

# Model reliability

DEVELOPING MACHINE LEARNING MODELS FOR PRODUCTION

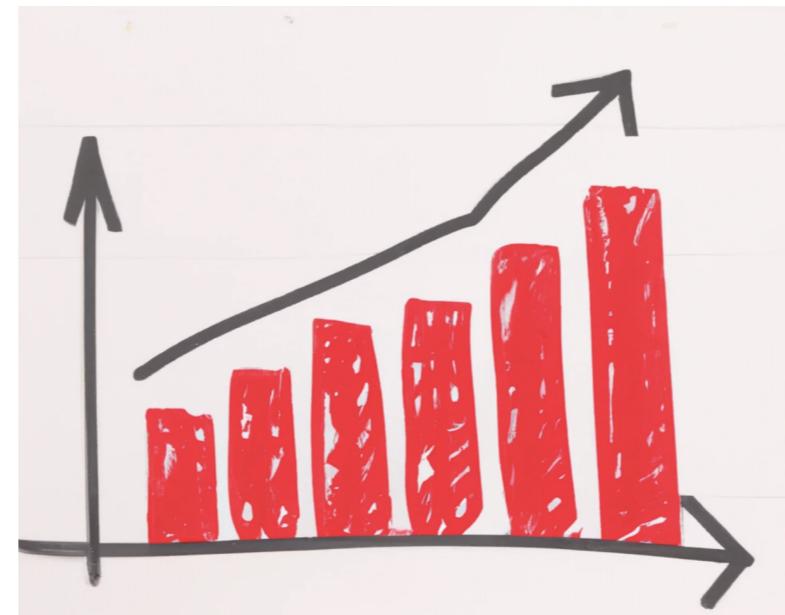


**Sinan Ozdemir**

Data Scientist, Entrepreneur, and Author

# Aligning ML models with business impact metrics

- Business impact metrics measure the impact of ML on the business
  - e.g. revenue, cost savings, customer satisfaction score (CSAT)
- Should be aligned with model metrics
  - e.g. churn predictor --> revenue
  - e.g. Manufacturing maintenance predictor --> cost savings
  - e.g. accuracy of chatbot intent detection --> CSAT



# Testing routines in ML pipelines

- **Unit tests** test individual components
  - e.g. test that a PCA instance returns the expected number of features
- **Integration tests** consider the entire pipeline
  - e.g. test that the input data is correctly preprocessed, the model makes accurate predictions, and the output is correctly post-processed
- **Smoke tests** are quick tests that give you confidence that the system is working
  - e.g. test that the model can correctly classify a small set of sample images
- Test early and test often
  - e.g. test the model on new data as soon as it becomes available

# Example unit test

```
def test_pipeline():
    # Generate mock data for testing
    X_train = pd.DataFrame({'age': [25, 30, 35, 40], 'income': [50000, 60000, 70000, 80000]})
    y_train = pd.Series([0, 0, 1, 1])

    pipeline = Pipeline([('preprocessing', DataPreprocessor()), # Set up pipeline
                         ('model', LogisticRegression())])
    pipeline.fit(X_train, y_train) # Fit pipeline on training data

    # Generate mock data for testing
    X_test = pd.DataFrame({'age': [30, 35, 40, 45], 'income': [55000, 65000, 75000, 85000]})
    y_test = pd.Series([0, 0, 1, 1])
    y_pred = pipeline.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred) # Evaluate pipeline on test data

    assert accuracy > 0.8, "Error: pipeline accuracy is too low."
```

# Monitoring model staleness

- **Model staleness** - model's performance decreases over time
  - change in data or environment
- Continuous Monitoring!



# Identifying and addressing model staleness

## Identifying

- Monitoring model performance
- Monitor changes in data + environment

## Addressing

- Re-training the model on new data
  - e.g. New feature needs to be included in the model
- Update data pipeline to account for environment changes
  - e.g. Changes in analytics platforms confuses your pipeline

# **Let's practice!**

**DEVELOPING MACHINE LEARNING MODELS FOR PRODUCTION**

# Testing data

DEVELOPING MACHINE LEARNING MODELS FOR PRODUCTION



**Sinan Ozdemir**

Data Scientist, Entrepreneur, and Author

# Data validation and schema tests

- **Data validation tests** look for missing values, inconsistencies, and abnormal data
  - E.g. detecting outliers
- **Schema tests** look for expected data formats and data types
  - E.g. make sure "time to value" is an integer measuring seconds and not minutes
- Tools like *Great Expectations* help automate this process



# Beyond simple testing

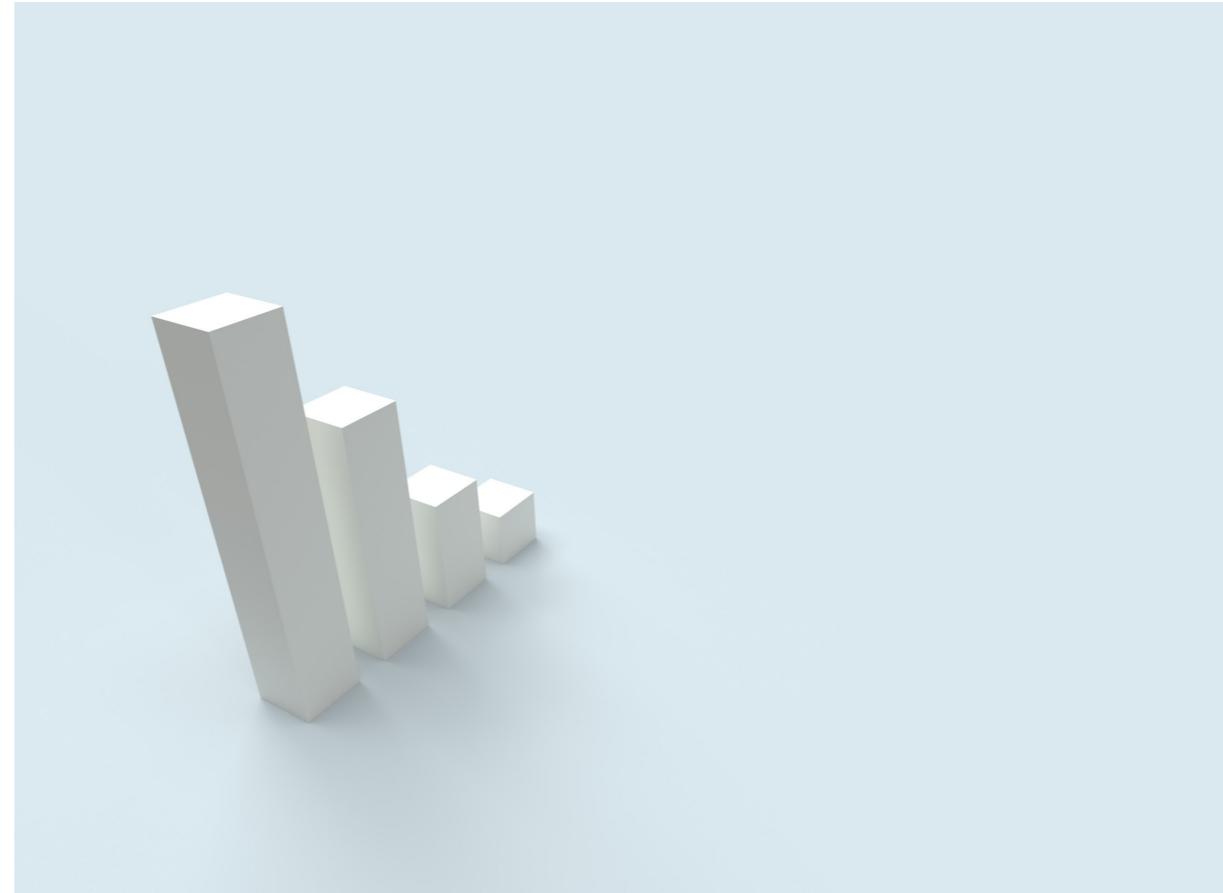
- Data and schema tests check for basic issues
- More complicated tests like **expectation tests** look for more complicated issues
  - Checking if values are in a certain range
  - Checking for known patterns/trends

# Expectation tests

- A type of data validation test
- Make sure data matches certain "expectations" as defined by the user or system
  - E.g. expecting time spent on website to be around 4 minutes with a standard deviation of 1 minute
  - E.g. expecting dates of a patient's past visits to be before the current time

# Feature importance tests

- Feature importance tests identify the most important features in an ML model
- Example: *Permutation importance*
  - Randomly permute feature values and measures how much model performance decreases



# Example of permutation importance

Set up:

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.inspection import permutation_importance
# Train a random forest classifier (assuming we have some data)
model = RandomForestClassifier().fit(X_train, y_train)
```

Running our permutation importance test:

```
# Calculate feature importances using permutation importance
results = permutation_importance(model, X_test, y_test, n_repeats=10, random_state=42)
# Print the feature importances
feature_names = ['feature_1', 'feature_2', 'feature_3', ...]
importances = results.importances_mean
for i in range(len(feature_names)):
    print(f'{feature_names[i]}: {importances[i]}')
```

# Looking for data drift

## Data drift

- Sometimes called feature drift
- A change in the distribution of a model's input data
- E.g. more people are using a new term that corresponds to a new product and a chatbot is scratching it's head

## Label drift

- A change in the label distribution
- E.g. A shift away from people asking for returns and they shift to people asking the status of their returns instead

# **Let's practice!**

**DEVELOPING MACHINE LEARNING MODELS FOR PRODUCTION**

# Testing models

DEVELOPING MACHINE LEARNING MODELS FOR PRODUCTION



**Sinan Ozdemir**

Data Scientist, Entrepreneur, and Author

# Individual vs Group fairness

## Individual fairness

Treat similar individuals similarly.

- E.g. Give similar job opportunities to individuals with similar experiences.

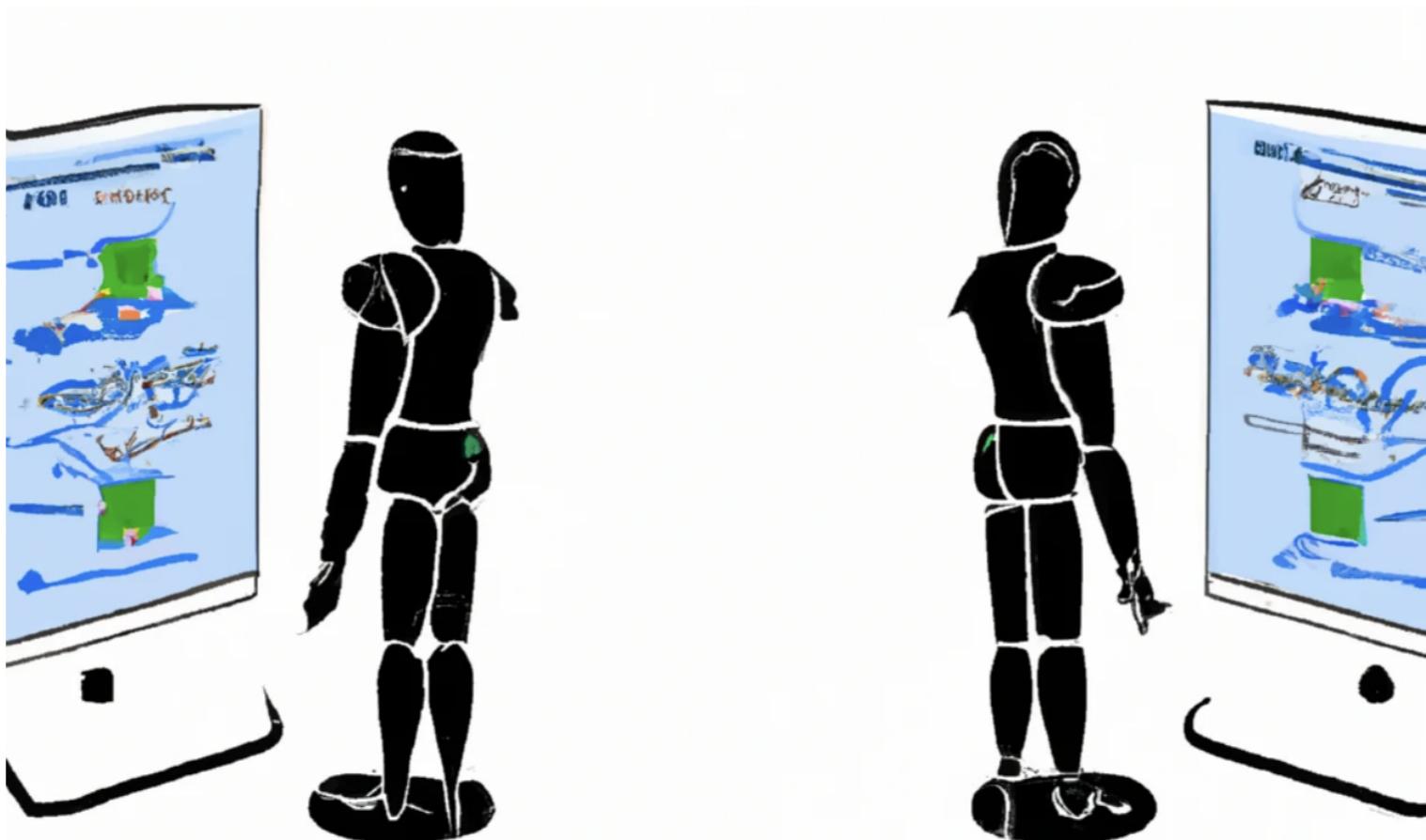
## Group fairness

Treat different groups equally.

- E.g. Do not discriminate against individuals from certain racial or ethnic groups in school admittance models.

# Holdout testing

- Holdout testing checks model reliability using a separate dataset
- Assesses performance on untrained data, like at training time
- Detects issues like overfitting or underfitting



# Looking for model drift

## Concept drift

- A shift in the relationship between the features and the response
  - E.g. a sentiment analysis algorithm
    - Terms can shift in meaning (saying something is "sick" is a good thing)

## Prediction drift

- A shift in a model's prediction distribution
  - E.g. a temporary outage leads to an increase to the "outage" intent for your chatbot
  - Nothing is "wrong" per say but this drift could lead to slower response times for users

# Example of looking for model drift

Setup:

```
# Import necessary libraries and split data
X_train, X_test, y_train, y_test = ...
```

Fitting our model:

```
# Train a classifier on the training data
clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)

# Calculate the accuracy on test data
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

Testing for drift later:

```
X_test_drift = X_test + 1.0 # Simulate data drift
# Calculate the accuracy on drifted data
y_pred_drift = clf.predict(X_test_drift)
accuracy_drift = accuracy_score(
    y_test, y_pred_drift)
print(f"Accuracy with drift: {accuracy_drift}")

# Drift detection threshold based on the accuracy
drift_threshold = accuracy * 0.9

# Check for drop in accuracy on the drifted data
if accuracy_drift < drift_threshold:
    print("Concept drift detected!")
else:
    print("No concept drift detected.")
```

# Cost of complex models vs baseline models

- Complex models are more expensive and can affect model efficiency
- **Latency** measures the time to process a single input and generate a prediction
- **Throughput** measures the number of predictions a model can make in a given time
- Test latency and throughput to identify how much complexity a model should have
- Goal is a balance between accuracy and efficiency

# **Let's practice!**

**DEVELOPING MACHINE LEARNING MODELS FOR PRODUCTION**

# Congratulations!

DEVELOPING MACHINE LEARNING MODELS FOR PRODUCTION

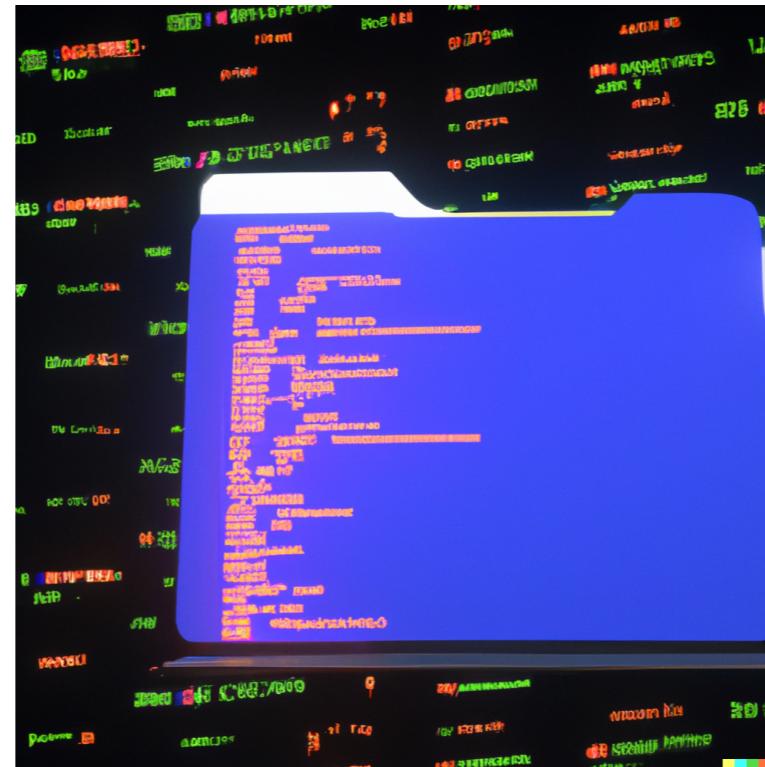


**Sinan Ozdemir**

Data Scientist, Entrepreneur, and Author

# Move from experiment to production

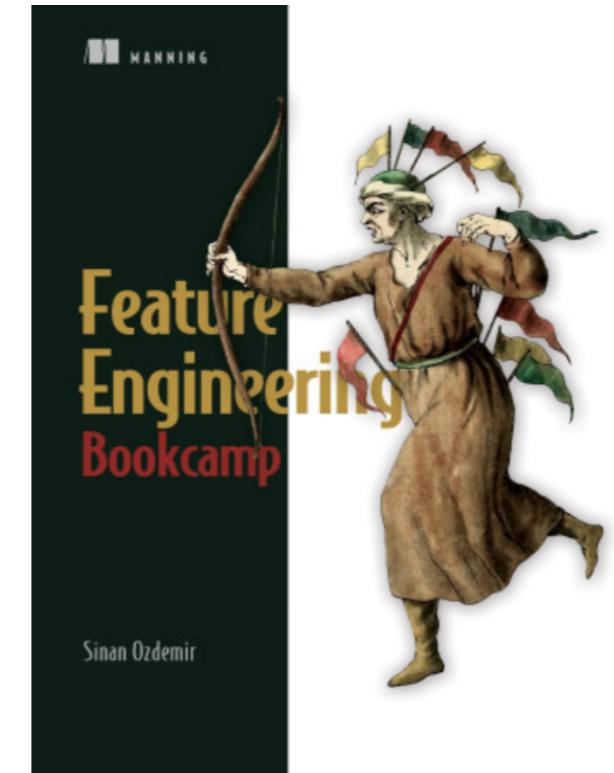
- Identify what makes an ML experiment ready to be moved into production
- Recognize the causes of technical debt



- Write effective documentation and maintainable code
- Effective documentation and maintainable code lead to reduced technical debt

# Ensure reproducibility

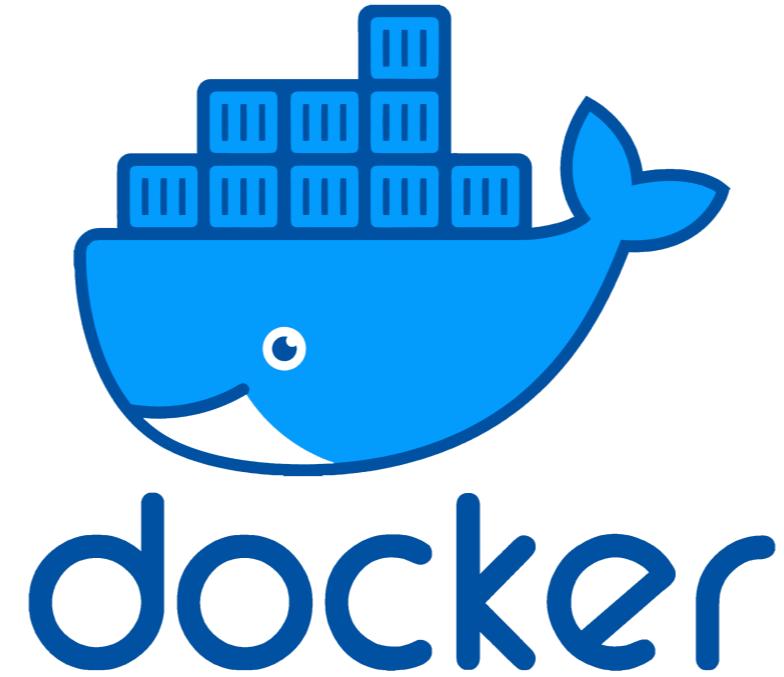
- Ensure reproducibility in machine learning
- Perform feature engineering to optimize machine learning pipeline performance



- Design reproducible experiments
- Implement data and model versioning to track changes

# Deploying ML models

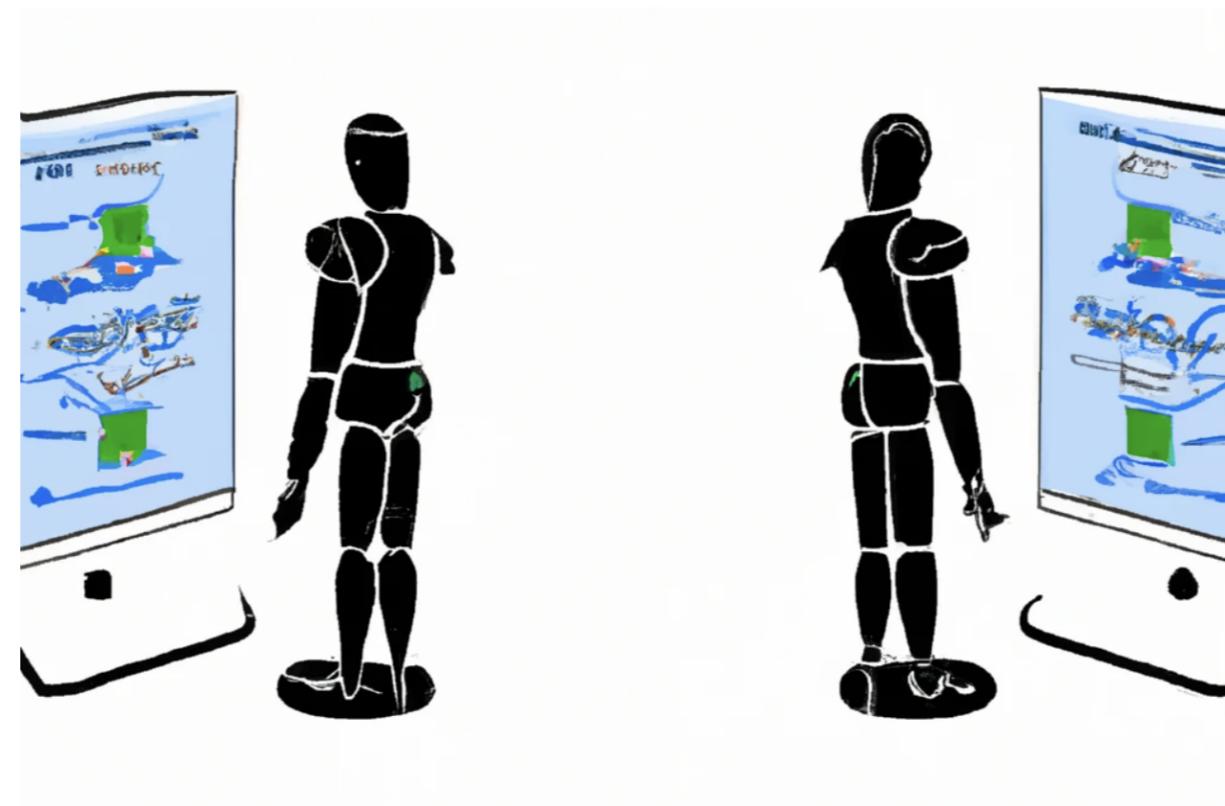
- Package ML models and environments for deployment in different frameworks
- Ensure scalability of machine learning models to handle larger workloads



- Four MLOps principles: Continuous Integration, Continuous Delivery, Continuous Training, and Continuous Monitoring

# Test + Evaluate

- Different types of testing routines - unit tests, integration tests, smoke tests



- Evaluate model reliability to ensure that the model produces accurate results
- Test models to identify potential issues and improve performance

# Keep going!

- Apply what you have learned in this course to a real-world project
- Ensure reproducibility, scalability, reliability, and monitor for data drift
- Set up an experimentation framework for A/B testing your model
- Regularly re-train and update your model to ensure it remains relevant and accurate over time
- Keep learning and exploring new developments in the field of machine learning
- Continue to improve your skills and expertise

# Let's go!

DEVELOPING MACHINE LEARNING MODELS FOR PRODUCTION