

# Testing & CI/CD Lab

---

**CS 203: Software Tools and Techniques for AI**

Duration: 3 hours

# Lab Overview

---

By the end of this lab, you will:

- Write unit tests with pytest
- Test ML models and data pipelines
- Set up GitHub Actions CI/CD
- Implement pre-commit hooks
- Create automated testing workflows

## Structure:

- Part 1: Unit Testing (45 min)
- Part 2: ML Testing (60 min)
- Part 3: CI/CD Setup (60 min)
- Part 4: Advanced Testing (15 min)

# Setup (10 minutes)

---

```
# Install testing tools
pip install pytest pytest-cov pytest-mock hypothesis great-expectations

# Install pre-commit
pip install pre-commit black flake8

# Create project structure
mkdir -p ml-project/{src,tests,data,.github/workflows}
cd ml-project

# Initialize git
git init
```

# Exercise 1.1: First Unit Test (10 min)

---

Create `src/utils.py` :

```
def preprocess_text(text):
    if text is None:
        raise ValueError("Text cannot be None")
    return text.lower().strip()

def calculate_accuracy(y_true, y_pred):
    if len(y_true) != len(y_pred):
        raise ValueError("Arrays must have same length")
    return sum(y_true == y_pred) / len(y_true)
```

Create `tests/test_utils.py` :

```
import pytest
from src.utils import preprocess_text, calculate_accuracy

def test_preprocess_text():
    assert preprocess_text(" Hello World ") == "hello world"
    assert preprocess_text("UPPERCASE") == "uppercase"
```

# Exercise 1.2: Fixtures (15 min)

---

Add to `tests/test_utils.py`:

```
import pandas as pd
import numpy as np

@pytest.fixture
def sample_data():
    return pd.DataFrame({
        'text': ['hello', 'world', 'test'],
        'label': [0, 1, 0]
    })

@pytest.fixture
def model():
    from sklearn.linear_model import LogisticRegression
    return LogisticRegression(random_state=42)

def test_data_shape(sample_data):
    assert sample_data.shape == (3, 2)
    assert list(sample_data.columns) == ['text', 'label']
```

# Exercise 1.3: Parametrized Tests (10 min)

```
@pytest.mark.parametrize("input,expected", [
    ("Hello", "hello"),
    ("WORLD", "world"),
    (" Test ", "test"),
    ("123", "123"),
    ("MiXeD CaSe", "mixed case"),
])

def test_preprocess_parametrized(input, expected):
    assert preprocess_text(input) == expected

@pytest.mark.parametrize("y_true,y_pred,expected", [
    ([1,1,1], [1,1,1], 1.0),
    ([1,0,1], [0,0,0], 0.333),
    ([1,1,0,0], [1,0,1,0], 0.5),
])

def test_accuracy_parametrized(y_true, y_pred, expected):
    acc = calculate_accuracy(np.array(y_true), np.array(y_pred))
    assert acc == pytest.approx(expected, 0.01)
```

Run: `pytest -v`

# Exercise 2.1: Test Data Loading (15 min)

---

Create `src/data.py` :

```
import pandas as pd

def load_data(filepath):
    df = pd.read_csv(filepath)
    if df.empty:
        raise ValueError("Dataset is empty")
    return df

def validate_schema(df, required_columns):
    missing = set(required_columns) - set(df.columns)
    if missing:
        raise ValueError(f"Missing columns: {missing}")
    return True
```

Create `tests/test_data.py` :

```
import pytest
import pandas as pd
```

# Exercise 2.2: Test Model Training (20 min)

---

Create `src/model.py` :

```
from sklearn.ensemble import RandomForestClassifier
import numpy as np

def train_model(X, y, **kwargs):
    if len(X) == 0:
        raise ValueError("Training data is empty")

    model = RandomForestClassifier(random_state=42, **kwargs)
    model.fit(X, y)
    return model

def evaluate_model(model, X, y):
    return model.score(X, y)
```

Create `tests/test_model.py` :

```
import pytest
import numpy as np
```



# Exercise 2.3: Test Model Robustness (15 min)

Add to `tests/test_model.py` :

```
def test_model_predictions_valid(dummy_data):
    X, y = dummy_data
    model = train_model(X, y)
    predictions = model.predict(X)

    # Check predictions are valid
    assert len(predictions) == len(X)
    assert set(predictions).issubset(set(y))

def test_model_probabilities_sum_to_one(dummy_data):
    X, y = dummy_data
    model = train_model(X, y)
    probs = model.predict_proba(X)

    # Probabilities should sum to 1
    np.testing.assert_array_almost_equal(probs.sum(axis=1), np.ones(len(X)))

def test_model_deterministic(dummy_data):
    X, y = dummy_data
    model1 = train_model(X, y)
    model2 = train_model(X, y)

    pred1 = model1.predict(X[:10])
    pred2 = model2.predict(X[:10])

    np.testing.assert_array_equal(pred1, pred2)
```

## Exercise 2.4: Test for Bias (10 min)

---

```
def test_model_class_balance_sensitivity(dummy_data):  
    """Model shouldn't be too sensitive to class imbalance."""  
    X, y = dummy_data  
  
    # Create imbalanced dataset  
    mask = (y == 0)  
    X_imb = np.vstack([X[mask][:10], X[~mask][:90]])  
    y_imb = np.hstack([y[mask][:10], y[~mask][:90]])  
  
    model = train_model(X_imb, y_imb)  
  
    # Model shouldn't always predict majority class  
    predictions = model.predict(X_imb)  
    minority_predicted = (predictions == 0).sum()  
  
    assert minority_predicted > 0, "Model ignoring minority class"
```

# Exercise 3.1: GitHub Actions Setup (20 min)

---

Create `.github/workflows/test.yml`:

```
name: Tests

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.10'

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
          pip install pytest pytest-cov

      - name: Run tests
        run: |
          pytest tests/ -v --cov=src --cov-report=xml

      - name: Upload coverage
        uses: codecov/codecov-action@v3
        with:
          file: ./coverage.xml
```

# Exercise 3.2: Linting Workflow (15 min)

---

Create `.github/workflows/lint.yml`:

```
name: Lint

on: [push, pull_request]

jobs:
  lint:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.10'

      - name: Install linters
        run: pip install black flake8 mypy

      - name: Run black
        run: black --check src/ tests/

      - name: Run flake8
        run: flake8 src/ tests/ --max-line-length=88

      - name: Run mypy
        run: mypy src/ --ignore-missing-imports
```

# Exercise 3.3: Pre-commit Hooks (15 min)

Create `.pre-commit-config.yaml` :

```
repos:
- repo: https://github.com/psf/black
  rev: 23.3.0
  hooks:
    - id: black
      language_version: python3.10

- repo: https://github.com/pycqa/flake8
  rev: 6.0.0
  hooks:
    - id: flake8
      args: [--max-line-length=88]

- repo: https://github.com/pre-commit/pre-commit-hooks
  rev: v4.4.0
  hooks:
    - id: trailing-whitespace
    - id: end-of-file-fixer
    - id: check-yaml
    - id: check-added-large-files

- repo: local
  hooks:
    - id: pytest-check
      name: pytest-check
      entry: pytest tests/ -v
      language: system
      pass_filenames: false
      always_run: true
```

# Exercise 3.4: Model Training Workflow (10 min)

Create `.github/workflows/train.yml` :

```
name: Train Model

on:
  workflow_dispatch:
    inputs:
      config:
        description: 'Config file'
        required: true
        default: 'configs/default.yaml'

jobs:
  train:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.10'

      - name: Install dependencies
        run: pip install -r requirements.txt

      - name: Train model
        run: python train.py --config ${github.event.inputs.config}

      - name: Upload model
        uses: actions/upload-artifact@v3
        with:
          name: model
          path: models/
          retention-days: 30
```

# Exercise 4.1: Property-Based Testing (15 min)

```
pip install hypothesis
```

Create advanced tests:

```
from hypothesis import given, strategies as st
import numpy as np

@given(st.lists(st.floats(min_value=-100, max_value=100), min_size=1))
def test_preprocess_preserves_length(texts):
    texts_str = [str(t) for t in texts]
    processed = [preprocess_text(t) for t in texts_str]
    assert len(processed) == len(texts_str)

@given(
    st.lists(st.integers(0, 1), min_size=10),
    st.lists(st.integers(0, 1), min_size=10)
)
def test_accuracy_bounds(y_true, y_pred):
    y_true = np.array(y_true[:10])
    y_pred = np.array(y_pred[:10])
```

# Deliverables

---

Submit:

1. GitHub repository with:
  - Complete test suite (unit, integration)
  - GitHub Actions workflows
  - Pre-commit configuration
  - Requirements.txt
2. Test coverage report (>70%)
3. CI/CD pipeline that:
  - Runs tests on push
  - Checks code quality
  - Trains model (optional)
4. README with:
  - Setup instructions



# Testing Checklist

---

- ☐ Unit tests for all functions
- ☐ Data loading/validation tests
- ☐ Model training tests
- ☐ API endpoint tests (if applicable)
- ☐ Test coverage >70%
- ☐ GitHub Actions workflows working
- ☐ Pre-commit hooks installed
- ☐ All tests passing
- ☐ Code formatted with black
- ☐ No flake8 errors

# Resources

---

- pytest: <https://docs.pytest.org/>
- GitHub Actions: <https://docs.github.com/en/actions>
- pre-commit: <https://pre-commit.com/>
- Hypothesis: <https://hypothesis.readthedocs.io/>

# Excellent Work!

---

Next: Model Deployment

Questions? Office hours tomorrow 3-5 PM