CDW Documentation

Github ML Pipeline

Github ML Pipeline

Github repository structure

screenshot_2025-08-08_at_10.37.34 am

mlops-pipeline.yml

```
name: MLOps CI/CD Pipeline

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

env:
  AZURE_ML_WORKSPACE: ${{ secrets.AZURE_ML_WORKSPACE }}
  AZURE_RESOURCE_GROUP: ${{ secrets.AZURE_RESOURCE_GROUP }}
  AZURE_SUBSCRIPTION_ID: ${{ secrets.AZURE_SUBSCRIPTION_ID }}
  MODEL_NAME: "customer-churn-model"
  ENDPOINT_NAME: "churn-prediction-endpoint"

permissions:
  id-token: write
  contents: read

jobs:
  unit-tests:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.9'

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt
        pip install pytest pytest-cov flake8
```

```
    - name: Run linting
      run: |
        flake8 src/ --count --select=E9,F63,F7,F82 --show-source --
statistics
        flake8 src/ --count --exit-zero --max-complexity=10 --max-line-
length=127 --statistics

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

    - name: Upload coverage reports
      uses: codecov/codecov-action@v3
      with:
        file: ./coverage.xml
        flags: unittests

  build-and-train:
    needs: unit-tests
    runs-on: ubuntu-latest
    if: github.event_name == 'push' && github.ref == 'refs/heads/main'
    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.9'

    - name: Install Azure CLI and ML extension
      run: |
        curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash
        az extension add -n ml
    - name: Azure Login
      uses: azure/login@v2
      with:
        client-id: ${{ secrets.AZURE_CLIENT_ID }}
        tenant-id: ${{ secrets.AZURE_TENANT_ID }}
        subscription-id: ${{ secrets.AZURE_SUBSCRIPTION_ID }}
    # auth-type: OIDC is implied in v2 when no client-secret is provided

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

    - name: Set Azure ML workspace
      run: |
        az configure --defaults group=$AZURE_RESOURCE_GROUP
```

```
workspace=$AZURE_ML_WORKSPACE

    - name: Create compute cluster if not exists
      run: |
        az ml compute create --name cpu-cluster --type amlcompute --min-
instances 0 --max-instances 4 --size Standard_DS3_v2 || true

    - name: Submit training job
      id: training
      run: |
        RUN_ID=$(az ml job create --file ml/training-job.yml --query name -o
tsv)
        echo "run_id=$RUN_ID" >> $GITHUB_OUTPUT
        echo "Training job submitted with ID: $RUN_ID"
        # Wait for job completion with timeout
        timeout 1800 az ml job stream --name $RUN_ID || {
          echo "Training job timed out or failed"
          az ml job show --name $RUN_ID --query status
          exit 1
        }

    - name: Get training metrics
      run: |
        az ml job show --name ${{ steps.training.outputs.run_id }} --query
tags

    - name: Debug job outputs
      run: |
        echo "Checking job outputs..."
        az ml job show --name ${{ steps.training.outputs.run_id }} --query
outputs
        echo "Listing job artifacts..."
        az ml job download --name ${{ steps.training.outputs.run_id }} --
output-name model_output --download-path ./debug_model --all || echo
"Download failed, checking alternative paths"

    - name: Debug job outputs and register model
      id: register
      run: |
        echo "=== Debugging Job Outputs ==="
        JOB_ID=${{ steps.training.outputs.run_id }}
        echo "Job ID: $JOB_ID"
        # Check job details
        echo "Job details:"
        az ml job show --name $JOB_ID --query "{status: status, outputs:
outputs}" -o json
        # List all outputs
        echo "Available outputs:"
        az ml job show --name $JOB_ID --query "outputs" -o json
        # Check if there's a model_output
        echo "Checking model_output specifically:"
```

```
        az ml job show --name $JOB_ID --query "outputs.model_output" -o json
|| echo "model_output not found"
        # Get MLflow run ID if available
        echo "Getting MLflow run ID:"
        MLFLOW_RUN_ID=$(az ml job show --name $JOB_ID --query
"properties.mlflow.runId" -o tsv 2>/dev/null || echo "not_found")
        echo "MLflow Run ID: $MLFLOW_RUN_ID"
        # Try to register model with correct path
        echo "=== Attempting Model Registration ==="
        if [ "$MLFLOW_RUN_ID" != "not_found" ] && [ ! -z "$MLFLOW_RUN_ID" ];
then
          echo "Trying MLflow run path..."
          MODEL_VERSION=$(az ml model create \
            --name $MODEL_NAME \
            --path "runs:/$MLFLOW_RUN_ID/model" \
            --type mlflow_model \
            --description "Customer churn prediction model trained via
GitHub Actions" \
            --query version -o tsv 2>/dev/null || echo "mlflow_failed")
        else
          MODEL_VERSION="mlflow_failed"
        fi
        if [ "$MODEL_VERSION" = "mlflow_failed" ] || [ -z "$MODEL_VERSION"
]; then
          echo "MLflow path failed, trying job output path..."
          MODEL_VERSION=$(az ml model create \
            --name $MODEL_NAME \
            --path "azureml://jobs/$JOB_ID/outputs/model_output" \
            --type mlflow_model \
            --description "Customer churn prediction model trained via
GitHub Actions" \
            --query version -o tsv 2>/dev/null || echo "job_output_failed")
        fi
        if [ "$MODEL_VERSION" = "job_output_failed" ] || [ -z
"$MODEL_VERSION" ]; then
          echo "Job output path failed, checking what files actually
exist..."
          # Try to download and check the structure
          mkdir -p ./debug_download
          az ml job download --name $JOB_ID --download-path ./debug_download
--all || echo "Download failed"
          echo "Downloaded structure:"
          find ./debug_download -type f -name "*.pkl" -o -name "MLmodel" -o
-name "*.json" | head -20
          # Look for any MLmodel files which indicate MLflow model format
          MLMODEL_PATH=$(find ./debug_download -name "MLmodel" | head -1)
          if [ ! -z "$MLMODEL_PATH" ]; then
            MODEL_DIR=$(dirname "$MLMODEL_PATH")
            RELATIVE_PATH=${MODEL_DIR#./debug_download/}
            echo "Found MLmodel at: $MLMODEL_PATH"
            echo "Relative path: $RELATIVE_PATH"
```

```
            # Try registering with the discovered path
            MODEL_VERSION=$(az ml model create \
              --name $MODEL_NAME \
              --path "azureml://jobs/$JOB_ID/outputs/$RELATIVE_PATH" \
              --type mlflow_model \
              --description "Customer churn prediction model trained via
GitHub Actions" \
              --query version -o tsv)
          else
            echo "No MLmodel file found. Available files:"
            find ./debug_download -type f | head -20
            exit 1
          fi
        fi
        if [ ! -z "$MODEL_VERSION" ] && [ "$MODEL_VERSION" !=
"mlflow_failed" ] && [ "$MODEL_VERSION" != "job_output_failed" ]; then
          echo "model_version=$MODEL_VERSION" >> $GITHUB_OUTPUT
          echo "Model successfully registered with version: $MODEL_VERSION"
        else
          echo "Failed to register model after all attempts"
          exit 1
        fi
    - name: Create/Update Online Endpoint
      run: |
        # Check if endpoint exists
        if az ml online-endpoint show --name $ENDPOINT_NAME &>/dev/null;
then
          echo "Endpoint $ENDPOINT_NAME already exists"
        else
          echo "Creating new endpoint $ENDPOINT_NAME"
          az ml online-endpoint create --file ml/endpoint.yml --name
$ENDPOINT_NAME
        fi

    - name: Deploy model to endpoint
      run: |
        # Create deployment configuration
        cat > deployment.yml << EOF
        \$schema:
https://azuremlschemas.azureedge.net/latest/managedOnlineDeployment.schema.j
son
        name: blue
        endpoint_name: $ENDPOINT_NAME
        model: azureml:$MODEL_NAME:${{ steps.register.outputs.model_version
}}
        instance_type: Standard_DS3_v2
        instance_count: 1
        environment_variables:
          MLFLOW_MODEL_DIRECTORY: /var/azureml-app/azureml-models/model/1
        request_settings:
          request_timeout_ms: 90000
```

```
          max_concurrent_requests_per_instance: 1
        liveness_probe:
          initial_delay: 10
          period: 10
          timeout: 2
          success_threshold: 1
          failure_threshold: 30
        readiness_probe:
          initial_delay: 10
          period: 10
          timeout: 2
          success_threshold: 1
          failure_threshold: 3
        EOF
        az ml online-deployment create --file deployment.yml --all-traffic

    - name: Test endpoint
      run: |
        az ml online-endpoint invoke --name $ENDPOINT_NAME --request-file
ml/sample-request.json

    - name: Send custom metrics to Azure Monitor
      run: |
        # Send custom metrics about the deployment
        az monitor metrics send \
          --resource
"/subscriptions/$AZURE_SUBSCRIPTION_ID/resourceGroups/$AZURE_RESOURCE_GROUP/
providers/Microsoft.MachineLearningServices/workspaces/$AZURE_ML_WORKSPACE"
\
          --metric-name "ModelDeploymentSuccess" \
          --metric-value 1 \
          --metric-timestamp $(date -u +"%Y-%m-%dT%H:%M:%SZ") || true

    - name: Create Application Insights alert
      run: |
        # Create alert rule for endpoint failures
        az monitor metrics alert create \
          --name "MLEndpointFailureAlert" \
          --resource-group $AZURE_RESOURCE_GROUP \
          --condition "avg requests/failed > 5" \
          --window-size 5m \
          --evaluation-frequency 1m \
          --severity 2 \
          --description "Alert when ML endpoint has more than 5 failed
requests" \
          --scopes
"/subscriptions/$AZURE_SUBSCRIPTION_ID/resourceGroups/$AZURE_RESOURCE_GROUP/
providers/Microsoft.MachineLearningServices/workspaces/$AZURE_ML_WORKSPACE/o
nlineEndpoints/$ENDPOINT_NAME" || true

  notify-on-failure:
```

```
    runs-on: ubuntu-latest
    needs: [unit-tests, build-and-train]
    if: failure()
    steps:
    - name: Notify on pipeline failure
      uses: 8398a7/action-slack@v3
      with:
        status: failure
        channel: '#mlops-alerts'
        webhook_url: ${{ secrets.SLACK_WEBHOOK }}
      env:
        SLACK_WEBHOOK_URL: ${{ secrets.SLACK_WEBHOOK }}

    - name: Create GitHub issue on failure
      if: github.event_name == 'push'
      uses: actions/github-script@v6
      with:
        script: |
          github.rest.issues.create({
            owner: context.repo.owner,
            repo: context.repo.repo,
            title: `MLOps Pipeline Failed - ${context.sha.substring(0, 7)}`,
            body: `The MLOps pipeline failed for commit
${context.sha}.\n\nWorkflow: ${context.workflow}\nRun:
${context.runNumber}\n\nPlease investigate the failure.`,
            labels: ['mlops', 'pipeline-failure', 'bug']
          })

  security-scan:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Run Trivy vulnerability scanner
      uses: aquasecurity/trivy-action@master
      with:
        scan-type: 'fs'
        scan-ref: '.'
        format: 'sarif'
        output: 'trivy-results.sarif'

    - name: Upload Trivy scan results
      uses: github/codeql-action/upload-sarif@v2
      with:
        sarif_file: 'trivy-results.sarif'
```

endpoint.yml

```
$schema:
```

```
https://azuremlschemas.azureedge.net/latest/managedOnlineEndpoint.schema.jso
n
name: churn-prediction-endpoint
description: Endpoint for customer churn prediction
auth_mode: key

tags:
  model: customer-churn-model
  environment: production
```

environment.yml

```
# ml/environment.yml
name: ml-environment
dependencies:
  - python=3.9
  - pip
  - pip:
    - scikit-learn==1.3.0
    - pandas==2.0.3
    - numpy==1.24.3
    - mlflow==2.5.0
    - azure-ai-ml==1.8.0
    - joblib==1.3.1
```

sample-request.json

```
{
  "input_data": {
    "columns": [
      "Account Length", "Area Code", "VMail Message", "Day Mins", "Day
Calls",
      "Day Charge", "Eve Mins", "Eve Calls", "Eve Charge", "Night Mins",
      "Night Calls", "Night Charge", "Intl Mins", "Intl Calls", "Intl
Charge",
      "CustServ Calls", "State", "Int'l Plan", "VMail Plan",
      "Avg_Day_Call_Duration", "Avg_Eve_Call_Duration",
"Avg_Night_Call_Duration",
      "Total_Charges", "Total_Usage_Mins"
    ],
    "index": [0],
    "data": [
      [142, 408, 28, 180.5, 95, 30.69, 210.2, 88, 17.87, 201.9, 82, 9.09,
       8.5, 4, 2.30, 2, 5, 0, 1, 1.9, 2.4, 2.5, 59.95, 601.1]
    ]
  }
}
```

====== training-job.yml

```
# ml/training-job.yml
$schema: https://azuremlschemas.azureedge.net/latest/commandJob.schema.json
type: command

display_name: Customer Churn Model Training
description: Training job for customer churn prediction model

experiment_name: customer-churn-experiment

compute: azureml:cpu-cluster

environment: azureml:AzureML-sklearn-1.0-ubuntu20.04-py38-cpu@latest

code: ../src

command: >
  python train.py
  --data_path ${{inputs.training_data}}
  --model_output ${{outputs.model_output}}
  --test_size 0.2
  --random_state 42

inputs:
  training_data:
    type: uri_file
    path:
https://raw.githubusercontent.com/albayraktaroglu/Datasets/master/churn.csv

outputs:
  model_output:
    type: uri_folder
    mode: rw_mount

environment_variables:
  MLFLOW_TRACKING_URI: "azureml://experiments/customer-churn-experiment"

tags:
  model_type: "classification"
  framework: "scikit-learn"
  training_method: "automated"
```

===== ====== train.py ====== =====

```
# src/train.py
import argparse
import os
```

```
import pandas as pd
import numpy as np
import mlflow
import mlflow.sklearn
import joblib
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, roc_auc_score
from sklearn.pipeline import Pipeline
import logging

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def parse_args():
    parser = argparse.ArgumentParser(description="Train customer churn
model")
    parser.add_argument("--data_path", type=str, required=True, help="Path
to training data")
    parser.add_argument("--model_output", type=str, required=True,
help="Path to save model")
    parser.add_argument("--test_size", type=float, default=0.2, help="Test
set size")
    parser.add_argument("--random_state", type=int, default=42, help="Random
state")
    parser.add_argument("--n_estimators", type=int, default=100,
help="Number of trees")
    parser.add_argument("--max_depth", type=int, default=10, help="Maximum
depth")
    return parser.parse_args()

def load_and_preprocess_data(data_path):
    """Load and preprocess the customer churn data"""
    logger.info(f"Loading data from {data_path}")
    # For demo purposes, creating sample data if file doesn't exist
    if not os.path.exists(data_path):
        logger.info("Creating sample data for demonstration")
        np.random.seed(42)
        n_samples = 10000
        data = {
            'CreditScore': np.random.randint(350, 850, n_samples),
            'Geography': np.random.choice(['France', 'Spain', 'Germany'],
n_samples),
            'Gender': np.random.choice(['Male', 'Female'], n_samples),
            'Age': np.random.randint(18, 92, n_samples),
            'Tenure': np.random.randint(0, 10, n_samples),
            'Balance': np.random.uniform(0, 250000, n_samples),
            'NumOfProducts': np.random.randint(1, 4, n_samples),
```

```
            'HasCrCard': np.random.choice([0, 1], n_samples),
            'IsActiveMember': np.random.choice([0, 1], n_samples),
            'EstimatedSalary': np.random.uniform(0, 200000, n_samples),
        }
        # Create target with some logic
        df = pd.DataFrame(data)
        churn_prob = (
            0.1 +
            0.3 * (df['Age'] > 50).astype(int) +
            0.2 * (df['NumOfProducts'] == 1).astype(int) +
            0.15 * (df['IsActiveMember'] == 0).astype(int) +
            0.1 * (df['Balance'] == 0).astype(int)
        )
        df['Exited'] = np.random.binomial(1, churn_prob)
    else:
        df = pd.read_csv(data_path)
    logger.info(f"Data shape: {df.shape}")
    logger.info(f"Column names: {list(df.columns)}")
    # Handle different target column names
    if 'Churn?' in df.columns:
        # Convert Churn? to binary Exited column
        df['Exited'] = (df['Churn?'] == 'True.').astype(int)
        logger.info(f"Churn rate: {df['Exited'].mean():.3f}")
        # Drop the original column
        df = df.drop('Churn?', axis=1)
    elif 'Exited' in df.columns:
        logger.info(f"Churn rate: {df['Exited'].mean():.3f}")
    else:
        logger.error("No target column found. Expected 'Exited' or
'Churn?'")
        raise ValueError("Target column not found")
    return df

def preprocess_features(df):
    """Preprocess features for training"""
    df_processed = df.copy()
    label_encoders = {}
    # Handle different dataset structures
    if 'Geography' in df.columns and 'Gender' in df.columns:
        # Bank churn dataset structure
        le_geography = LabelEncoder()
        le_gender = LabelEncoder()
        df_processed['Geography'] =
le_geography.fit_transform(df['Geography'])
        df_processed['Gender'] = le_gender.fit_transform(df['Gender'])
        label_encoders['Geography'] = le_geography
        label_encoders['Gender'] = le_gender
        # Feature engineering for bank dataset
        if 'CreditScore' in df.columns and 'Age' in df.columns:
            df_processed['CreditScore_Age_Ratio'] =
df_processed['CreditScore'] / df_processed['Age']
```

```
        if 'Balance' in df.columns and 'EstimatedSalary' in df.columns:
            df_processed['Balance_Salary_Ratio'] = df_processed['Balance'] /
(df_processed['EstimatedSalary'] + 1)
            df_processed['Balance_Log'] = np.log1p(df_processed['Balance'])
            df_processed['EstimatedSalary_Log'] =
np.log1p(df_processed['EstimatedSalary'])
    else:
        # Telecom churn dataset structure
        logger.info("Processing telecom churn dataset")
        # Handle categorical columns
        categorical_columns = []
        for col in df_processed.columns:
            if df_processed[col].dtype == 'object' and col != 'Exited':
                categorical_columns.append(col)
        logger.info(f"Categorical columns found: {categorical_columns}")
        # Encode categorical variables
        for col in categorical_columns:
            if col not in ['Phone']:  # Skip phone numbers
                le = LabelEncoder()
                df_processed[col] =
le.fit_transform(df_processed[col].astype(str))
                label_encoders[col] = le
        # Drop non-predictive columns
        columns_to_drop = ['Phone']
        df_processed = df_processed.drop(columns=[col for col in
columns_to_drop if col in df_processed.columns])
        # Feature engineering for telecom dataset
        if 'Day Mins' in df_processed.columns and 'Day Calls' in
df_processed.columns:
            df_processed['Avg_Day_Call_Duration'] = df_processed['Day Mins']
/ (df_processed['Day Calls'] + 1)
        if 'Eve Mins' in df_processed.columns and 'Eve Calls' in
df_processed.columns:
            df_processed['Avg_Eve_Call_Duration'] = df_processed['Eve Mins']
/ (df_processed['Eve Calls'] + 1)
        if 'Night Mins' in df_processed.columns and 'Night Calls' in
df_processed.columns:
            df_processed['Avg_Night_Call_Duration'] = df_processed['Night
Mins'] / (df_processed['Night Calls'] + 1)
        # Total usage features
        usage_cols = [col for col in df_processed.columns if 'Mins' in col]
        if usage_cols:
            df_processed['Total_Usage_Mins'] =
df_processed[usage_cols].sum(axis=1)
        charge_cols = [col for col in df_processed.columns if 'Charge' in
col]
        if charge_cols:
            df_processed['Total_Charges'] =
df_processed[charge_cols].sum(axis=1)
    # Handle boolean columns
    for col in df_processed.columns:
```

```
        if df_processed[col].dtype == 'bool':
            df_processed[col] = df_processed[col].astype(int)
    return df_processed, label_encoders

def train_model(X_train, y_train, n_estimators, max_depth, random_state):
    """Train the random forest model"""
    logger.info("Training Random Forest model")
    # Create pipeline with scaling
    pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('classifier', RandomForestClassifier(
            n_estimators=n_estimators,
            max_depth=max_depth,
            random_state=random_state,
            n_jobs=-1
        ))
    ])
    pipeline.fit(X_train, y_train)
    return pipeline

def evaluate_model(model, X_test, y_test):
    """Evaluate model performance"""
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1]
    metrics = {
        'accuracy': accuracy_score(y_test, y_pred),
        'precision': precision_score(y_test, y_pred),
        'recall': recall_score(y_test, y_pred),
        'f1_score': f1_score(y_test, y_pred),
        'roc_auc': roc_auc_score(y_test, y_pred_proba)
    }
    logger.info("Model Performance:")
    for metric, value in metrics.items():
        logger.info(f"{metric}: {value:.4f}")
    return metrics

def main():
    args = parse_args()
    # Start MLflow run
    with mlflow.start_run():
        # Log parameters
        mlflow.log_param("test_size", args.test_size)
        mlflow.log_param("random_state", args.random_state)
        mlflow.log_param("n_estimators", args.n_estimators)
        mlflow.log_param("max_depth", args.max_depth)
        # Load and preprocess data
        df = load_and_preprocess_data(args.data_path)
        df_processed, label_encoders = preprocess_features(df)
        # Prepare features and target
        feature_columns = [col for col in df_processed.columns if col !=
'Exited']
```

```
        X = df_processed[feature_columns]
        y = df_processed['Exited']
        logger.info(f"Feature columns: {feature_columns}")
        logger.info(f"Features shape: {X.shape}")
        logger.info(f"Target shape: {y.shape}")
        # Split data
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=args.test_size, random_state=args.random_state,
stratify=y
        )
        logger.info(f"Training set size: {X_train.shape[0]}")
        logger.info(f"Test set size: {X_test.shape[0]}")
        # Train model
        model = train_model(X_train, y_train, args.n_estimators,
args.max_depth, args.random_state)
        # Evaluate model
        metrics = evaluate_model(model, X_test, y_test)
        # Log metrics
        for metric, value in metrics.items():
            mlflow.log_metric(metric, value)
        # Log feature importance
        feature_importance =
model.named_steps['classifier'].feature_importances_
        for i, (feature, importance) in enumerate(zip(feature_columns,
feature_importance)):
            # Sanitize feature name for MLflow (remove invalid characters)
            sanitized_feature = feature.replace("'", "").replace(" ",
"_").replace("-", "_")
            mlflow.log_metric(f"feature_importance_{sanitized_feature}",
importance)
        # Log additional info
        mlflow.log_metric("training_samples", len(X_train))
        mlflow.log_metric("test_samples", len(X_test))
        mlflow.log_metric("n_features", X_train.shape[1])
        # Save model artifacts
        os.makedirs(args.model_output, exist_ok=True)
        # Save the model using MLflow
        model_path = os.path.join(args.model_output, "model")
        mlflow.sklearn.save_model(model, model_path)
        # Log the model path for debugging
        logger.info(f"Model saved to: {model_path}")
        logger.info(f"Contents of model output directory:
{os.listdir(args.model_output)}")
        # Save preprocessing objects
        joblib.dump(label_encoders, os.path.join(args.model_output,
"label_encoders.pkl"))
        joblib.dump(feature_columns, os.path.join(args.model_output,
"feature_columns.pkl"))
        # Log model with MLflow
        mlflow.sklearn.log_model(
            model,
```

```
            "model",
            registered_model_name="customer-churn-model"
        )
        logger.info(f"Model saved to {args.model_output}")
        logger.info("Training completed successfully!")

if __name__ == "__main__":
    main()
```

test_integration.py

```
# tests/test_integration.py
import pytest
import tempfile
import os
import pandas as pd
import numpy as np
from src.train import main
import sys
from unittest.mock import patch

class TestIntegration:
    def test_end_to_end_training(self):
        """Test the complete training pipeline"""
        with tempfile.TemporaryDirectory() as temp_dir:
            # Create test arguments
            test_args = [
                'train.py',
                '--data_path', 'non_existent_path.csv',  # Will trigger
sample data creation
                '--model_output', temp_dir,
                '--test_size', '0.2',
                '--random_state', '42',
                '--n_estimators', '10',
                '--max_depth', '5'
            ]
            # Mock sys.argv
            with patch.object(sys, 'argv', test_args):
                try:
                    main()
                    # Check that model files are created
                    assert os.path.exists(os.path.join(temp_dir, 'model'))
                    assert os.path.exists(os.path.join(temp_dir,
'label_encoders.pkl'))
                    assert os.path.exists(os.path.join(temp_dir,
'feature_columns.pkl'))
                except SystemExit as e:
```

```
                    # MLflow might cause a system exit, which is okay for
testing
                    if e.code != 0:
                        raise
    def test_end_to_end_training_with_telecom_data(self):
        """Test the complete training pipeline with telecom-style data"""
        with tempfile.TemporaryDirectory() as temp_dir:
            # Create a temporary CSV file with telecom data structure
            test_data_path = os.path.join(temp_dir, 'test_churn.csv')
            # Create sample telecom data
            np.random.seed(42)
            n_samples = 100
            telecom_data = {
                'State': np.random.choice(['TX', 'CA', 'NY', 'FL'],
n_samples),
                'Account Length': np.random.randint(1, 243, n_samples),
                'Area Code': np.random.choice([408, 415, 510], n_samples),
                'Phone': [f"{np.random.randint(100,999)}-
{np.random.randint(1000,9999)}" for _ in range(n_samples)],
                "Int'l Plan": np.random.choice(['yes', 'no'], n_samples),
                'VMail Plan': np.random.choice(['yes', 'no'], n_samples),
                'VMail Message': np.random.randint(0, 51, n_samples),
                'Day Mins': np.random.uniform(0, 351, n_samples),
                'Day Calls': np.random.randint(0, 166, n_samples),
                'Day Charge': np.random.uniform(0, 60, n_samples),
                'Eve Mins': np.random.uniform(0, 364, n_samples),
                'Eve Calls': np.random.randint(0, 171, n_samples),
                'Eve Charge': np.random.uniform(0, 31, n_samples),
                'Night Mins': np.random.uniform(0, 396, n_samples),
                'Night Calls': np.random.randint(0, 176, n_samples),
                'Night Charge': np.random.uniform(0, 18, n_samples),
                'Intl Mins': np.random.uniform(0, 20, n_samples),
                'Intl Calls': np.random.randint(0, 21, n_samples),
                'Intl Charge': np.random.uniform(0, 6, n_samples),
                'CustServ Calls': np.random.randint(0, 10, n_samples),
                'Churn?': np.random.choice(['True.', 'False.'], n_samples)
            }
            df = pd.DataFrame(telecom_data)
            df.to_csv(test_data_path, index=False)
            # Create test arguments
            test_args = [
                'train.py',
                '--data_path', test_data_path,
                '--model_output', temp_dir,
                '--test_size', '0.2',
                '--random_state', '42',
                '--n_estimators', '10',
                '--max_depth', '5'
            ]
            # Mock sys.argv
            with patch.object(sys, 'argv', test_args):
```

```
                try:
                    main()
                    # Check that model files are created
                    assert os.path.exists(os.path.join(temp_dir, 'model'))
                    assert os.path.exists(os.path.join(temp_dir,
'label_encoders.pkl'))
                    assert os.path.exists(os.path.join(temp_dir,
'feature_columns.pkl'))
                    # Verify the saved files contain expected data
                    import joblib
                    label_encoders = joblib.load(os.path.join(temp_dir,
'label_encoders.pkl'))
                    feature_columns = joblib.load(os.path.join(temp_dir,
'feature_columns.pkl'))
                    # Should have label encoders for categorical columns
                    assert isinstance(label_encoders, dict)
                    assert len(label_encoders) > 0
                    # Should have feature columns
                    assert isinstance(feature_columns, list)
                    assert len(feature_columns) > 0
                    assert 'Exited' not in feature_columns  # Target should
not be in features
                except SystemExit as e:
                    # MLflow might cause a system exit, which is okay for
testing
                    if e.code != 0:
                        raise
    def test_end_to_end_training_with_bank_data(self):
        """Test the complete training pipeline with bank-style data"""
        with tempfile.TemporaryDirectory() as temp_dir:
            # Create a temporary CSV file with bank data structure
            test_data_path = os.path.join(temp_dir, 'test_bank_churn.csv')
            # Create sample bank data
            np.random.seed(42)
            n_samples = 100
            bank_data = {
                'CreditScore': np.random.randint(350, 850, n_samples),
                'Geography': np.random.choice(['France', 'Spain',
'Germany'], n_samples),
                'Gender': np.random.choice(['Male', 'Female'], n_samples),
                'Age': np.random.randint(18, 92, n_samples),
                'Tenure': np.random.randint(0, 10, n_samples),
                'Balance': np.random.uniform(0, 250000, n_samples),
                'NumOfProducts': np.random.randint(1, 4, n_samples),
                'HasCrCard': np.random.choice([0, 1], n_samples),
                'IsActiveMember': np.random.choice([0, 1], n_samples),
                'EstimatedSalary': np.random.uniform(0, 200000, n_samples),
                'Exited': np.random.choice([0, 1], n_samples)
            }
            df = pd.DataFrame(bank_data)
            df.to_csv(test_data_path, index=False)
```

```
            # Create test arguments
            test_args = [
                'train.py',
                '--data_path', test_data_path,
                '--model_output', temp_dir,
                '--test_size', '0.2',
                '--random_state', '42',
                '--n_estimators', '10',
                '--max_depth', '5'
            ]
            # Mock sys.argv
            with patch.object(sys, 'argv', test_args):
                try:
                    main()
                    # Check that model files are created
                    assert os.path.exists(os.path.join(temp_dir, 'model'))
                    assert os.path.exists(os.path.join(temp_dir,
'label_encoders.pkl'))
                    assert os.path.exists(os.path.join(temp_dir,
'feature_columns.pkl'))
                    # Verify the saved files contain expected data
                    import joblib
                    label_encoders = joblib.load(os.path.join(temp_dir,
'label_encoders.pkl'))
                    feature_columns = joblib.load(os.path.join(temp_dir,
'feature_columns.pkl'))
                    # Should have label encoders for Geography and Gender
                    assert isinstance(label_encoders, dict)
                    assert 'Geography' in label_encoders
                    assert 'Gender' in label_encoders
                    # Should have feature columns including engineered
features
                    assert isinstance(feature_columns, list)
                    assert len(feature_columns) > 10  # Should have original
+ engineered features
                    assert 'Exited' not in feature_columns  # Target should
not be in features
                    # Check for engineered features
                    assert 'CreditScore_Age_Ratio' in feature_columns
                    assert 'Balance_Salary_Ratio' in feature_columns
                except SystemExit as e:
                    # MLflow might cause a system exit, which is okay for
testing
                    if e.code != 0:
                        raise
```

test_train.py

```
# tests/test_train.py
import pytest
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from src.train import load_and_preprocess_data, preprocess_features,
train_model, evaluate_model

class TestTrainingPipeline:
    @pytest.fixture
    def sample_bank_data(self):
        """Create sample bank churn data for testing"""
        np.random.seed(42)
        n_samples = 100
        data = {
            'CreditScore': np.random.randint(350, 850, n_samples),
            'Geography': np.random.choice(['France', 'Spain', 'Germany'],
n_samples),
            'Gender': np.random.choice(['Male', 'Female'], n_samples),
            'Age': np.random.randint(18, 92, n_samples),
            'Tenure': np.random.randint(0, 10, n_samples),
            'Balance': np.random.uniform(0, 250000, n_samples),
            'NumOfProducts': np.random.randint(1, 4, n_samples),
            'HasCrCard': np.random.choice([0, 1], n_samples),
            'IsActiveMember': np.random.choice([0, 1], n_samples),
            'EstimatedSalary': np.random.uniform(0, 200000, n_samples),
            'Exited': np.random.choice([0, 1], n_samples)
        }
        return pd.DataFrame(data)
    @pytest.fixture
    def sample_telecom_data(self):
        """Create sample telecom churn data for testing"""
        np.random.seed(42)
        n_samples = 100
        data = {
            'State': np.random.choice(['TX', 'CA', 'NY', 'FL'], n_samples),
            'Account Length': np.random.randint(1, 243, n_samples),
            'Area Code': np.random.choice([408, 415, 510], n_samples),
            'Phone': [f"{np.random.randint(100,999)}-
{np.random.randint(1000,9999)}" for _ in range(n_samples)],
            "Int'l Plan": np.random.choice(['yes', 'no'], n_samples),
            'VMail Plan': np.random.choice(['yes', 'no'], n_samples),
            'VMail Message': np.random.randint(0, 51, n_samples),
            'Day Mins': np.random.uniform(0, 351, n_samples),
            'Day Calls': np.random.randint(0, 166, n_samples),
            'Day Charge': np.random.uniform(0, 60, n_samples),
```

```
            'Eve Mins': np.random.uniform(0, 364, n_samples),
            'Eve Calls': np.random.randint(0, 171, n_samples),
            'Eve Charge': np.random.uniform(0, 31, n_samples),
            'Night Mins': np.random.uniform(0, 396, n_samples),
            'Night Calls': np.random.randint(0, 176, n_samples),
            'Night Charge': np.random.uniform(0, 18, n_samples),
            'Intl Mins': np.random.uniform(0, 20, n_samples),
            'Intl Calls': np.random.randint(0, 21, n_samples),
            'Intl Charge': np.random.uniform(0, 6, n_samples),
            'CustServ Calls': np.random.randint(0, 10, n_samples),
            'Churn?': np.random.choice(['True.', 'False.'], n_samples)
        }
        return pd.DataFrame(data)
    def test_preprocess_bank_features(self, sample_bank_data):
        """Test feature preprocessing for bank data"""
        df_processed, label_encoders = preprocess_features(sample_bank_data)
        # Check that categorical variables are encoded
        assert df_processed['Geography'].dtype in ['int64', 'int32']
        assert df_processed['Gender'].dtype in ['int64', 'int32']
        # Check that new features are created
        assert 'CreditScore_Age_Ratio' in df_processed.columns
        assert 'Balance_Salary_Ratio' in df_processed.columns
        assert 'Balance_Log' in df_processed.columns
        assert 'EstimatedSalary_Log' in df_processed.columns
        # Check that encoders work
        assert 'Geography' in label_encoders
        assert 'Gender' in label_encoders
        assert len(label_encoders['Geography'].classes_) <= 3
        assert len(label_encoders['Gender'].classes_) <= 2
    def test_preprocess_telecom_features(self, sample_telecom_data):
        """Test feature preprocessing for telecom data"""
        # First convert Churn? to Exited
        sample_telecom_data['Exited'] = (sample_telecom_data['Churn?'] ==
'True.').astype(int)
        sample_telecom_data = sample_telecom_data.drop('Churn?', axis=1)
        df_processed, label_encoders =
preprocess_features(sample_telecom_data)
        # Check that categorical variables are encoded
        categorical_cols = ['State', "Int'l Plan", 'VMail Plan']
        for col in categorical_cols:
            if col in df_processed.columns:
                assert df_processed[col].dtype in ['int64', 'int32']
                assert col in label_encoders
        # Check that phone numbers are dropped
        assert 'Phone' not in df_processed.columns
        # Check that new telecom features are created
        assert 'Avg_Day_Call_Duration' in df_processed.columns
        assert 'Avg_Eve_Call_Duration' in df_processed.columns
        assert 'Avg_Night_Call_Duration' in df_processed.columns
        assert 'Total_Usage_Mins' in df_processed.columns
        assert 'Total_Charges' in df_processed.columns
```

```
    def test_train_model_bank_data(self, sample_bank_data):
        """Test model training with bank data"""
        df_processed, _ = preprocess_features(sample_bank_data)
        feature_columns = [col for col in df_processed.columns if col !=
'Exited']
        X = df_processed[feature_columns]
        y = df_processed['Exited']
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42
        )
        model = train_model(X_train, y_train, n_estimators=10, max_depth=5,
random_state=42)
        # Check that model is trained
        assert hasattr(model, 'predict')
        assert hasattr(model, 'predict_proba')
        # Check predictions
        predictions = model.predict(X_test)
        assert len(predictions) == len(X_test)
        assert all(pred in [0, 1] for pred in predictions)
    def test_train_model_telecom_data(self, sample_telecom_data):
        """Test model training with telecom data"""
        # First convert Churn? to Exited
        sample_telecom_data['Exited'] = (sample_telecom_data['Churn?'] ==
'True.').astype(int)
        sample_telecom_data = sample_telecom_data.drop('Churn?', axis=1)
        df_processed, _ = preprocess_features(sample_telecom_data)
        feature_columns = [col for col in df_processed.columns if col !=
'Exited']
        X = df_processed[feature_columns]
        y = df_processed['Exited']
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42
        )
        model = train_model(X_train, y_train, n_estimators=10, max_depth=5,
random_state=42)
        # Check that model is trained
        assert hasattr(model, 'predict')
        assert hasattr(model, 'predict_proba')
        # Check predictions
        predictions = model.predict(X_test)
        assert len(predictions) == len(X_test)
        assert all(pred in [0, 1] for pred in predictions)
    def test_evaluate_model(self, sample_bank_data):
        """Test model evaluation"""
        df_processed, _ = preprocess_features(sample_bank_data)
        feature_columns = [col for col in df_processed.columns if col !=
'Exited']
        X = df_processed[feature_columns]
        y = df_processed['Exited']
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42
```

```
        )
        model = train_model(X_train, y_train, n_estimators=10, max_depth=5,
random_state=42)
        metrics = evaluate_model(model, X_test, y_test)
        # Check that all metrics are present
        expected_metrics = ['accuracy', 'precision', 'recall', 'f1_score',
'roc_auc']
        for metric in expected_metrics:
            assert metric in metrics
            assert 0 <= metrics[metric] <= 1
    def test_data_shapes_bank(self, sample_bank_data):
        """Test data shapes after preprocessing bank data"""
        df_processed, _ = preprocess_features(sample_bank_data)
        # Should have more columns after feature engineering
        assert df_processed.shape[1] > sample_bank_data.shape[1]
        # Should have same number of rows
        assert df_processed.shape[0] == sample_bank_data.shape[0]
    def test_data_shapes_telecom(self, sample_telecom_data):
        """Test data shapes after preprocessing telecom data"""
        # First convert Churn? to Exited
        original_cols = sample_telecom_data.shape[1]
        sample_telecom_data['Exited'] = (sample_telecom_data['Churn?'] ==
'True.').astype(int)
        sample_telecom_data = sample_telecom_data.drop('Churn?', axis=1)
        df_processed, _ = preprocess_features(sample_telecom_data)
        # Should have more columns after feature engineering (considering
Phone is dropped)
        assert df_processed.shape[1] > original_cols - 2  # -1 for Phone
drop, -1 for Churn? -> Exited
        # Should have same number of rows
        assert df_processed.shape[0] == sample_telecom_data.shape[0]
    def test_load_and_preprocess_data_nonexistent_file(self):
        """Test load_and_preprocess_data with non-existent file (should
create sample data)"""
        df = load_and_preprocess_data('nonexistent_file.csv')
        # Should create sample data
        assert isinstance(df, pd.DataFrame)
        assert 'Exited' in df.columns
        assert df.shape[0] > 0
        # Should have bank data structure when creating sample data
        expected_bank_cols = ['CreditScore', 'Geography', 'Gender', 'Age',
'Tenure',
                              'Balance', 'NumOfProducts', 'HasCrCard',
'IsActiveMember',
                              'EstimatedSalary', 'Exited']
        for col in expected_bank_cols:
            assert col in df.columns
```

requirements.txt

```
# requirements.txt
scikit-learn==1.3.0
pandas==2.0.3
numpy==1.24.3
mlflow==2.5.0
azure-ai-ml==1.8.0
azure-identity==1.13.0
azure-storage-blob==12.17.0
joblib==1.3.1
pytest==7.4.0
pytest-cov==4.1.0
flake8==6.0.0
```