints & triggers" for ML pipeline health

Concrete Problem #3 🕵🏽‍♀️

How do we develop observability interfaces for users?

- Need to support a wide range of query patterns
- Efficiently and quickly "slice and dice" traces for outputs
- Tracking "end-to-end" data flow means we need to support heterogenous stacks of tools
- Build reusable tests, metrics to monitor, and pipeline components

`mltrace`: Introduction and Demo

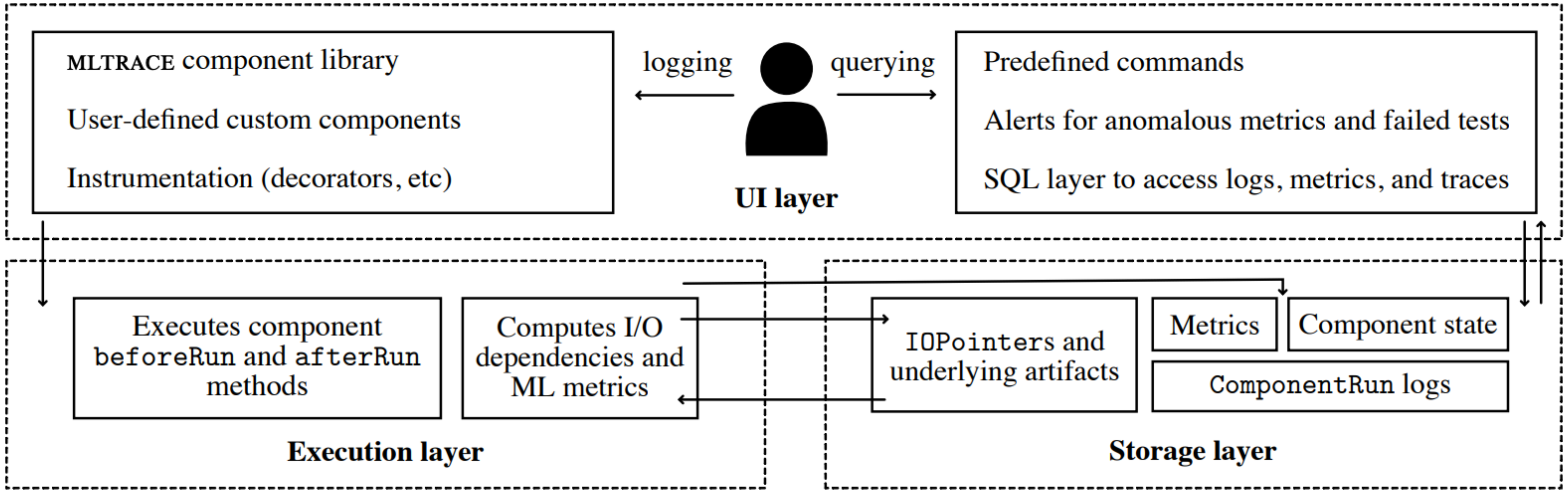
mltrace

- Lightweight, end-to-end framework centered around the execution of individual components
- Wraps around existing MLOps tools and tech stacks
- Open-source, can set up and run in-house
- Eventually want to support:
  - Logging of component runs and I/O
  - Pluggable library of components with metrics and alerts to sustain ML performance
  - Flexible querying framework for users to debug pipelines

mltrace

Proposed architecture

- UI layer: log data (e.g., inputs, outputs, test results) at runtime & query logs
- Execution layer: runs “triggers” and commits logs to the DB
- Storage layer: houses logs (e.g., inputs, outputs, metrics, test results)



`mltrace`

Client-facing abstractions

- Component: a stage of a pipeline, typically owned by one person or team
  - E.g., model training, windowed feature generation
  - Represented by a name, owner, description, and (optional) identifier tags
- `ComponentRun`: a log that represents an instance of a component being run
  - Contains start & end timestamps, inputs, outputs, and other metadata
- Test: reusable computation to perform before and after components are run
  - Added to `Component` specifications

Demo: Building an ML Pipeline 👷🏽

- Goal: Showcase `mltrace` testing functionality
- Steps
    - Build ML training and inference pipelines in Python with `Pandas` and `sklearn`
    - Experience performance drop when inference pipeline runs for a long period of time
    - Instrument pipelines with `mltrace` components
    - Debug pipeline and add tests

Demo: Building an ML Pipeline 👷🏽

1. Set up

- Prereqs: Python 3.7+, unix-based shell, Docker
- Clone mltrace-demo repository
- Install required packages in first step of README
  - `pip install -r requirements.txt`
- Clone mltrace repository (for serving the log DB & querying app)
  - `export DB_SERVER=localhost`
  - `docker-compose build; docker-compose up`

Demo: Building an ML Pipeline 👷

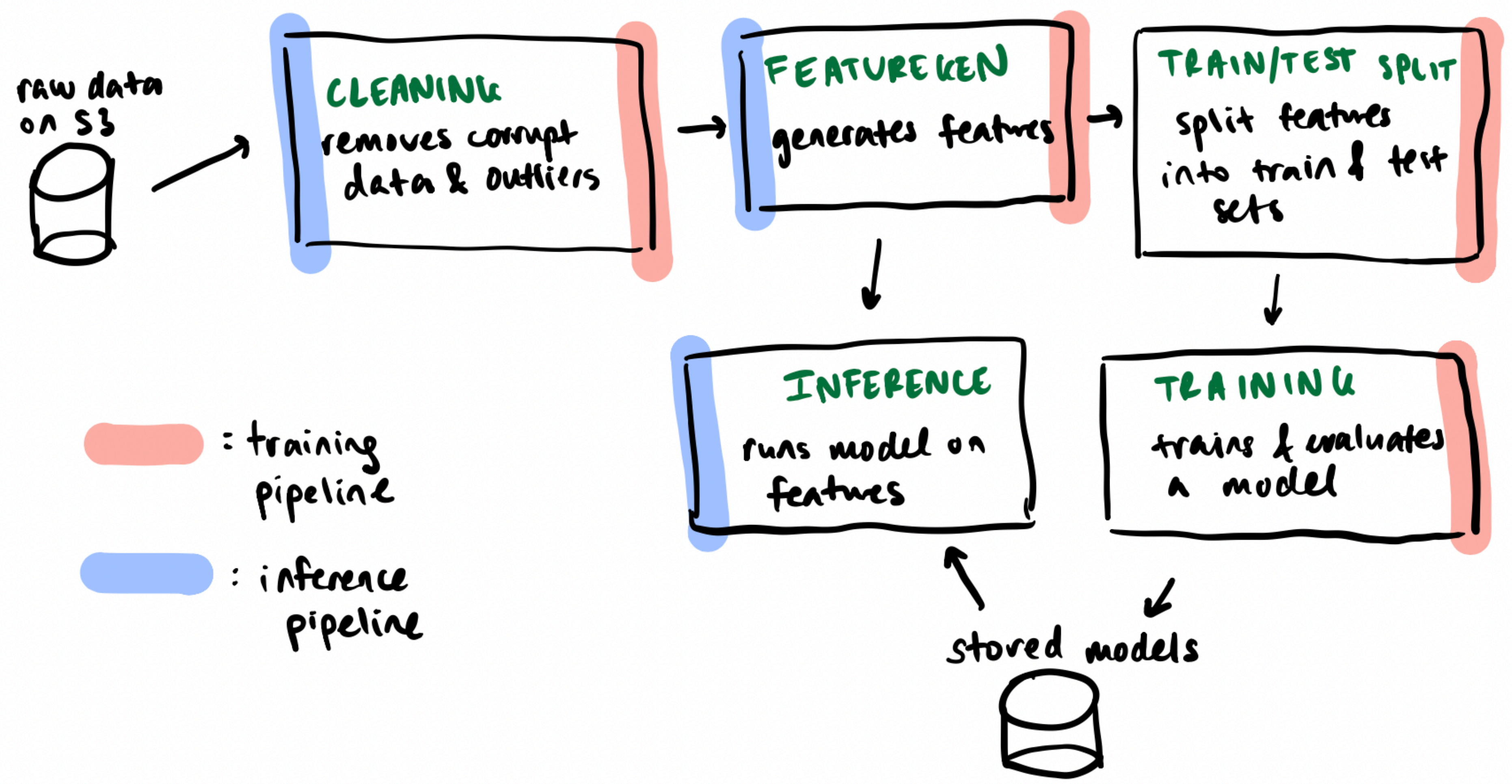
2. Task familiarization 🚕

- Binary classification task: predict whether a passenger in a NYC taxi ride will give the driver a "reasonable" tip (>10% of fare)
- Evaluation metric: F1 score
  - F1 is a combination of precision and recall
  - We also measure accuracy, precision, and recall
- Using subsampled data from NYC Taxi & Limousine Commission public dataset
- Using `pd.DataFrame` and `sklearn` Random Forest Classifier

Demo: Building an ML Pipeline 👷🏽

3. Pipeline familiarization

- One shared script `main.py` that contains code for training and inference pipelines
- Train model on data from January 2020
- "Deploy" / run inference weekly from February 1, 2020 to May 31, 2020



Demo: Building an ML Pipeline 👷

4. Run pipelines 🏁

- Download data
  - `source download.sh`
- Train model on data from January 2020
  - `python main.py --mode=training`
- "Deploy" / run inference weekly from February 1, 2020 to May 31, 2020
  - `python run_weekly_inference.py`

Demo: Building an ML Pipeline 👷🏽

5. Observe performance drop 📉

- Metrics are dropping over time 😱
- If we were at a company, alarms would be going off 🔔
  - Some poor on-call engineer would have to debug the pipeline 😤
  - They might not have built the pipeline!
- *Where do we begin debugging?*
  - Code and data are centralized in this example, but in practice they are not

Demo: Building an ML Pipeline 👷🏽

6. Instrument pipelines with `mltrace`

- Add code to get *tracing*
- Define component specifications

```
from mltrace import Component

class Cleaning(Component):
    def __init__(self, beforeTests=[], afterTests=[]):

        super().__init__(
            name="cleaning",
            owner="plumber",
            description="Cleans raw NYC taxicab data",
            tags=["nyc-taxicab"],
            beforeTests=beforeTests,
            afterTests=afterTests,
        )
```

Example component spec in `components.py`

```
from components import *

@Cleaning().run(auto_log=True) # This is the only line of mltrace code to add
def clean_data(
    df: pd.DataFrame, start_date: str = None, end_date: str = None
) -> pd.DataFrame:
    """
    This function removes rows with negligible fare amounts and out of bounds of the start and er

    Args:
        df: pd dataframe representing data
        start_date (optional): minimum date in the resulting dataframe
        end_date (optional): maximum date in the resulting dataframe (not inclusive)

    Returns:
        pd: DataFrame representing the cleaned dataframe
    """
    df = df[df.fare_amount > 5]  # throw out neglibible fare amounts
    if start_date:
        df = df[df.tpep_dropoff_datetime.dt.strftime("%m/%d/%Y") >= start_date]
    if end_date:
        df = df[df.tpep_dropoff_datetime.dt.strftime("%m/%d/%Y") < end_date]

    clean_df = df.reset_index(drop=True)
    return clean_df
```

Example integration of component spec into pipeline code in `main.py`

Exercise 1: Instrument other functions in `main.py`

Hint: you will only have to instrument 4 other functions!

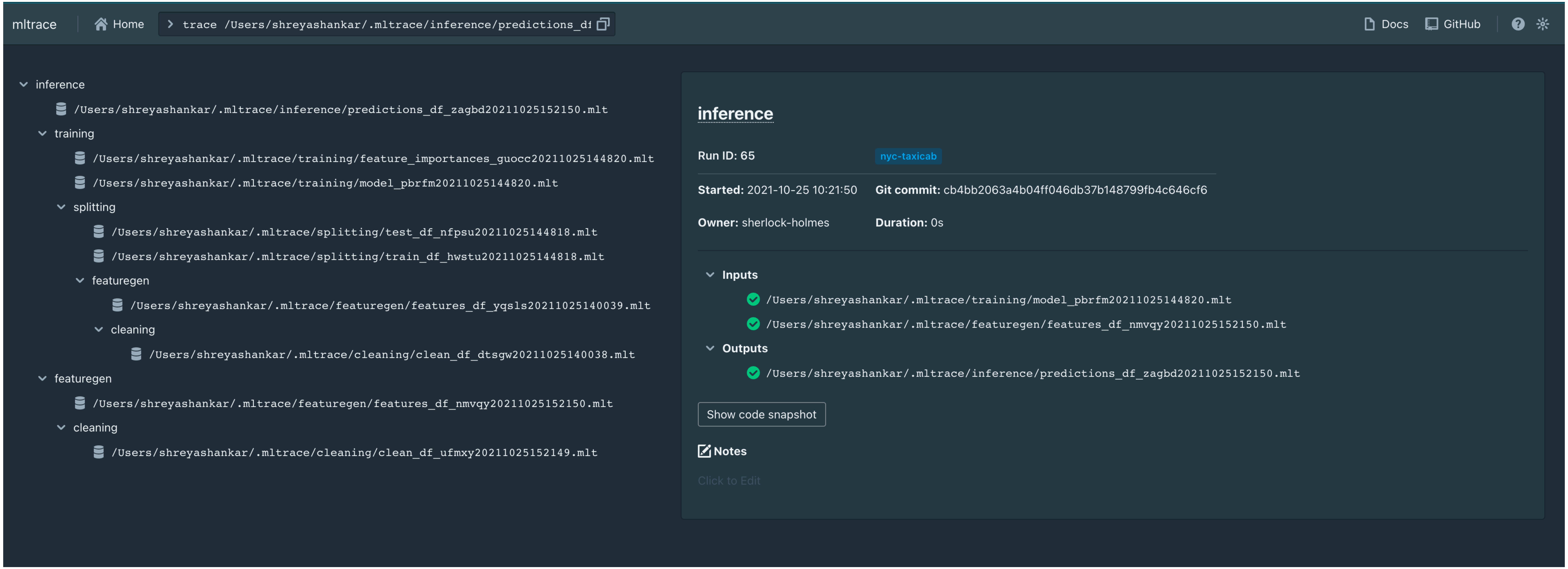
Demo: Building an ML Pipeline 👷

7. Tracing and debugging 🐛

- Rerun pipelines with `mltrace` instrumentation
  - `python main.py --mode=training`
  - `python run_weekly_inference.py`
- Trace outputs in UI (localhost:8080)
  - `history inference`

Demo: Building an ML Pipeline 👷🏽

7. Tracing and debugging 🐛



Exercise 2: Load and analyze intermediates

1. Open a Jupyter notebook
2. Call `mltrace.load(filename)` to get intermediate data in a dataframe

Demo: Building an ML Pipeline 👷🏽

8. Encode tests

- `mltrace` components can execute tests before and after they are run
- Existing tests in tutorial (`tests.py`):

Test Class Name	Description
`OutliersTest`	Prints summary statistics of the data and tests for z-score outliers.
`TrainingAssumptionsTest`	Tests for train-test leakage and makes sure the data does not have a large class imbalance.
`ModelIntegrityTest`	Checks that the model did not overfit and that feature importances aren't heavily skewed towards a small fraction of features.

- Example integration:

```
from tests import *

@Featuregen(afterTests=[OutliersTest]).run(auto_log=True)
def featuregen(...):
```

Exercise 3: Add tests to `mltrace` components

1. Add the `TrainingAssumptionsTest` and `ModelIntegrityTest` to components in the training pipeline. *Hint: training assumptions should be satisfied before training, and model integrity should be satisfied after training!*
2. Run the pipelines (`python main.py --mode=training; python run_weekly_inference.py`) as we have done before. Some inference runs should fail the outliers test.

Demo: Building an ML Pipeline 👷🏽

Takeaways

- In this tutorial, we did the following:
  - Train a model
  - Simulate deployment by running inference on a weekly basis
  - Use `mltrace` to investigate the performance drop and add tests to our pipeline
- **Lots of room for improvement!**

`mltrace` Ongoing and Future Work

ML Pipeline Testing

- We are working on...
  - Materializing historical component run inputs and outputs to use while writing running tests (e.g., to compare successive batches of data fed into a component)
  - Logging component run parameters and showing visualizations in the UI
  - Predefined components with tests that practitioners can use to construct pipelines "off-the-shelf

“Distribution Shift”

- No one really knows when they need to retrain their model...
  - Theorists: “it is impossible to know when data has changed if you don’t define strong assumptions”
  - DB researchers: “Sketches? AQP?”
  - ML engineers: `df.hist(); plt.show()`
- Maybe we need better (dynamic) data benchmarks?
- **Hard to find time series data streams with tractable ML tasks to do research on**

Optimizations

- Efficient artifact and model versioning
- gRPC integrations to support languages other than Python
- Asynchronous testing and logging

Miscellaneous

- `mltrace-demo` Github repository
- `mltrace` Github repository
- Arxiv preprint: Towards Observability for Machine Learning Pipelines
- Want to get involved? Have feedback?
  - Email shreyashankar@berkeley.edu
- Supported by the RISELab at UC Berkeley, Prof. Aditya Parameswaran, and undergrads Aditi Mahajan and Boyuan Deng