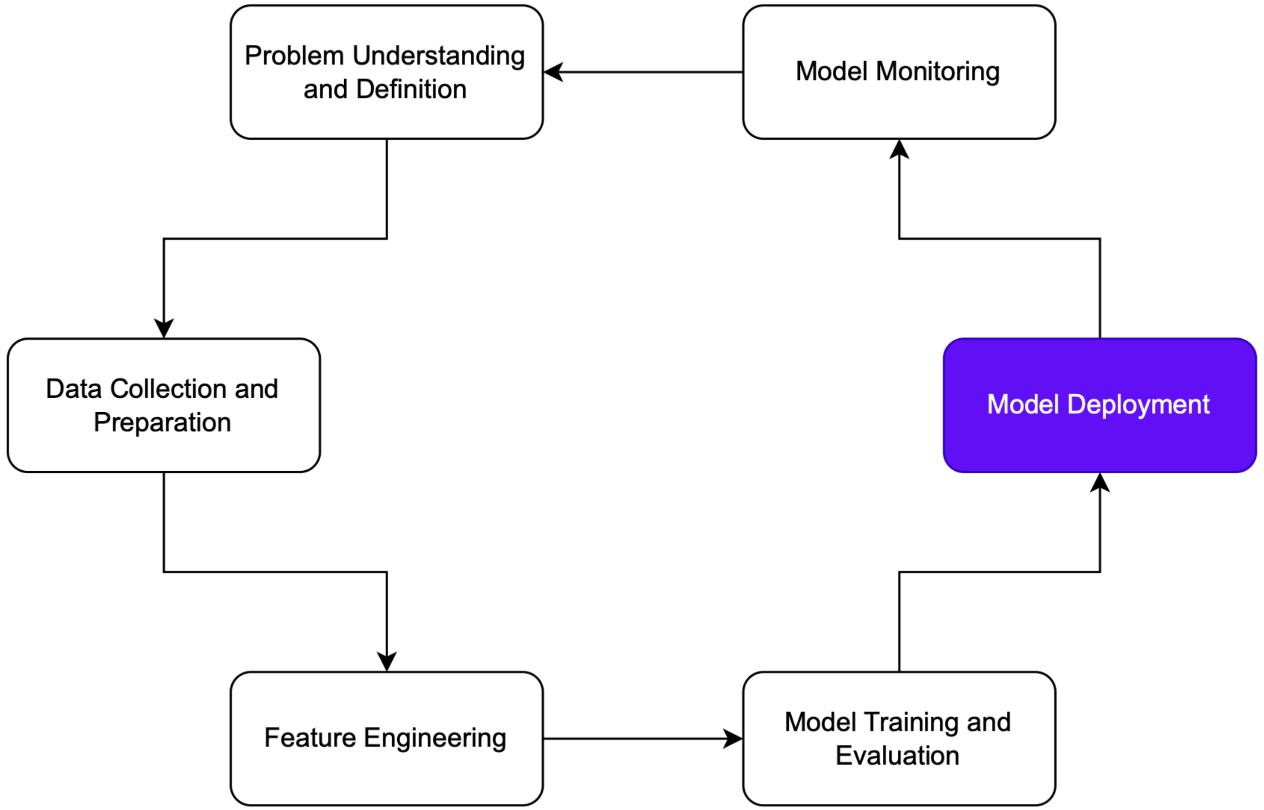
Testing a model

END-TO-END MACHINE LEARNING

Joshua Stapleton
Machine Learning Engineer

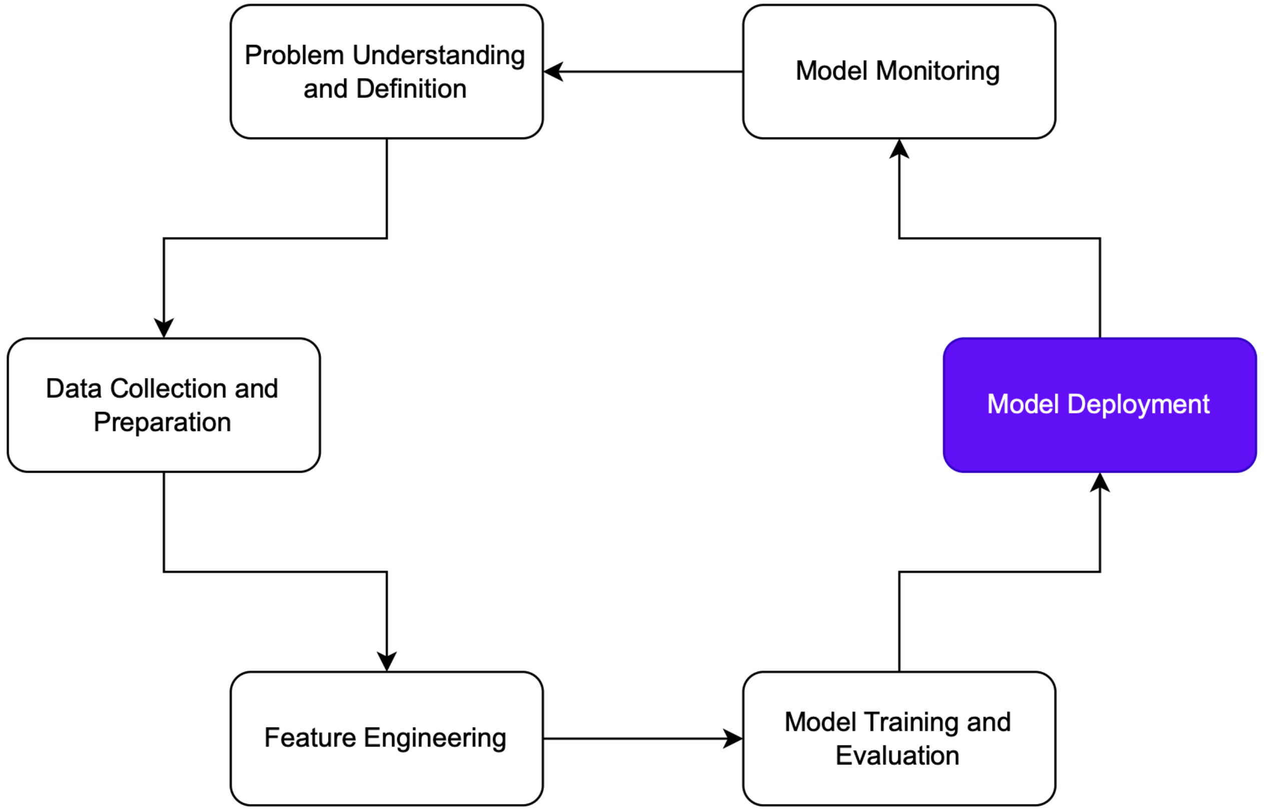
Deployment

- Next step in ML lifecycle
- After training and evaluation
- Make model available for use



Testing

- Testing:
  - Model does not crash
  - Returning reasonable outputs at inference time
  - Returning outputs in reasonable time



Unittest

- Testing
  - Flag anomalous / unexpected events.
  - Check model is performing as expected.

- `unittest`
  - Built-in Python library for test-writing.
  - Test case: covers type of test.
  - Test case method: single test for one aspect of test case.

Unittest usage

```python
import unittest

class TestModelInference(unittest.TestCase):
    def setUp(self):
        self.model = fitted_model
        self.X_test = X_test
    def test_prediction_output_shape(self):
        y_pred = self.model.predict(self.X_test)
        self.assertEqual(y_pred.shape[0], self.X_test.shape[0])

if __name__ == '__main__':
    unittest.main()
```

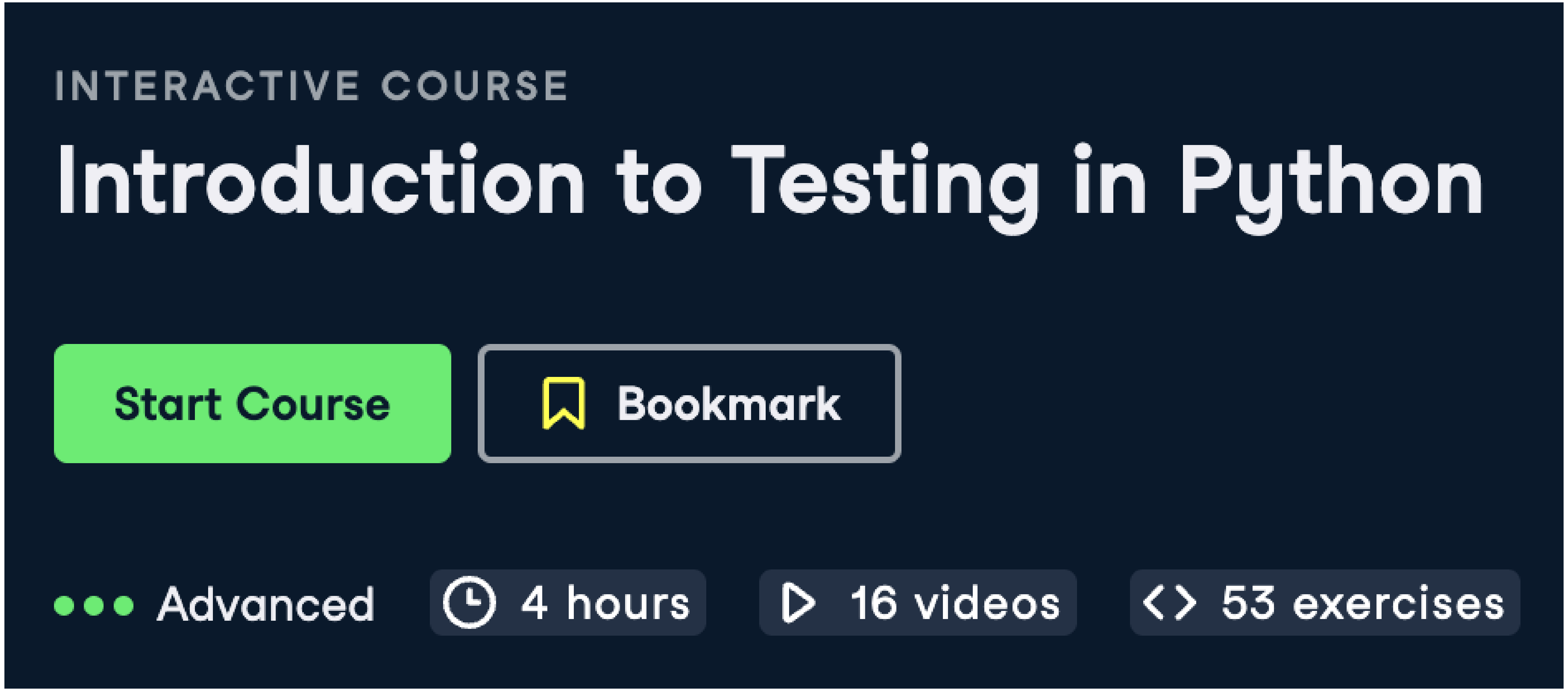
Unittest usage (cont.)

```
def test_input_values(self):
    print("Running test_input_values test case")

    # Get inputs (each row in testing set)
    for input in X_test:
        for value in input:
            # if value is cholestrol, for example:
            self.assertIn(value, [0, 500])
```

Testing in Python

Introduction to Testing in Python



Testing do's and dont's

Best-practices

- DON'T...
  - Write too many tests
  - Write redundant tests
  - Write tests for highly reliable components.
- DO...
  - Write tests to increase reliability.
  - Write tests to check/manage expectations.
  - Write tests for new functionality.

Testing benefits

Benefits of TDD

- Confidence in development
  - Stable iteration
  - Less worry about bugs
- Performance
  - Reliability
  - Production-grade

Let's practice!

END-TO-END MACHINE LEARNING

Architectural components in end-to-end machine learning frameworks

END-TO-END MACHINE LEARNING

Joshua Stapleton
Machine Learning Engineer

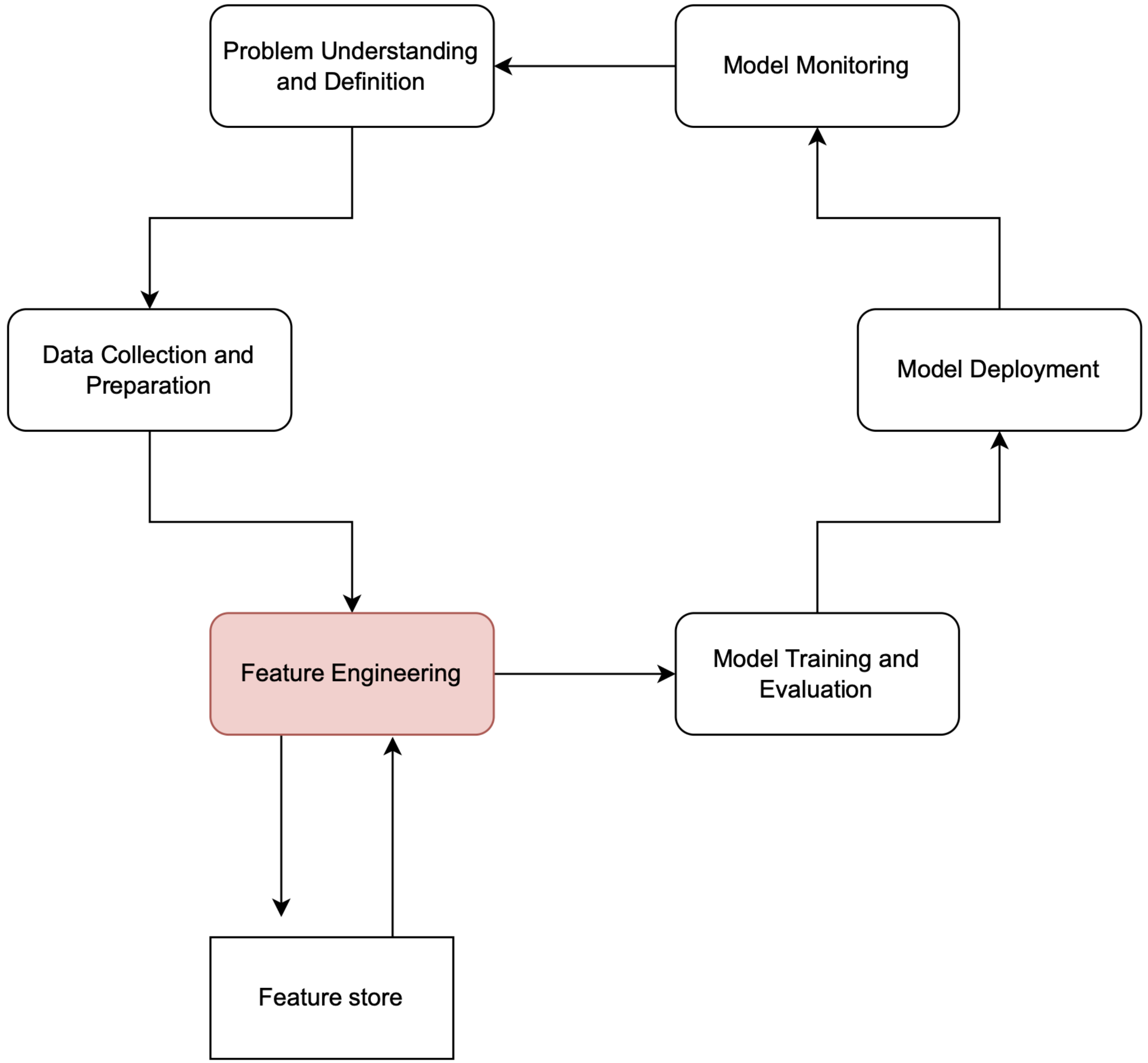
Feature stores

Features

- Feature selection
- Feature engineering

Feature store

- Central repository for features
- Ensures consistency, reduces duplication
- Enables sharing, discovery
- Standardizes feature transformations and calculations



Feast

Feast

- Popular tool for implementation of feature stores
- Provides unified management, storage, serving, and discovery for ML features

Principles

- Define, register features with feature sets
- Feature sets: grouping of related features + metadata

Example: heart disease features

- Patient entity
- Associated features (cholesterol, age, sex)

Feast feature stores part 1

```python
from feast import Field, Entity, ValueType, FeatureStore
from feast.data_source import FileSource

# Define the entity, which in this case is a patient, and features
patient = Entity(name="patient", join_keys=["patient_id"])
chol = Field(name="chol", dtype=Float32)
age = Field(name="age", dtype=Int32)
...
# Define the data source
data_source = FileSource(
    path="/path_to_heart_disease_dataset.csv",
    event_timestamp_column="event_timestamp",
    created_timestamp_column="created")
```

Feast feature stores part 2

```
# ... continued
# Create a feature view of the data
heart_disease_fv = FeatureView(name="heart_disease", entities=[patient],
    schema=[cholesterol, ...], ttl=timedelta(days=1), input=data_source,)

# Create a FeatureStore object
store = FeatureStore(repo_path=".")

# Register the FeatureView
store.apply([patient, heart_disease_fv])
```

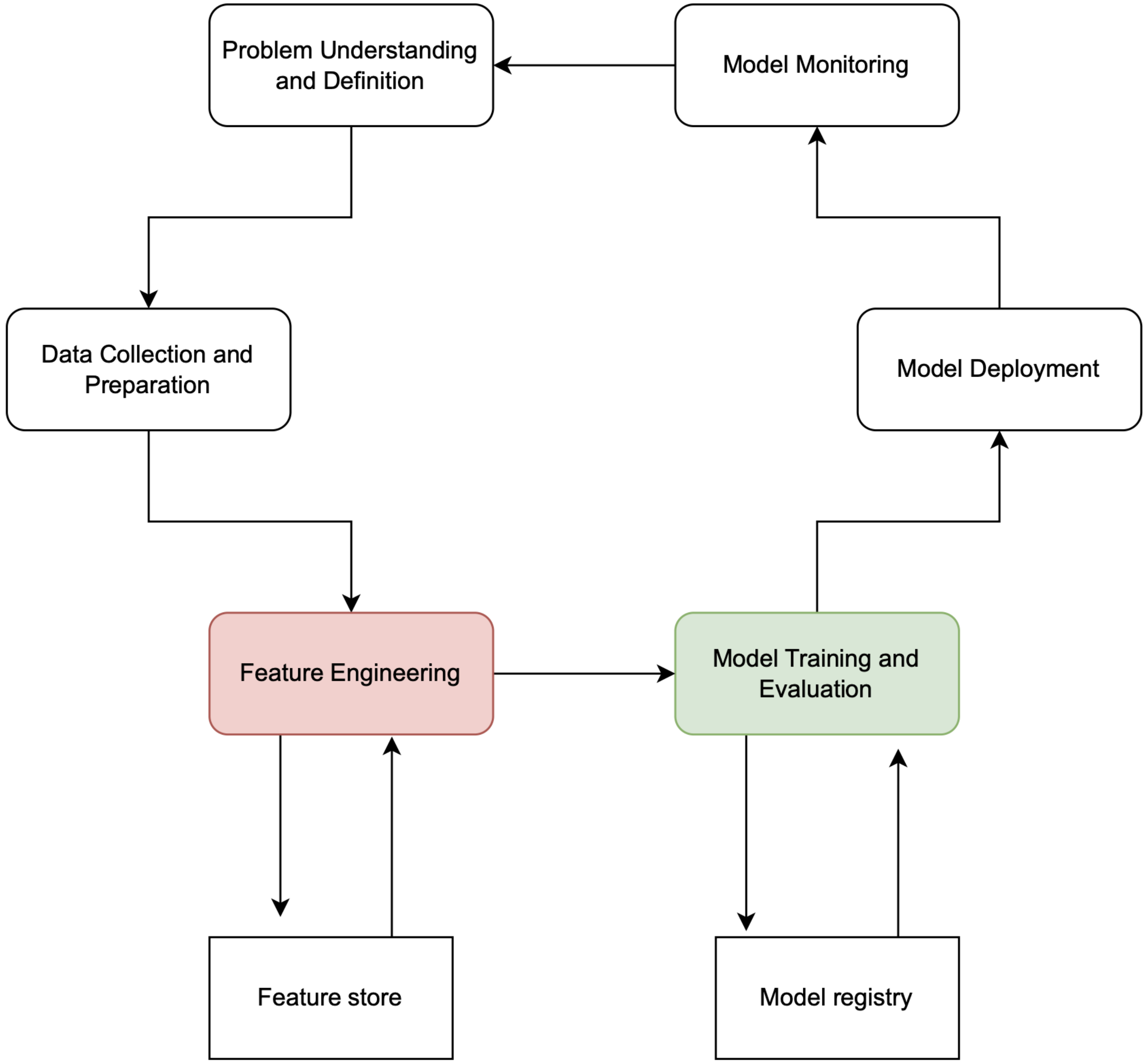
Model registries

Model registry

- Version control systems
- Keep track of different versions of model
- Annotate models
- Track performance over time

Benefits

- Organization
- Transparency
- Reproducibility



Let's practice!

END-TO-END MACHINE LEARNING

Packaging and containerization

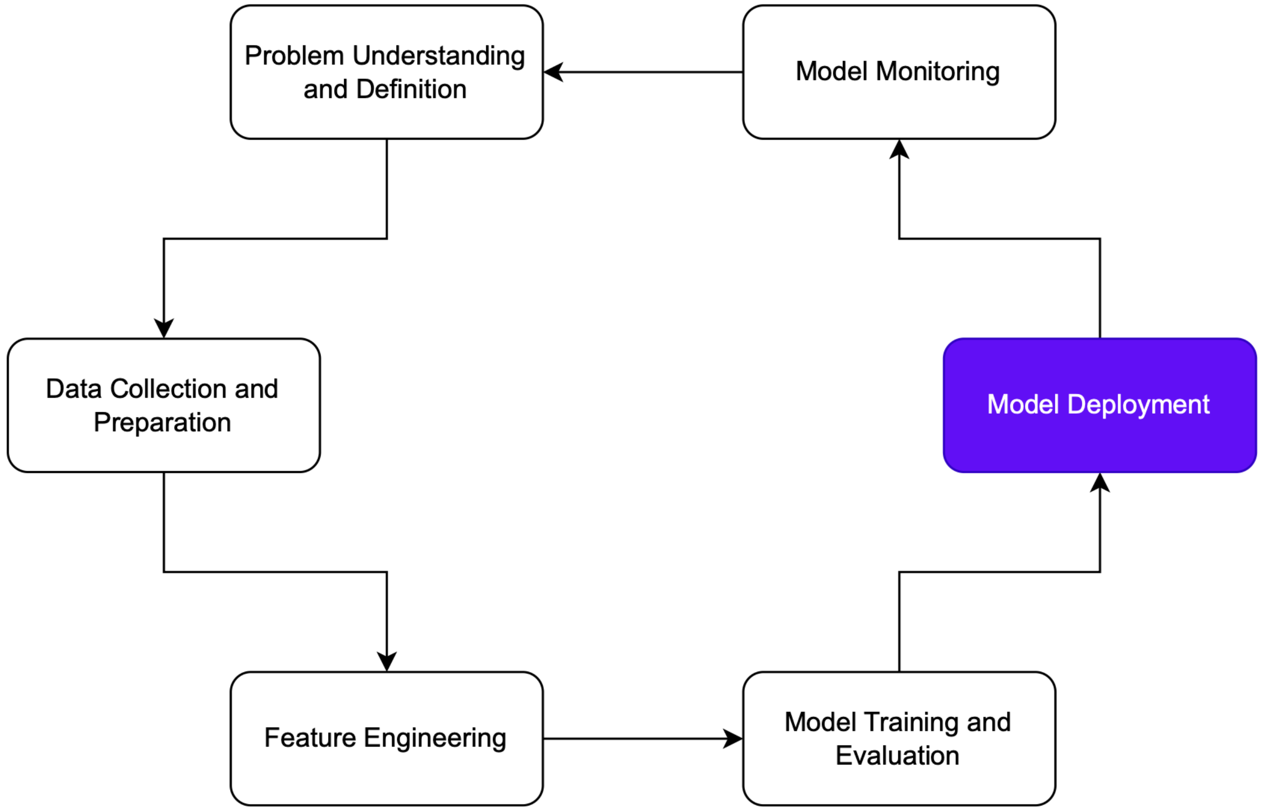
END-TO-END MACHINE LEARNING

Joshua Stapleton
Machine Learning Engineer

Deployment and containerization

Deployment

- Packing model + dependencies into units
- For running in different environment
- De-facto framework for containerization and deployment is Docker



Docker

- Platform for simplifying development using containers

Containers:

- Package application into standalone assets
- Containers designed as platform-agnostic
- Get Docker

DataCamp Docker Course

Docker usage part 1

Dockerfile: instructions for building container

```
# Use an official Python runtime as a parent image
FROM Python:3.7

# Set the working directory in the container to /app
WORKDIR /ML_pipeline

# Copy the current directory contents into the container at /app
ADD . /ML_pipeline

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt
```

Docker usage part 2

```
# ... continued
# Make port 80 available to the world outside this container
EXPOSE 80
```

```
# Define environment variable
ENV NAME World
```

```
# Run app.py when the container launches
CMD ["Python", "ML_pipeline.py"]
```

Build the defined image:

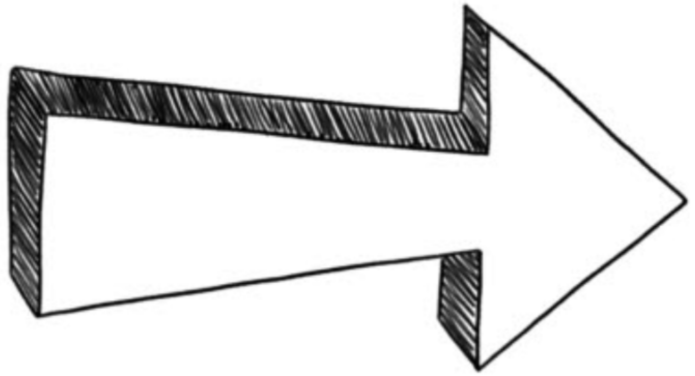
```
docker build -t heart_disease_model .
```

Tagging containers

Tagging:

```
docker tag heart_disease_model:latest heart_disease_model:1.0
```

- Makes images / containers easier to identify and manage.
- Helps in maintaining a detailed and robust model registry.
- After tagging, we are ready to deploy!



Best practices

While Docker makes packaging models easy...

- Be security-minded
- Don't include sensitive data
- Use trusted images (from verified developers)

If you application does have sensitive information...

- Use environment variables
- Eg: for connection strings/passwords

Let's practice!

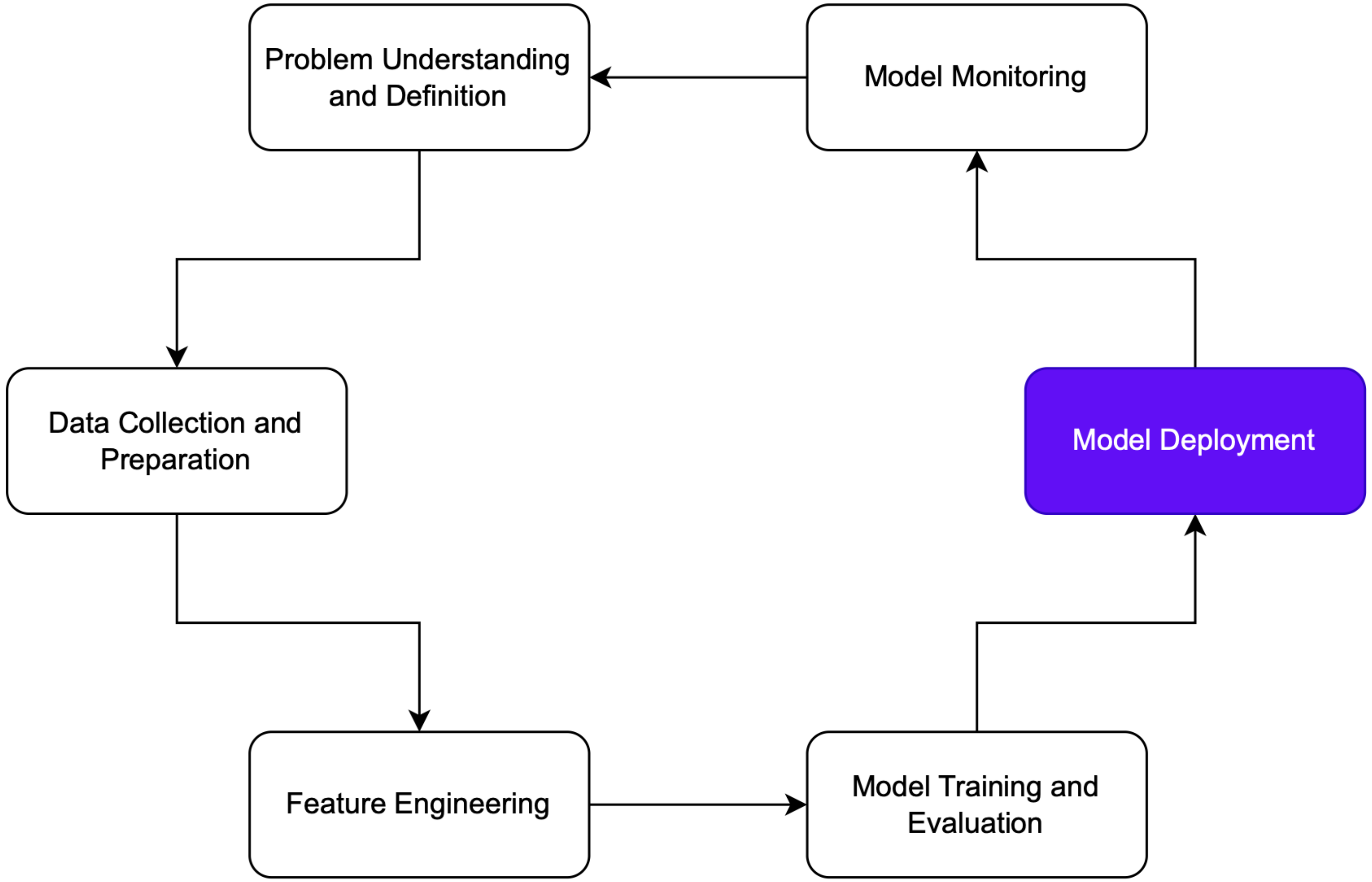
END-TO-END MACHINE LEARNING

Continuous integration and continuous deployment (CI/CD)

END-TO-END MACHINE LEARNING

Joshua Stapleton
Machine Learning Engineer

CI/CD in the ML lifecycle



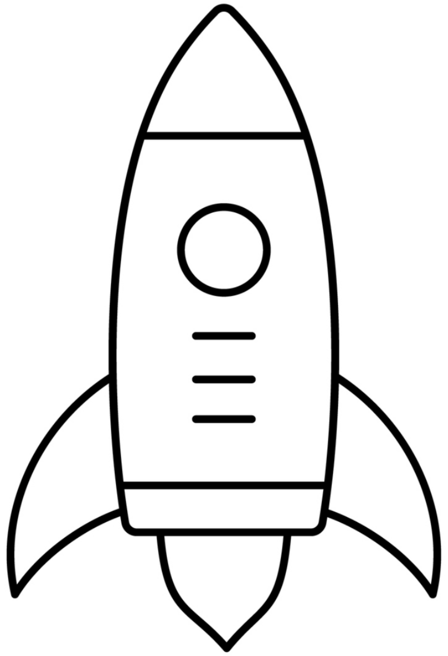
CI/CD principles

Continuous Integration(CI)

- Regularly merging to central repository
- Often involves automatic testing for identifying bugs

Continuous Deployment (CD)

- Automatically deploying updates in codebase to production
- Often combined with CI



CI/CD in machine learning

CI/CD is critical for production / iteration

- E.g.: automate including new patient data
- Helps to avoid data drift

CI/CD in ML:

- Regularly retrain models
- Testing performance
- Automated, rule-based deployment

CI/CD with AWS Elastic Beanstalk

AWS Elastic Beanstalk (EB):

- Fully managed service for deployment and scaling of applications + services
- Install EB

```
eb init

eb create heart_disease_env

eb deploy

eb open
```

Alternatives to EB (1)

Azure Machine Learning:

- Real-time scoring services
- Managed compute resources for training
- Performance monitoring in production environments

Alternatives to EB (2)

GCP App Engine:

- Similar alternative to AWS EB or Azure Machine Learning

Alternatives to EB (3)

Kubernetes:

- Open-source container orchestration system
- Automates deployment, scaling, management of containerized applications
- Compatibility with multiple major cloud platforms
- Steeper learning curve, but offers greater control

Alternatives to EB (4)

Many, many more!

Let's practice!

END-TO-END MACHINE LEARNING