Week 14 Lab: Model Monitoring

CS 203: Software Tools and Techniques for AI

Duration: 3 hours

Lab Overview

Objective: Build a complete monitoring system to detect model degradation.

Structure:

- **Part 1**: Baseline model training (30 min)
- **Part 2**: Simulate production environment with drift (30 min)
- **Part 3**: Drift detection with Evidently AI (45 min)
- **Part 4**: Create monitoring dashboard (40 min)
- **Part 5**: Automated alerting system (30 min)
- **Part 6**: Performance degradation analysis (35 min)

Dataset: Bike Sharing dataset (weather features → bike rentals)

- **Reference data**: 2011 (training)
- **Production data**: 2012 (gradual weather/seasonal drift)

Setup and Installation

Create a new notebook:

```
# Install required packages
!pip install evidently pandas scikit-learn matplotlib seaborn numpy plotly kaleido

# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime, timedelta
import warnings
warnings.filterwarnings('ignore')

# Scikit-learn
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
import joblib

# Evidently
from evidently.report import Report
from evidently.metric_preset import DataDriftPreset, RegressionPreset, TargetDriftPreset
from evidently.test_suite import TestSuite
from evidently.test_preset import DataDriftTestPreset, DataQualityTestPreset

print("✓ Packages imported successfully")
```

Part 1: Baseline Model Training (30 min)

Goal: Train a baseline model and establish reference metrics.

Exercise 1.1: Load and explore the bike sharing dataset (10 min)

```
# Download dataset
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/00275/Bike-Sharing-Dataset.zip
!unzip -o Bike-Sharing-Dataset.zip

# Load data
df_hour = pd.read_csv('hour.csv')
df_day = pd.read_csv('day.csv')

print(f"Hourly data shape: {df_hour.shape}")
print(f"Daily data shape: {df_day.shape}")

# Use daily data for simplicity
df = df_day.copy()

# Convert date
df['dteday'] = pd.to_datetime(df['dteday'])

# Display first few rows
print("\nFirst 5 rows:")
print(df.head())

# Data description
print("\nDataset summary:")
print(df.describe())
```

Exercise 1.1 (continued)

```
# Feature descriptions
feature_info = {
    'season': '1=spring, 2=summer, 3=fall, 4=winter',
    'yr': '0=2011, 1=2012',
    'mnth': 'Month (1-12)',
    'holiday': 'Whether day is holiday',
    'weekday': 'Day of week',
    'workingday': 'Neither weekend nor holiday',
    'weathersit': '1=Clear, 2=Mist, 3=Light Snow/Rain, 4=Heavy Rain',
    'temp': 'Normalized temperature (-8 to 39°C)',
    'atemp': 'Normalized feeling temperature',
    'hum': 'Normalized humidity',
    'windspeed': 'Normalized wind speed',
    'cnt': 'Total rental bikes (TARGET)'
}

print("\nFeature Information:")
for feat, desc in feature_info.items():
    if feat in df.columns:
        print(f"  {feat:15s}: {desc}")

# Quick EDA
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# 1. Target distribution
axes[0, 0].hist(df['cnt'], bins=50, edgecolor='black', alpha=0.7)
axes[0, 0].set_xlabel('Bike Rentals')
axes[0, 0].set_ylabel('Frequency')
axes[0, 0].set_title('Distribution of Bike Rentals')
axes[0, 0].grid(alpha=0.3)

# 2. Rentals over time
axes[0, 1].plot(df['dteday'], df['cnt'], alpha=0.7)
axes[0, 1].set_xlabel('Date')
axes[0, 1].set_ylabel('Bike Rentals')
axes[0, 1].set_title('Bike Rentals Over Time')
axes[0, 1].grid(alpha=0.3)

# 3. Correlation with features
corr_features = ['temp', 'atemp', 'hum', 'windspeed', 'cnt']
corr = df[corr_features].corr()['cnt'].drop('cnt').sort_values(ascending=False)
axes[1, 0].barh(corr.index, corr.values)
axes[1, 0].set_xlabel('Correlation with Rentals')
axes[1, 0].set_title('Feature Correlations')
axes[1, 0].grid(alpha=0.3)

# 4. Seasonal pattern
seasonal_avg = df.groupby('season')['cnt'].mean()
season_names = ['Spring', 'Summer', 'Fall', 'Winter']
axes[1, 1].bar(season_names, seasonal_avg.values, color=['green', 'orange', 'brown', 'blue'])
axes[1, 1].set_ylabel('Average Rentals')
axes[1, 1].set_title('Average Rentals by Season')
axes[1, 1].grid(alpha=0.3)

plt.tight_layout()
plt.savefig('eda_bike_sharing.png', dpi=150)
plt.show()

print("\n✓ Exploratory Data Analysis complete")
```

Exercise 1.2: Prepare training data (10 min)

```python
# Split by year: 2011 for training (reference), 2012 for production
reference_data = df[df['yr'] == 0].copy()  # 2011
production_data = df[df['yr'] == 1].copy()  # 2012

print(f"Reference data (2011): {len(reference_data)} days")
print(f"Production data (2012): {len(production_data)} days")

# Define features and target
feature_cols = [
    'season', 'mnth', 'holiday', 'weekday', 'workingday',
    'weathersit', 'temp', 'atemp', 'hum', 'windspeed'
]

target_col = 'cnt'

# Prepare training data
X_train = reference_data[feature_cols]
y_train = reference_data[target_col]

# Further split for validation
X_train_split, X_val, y_train_split, y_val = train_test_split(
    X_train, y_train, test_size=0.2, random_state=42
)

print(f"\nTraining set: {len(X_train_split)} samples")
print(f"Validation set: {len(X_val)} samples")

# Save reference data for later
reference_data.to_csv('reference_data.csv', index=False)
production_data.to_csv('production_data.csv', index=False)

print("\n✓ Data preparation complete")
```

Exercise 1.3: Train baseline model (10 min)

```
# Train Random Forest model
print("Training Random Forest model...")

model = RandomForestRegressor(
    n_estimators=100,
    max_depth=15,
    min_samples_split=5,
    min_samples_leaf=2,
    random_state=42,
    n_jobs=-1
)

model.fit(X_train_split, y_train_split)

# Evaluate on validation set
y_val_pred = model.predict(X_val)

val_mae = mean_absolute_error(y_val, y_val_pred)
val_rmse = np.sqrt(mean_squared_error(y_val, y_val_pred))
val_r2 = r2_score(y_val, y_val_pred)

print(f"\nValidation Metrics:")
print(f"  MAE:  {val_mae:.2f}")
print(f"  RMSE: {val_rmse:.2f}")
print(f"  R²:   {val_r2:.4f}")

# Feature importance
feature_importance = pd.DataFrame({
    'feature': feature_cols,
    'importance': model.feature_importances_
}).sort_values('importance', ascending=False)

print(f"\nTop 5 Important Features:")
print(feature_importance.head())

# Visualize feature importance
plt.figure(figsize=(10, 6))
plt.barh(feature_importance['feature'], feature_importance['importance'])
plt.xlabel('Importance')
plt.title('Feature Importance')
plt.grid(alpha=0.3)
plt.tight_layout()
plt.savefig('feature_importance.png', dpi=150)
plt.show()

# Save model
joblib.dump(model, 'bike_rental_model.pkl')
print("\n✓ Model saved to bike_rental_model.pkl")

# Store baseline metrics
baseline_metrics = {
    'mae': val_mae,
    'rmse': val_rmse,
    'r2': val_r2,
    'model': 'RandomForest',
    'trained_on': '2011 data',
    'n_samples': len(X_train_split)
}

import json
with open('baseline_metrics.json', 'w') as f:
    json.dump(baseline_metrics, f, indent=2)

print("\n✓ Baseline metrics saved")
```

Part 2: Simulate Production Environment (30 min)

Goal: Simulate a production deployment where data gradually drifts.

Exercise 2.1: Create production batches (15 min)

```
# Simulate batches of production data (weekly batches)
production_data['dteday'] = pd.to_datetime(production_data['dteday'])

# Sort by date
production_data = production_data.sort_values('dteday').reset_index(drop=True)

# Create weekly batches
production_data['week'] = ((production_data['dteday'] - production_data['dteday'].min()).dt.days // 7)

num_weeks = production_data['week'].max() + 1
print(f"Number of weeks in production data: {num_weeks}")

# Create batches
batches = []
for week in range(num_weeks):
    batch_data = production_data[production_data['week'] == week].copy()
    if len(batch_data) > 0:
        batches.append(batch_data)

print(f"Created {len(batches)} weekly batches")
print(f"Average batch size: {np.mean([len(b) for b in batches]):.1f} days")

# Save batches
import os
os.makedirs('production_batches', exist_ok=True)

for i, batch in enumerate(batches):
```

Exercise 2.2: Run model on production data (15 min)

```
# Simulate production predictions and track performance

# Load model
model = joblib.load('bike_rental_model.pkl')

# Track predictions and actual values
production_results = []

for week_idx, batch in enumerate(batches):
    X_batch = batch[feature_cols]
    y_batch_true = batch[target_col].values

    # Make predictions
    y_batch_pred = model.predict(X_batch)

    # Calculate metrics for this batch
    batch_mae = mean_absolute_error(y_batch_true, y_batch_pred)
    batch_rmse = np.sqrt(mean_squared_error(y_batch_true, y_batch_pred))
    batch_r2 = r2_score(y_batch_true, y_batch_pred)

    # Store results
    production_results.append({
        'week': week_idx,
        'start_date': batch['dteday'].min(),
        'end_date': batch['dteday'].max(),
        'n_samples': len(batch),
        'mae': batch_mae,
        'rmse': batch_rmse,
        'r2': batch_r2,
        'mean_prediction': y_batch_pred.mean(),
        'mean_actual': y_batch_true.mean()
    })

    # Log predictions
    batch['prediction'] = y_batch_pred
    batch.to_csv(f'production_batches/week_{week_idx:02d}_with_predictions.csv', index=False)

# Convert to DataFrame
results_df = pd.DataFrame(production_results)

print("Production Performance Summary:")
print(results_df.head(10))

# Save results
results_df.to_csv('production_performance.csv', index=False)
print("\n✓ Production results saved")
```

Exercise 2.2 (continued)

```
# Visualize performance degradation
fig, axes = plt.subplots(3, 1, figsize=(12, 12))

# 1. MAE over time
axes[0].plot(results_df['week'], results_df['mae'], marker='o', linewidth=2, markersize=6)
axes[0].axhline(y=baseline_metrics['mae'], color='red', linestyle='--', label='Baseline MAE')
axes[0].set_xlabel('Week')
axes[0].set_ylabel('MAE')
axes[0].set_title('Mean Absolute Error Over Time')
axes[0].legend()
axes[0].grid(alpha=0.3)

# 2. RMSE over time
axes[1].plot(results_df['week'], results_df['rmse'], marker='o', linewidth=2, markersize=6, color='orange')
axes[1].axhline(y=baseline_metrics['rmse'], color='red', linestyle='--', label='Baseline RMSE')
axes[1].set_xlabel('Week')
axes[1].set_ylabel('RMSE')
axes[1].set_title('Root Mean Squared Error Over Time')
axes[1].legend()
axes[1].grid(alpha=0.3)

# 3. R² over time
axes[2].plot(results_df['week'], results_df['r2'], marker='o', linewidth=2, markersize=6, color='green')
axes[2].axhline(y=baseline_metrics['r2'], color='red', linestyle='--', label='Baseline R²')
axes[2].set_xlabel('Week')
axes[2].set_ylabel('R²')
axes[2].set_title('R² Score Over Time')
axes[2].legend()
axes[2].grid(alpha=0.3)

plt.tight_layout()
plt.savefig('production_performance_over_time.png', dpi=150)
plt.show()

# Statistical summary
print("\nPerformance Statistics:")
print(f"  Mean MAE: {results_df['mae'].mean():.2f} (±{results_df['mae'].std():.2f})")
print(f"  Mean RMSE: {results_df['rmse'].mean():.2f} (±{results_df['rmse'].std():.2f})")
print(f"  Mean R²: {results_df['r2'].mean():.4f} (±{results_df['r2'].std():.4f})")

# Identify degradation
degradation_threshold = baseline_metrics['mae'] * 1.2  # 20% worse
degraded_weeks = results_df[results_df['mae'] > degradation_threshold]

if len(degraded_weeks) > 0:
    print(f"\n
⚠️
 Performance degradation detected in {len(degraded_weeks)} weeks:")
    print(degraded_weeks[['week', 'start_date', 'mae', 'rmse']])
else:
    print("\n✓ No significant performance degradation detected")
```

Part 3: Drift Detection with Evidently (45 min)

Goal: Use Evidently AI to detect data drift across production batches.

Exercise 3.1: Generate drift reports per batch (25 min)

```
# Load reference data
reference_data = pd.read_csv('reference_data.csv')

# Create drift reports for each week
drift_results = []

print("Generating drift reports for each week...")

for week_idx in range(len(batches)):
    # Load current batch
    current_batch = pd.read_csv(f'production_batches/week_{week_idx:02d}.csv')

    # Create drift report
    report = Report(metrics=[
        DataDriftPreset(),
        TargetDriftPreset()
    ])

    # Run report
    report.run(
        reference_data=reference_data[feature_cols + [target_col]],
        current_data=current_batch[feature_cols + [target_col]],
        column_mapping=None
    )

    # Save HTML report
    report.save_html(f'drift_reports/week_{week_idx:02d}_drift_report.html')

    # Extract drift metrics
    report_dict = report.as_dict()

    # Parse drift results (simplified extraction)
    dataset_drift = report_dict['metrics'][0]['result']['dataset_drift']

    drift_results.append({
        'week': week_idx,
        'dataset_drift': dataset_drift,
        'start_date': current_batch['dteday'].min()
    })
```

Exercise 3.2: Analyze drift patterns (20 min)

```
# Convert drift results to DataFrame
drift_df = pd.DataFrame(drift_results)

print("Drift Detection Summary:")
print(drift_df.head(10))

# Count weeks with drift
drifted_weeks = drift_df[drift_df['dataset_drift'] == True]
print(f"\nWeeks with drift detected: {len(drifted_weeks)} / {len(drift_df)}")

if len(drifted_weeks) > 0:
    print("\nFirst occurrence of drift:")
    print(drifted_weeks.iloc[0])

# Visualize drift over time
plt.figure(figsize=(12, 4))
plt.plot(drift_df['week'], drift_df['dataset_drift'].astype(int), marker='o', markersize=8)
plt.fill_between(drift_df['week'], 0, drift_df['dataset_drift'].astype(int), alpha=0.3, color='red')
plt.xlabel('Week')
plt.ylabel('Drift Detected (0=No, 1=Yes)')
plt.title('Data Drift Detection Over Time')
plt.ylim(-0.1, 1.1)
plt.grid(alpha=0.3)
plt.tight_layout()
plt.savefig('drift_over_time.png', dpi=150)
plt.show()

# Save drift results
drift_df.to_csv('drift_detection_results.csv', index=False)
print("\n✓ Drift results saved to drift_detection_results.csv")
```

Exercise 3.2 (continued): Feature-level drift analysis

```
# Analyze drift for individual features
from scipy.stats import ks_2samp

feature_drift_scores = []

print("\nAnalyzing feature-level drift...")

for week_idx in range(len(batches)):
    current_batch = pd.read_csv(f'production_batches/week_{week_idx:02d}.csv')

    week_drift = {'week': week_idx}

    for feature in feature_cols:
        ref_values = reference_data[feature].values
        curr_values = current_batch[feature].values

        # KS test
        statistic, p_value = ks_2samp(ref_values, curr_values)

        week_drift[f'{feature}_pval'] = p_value
        week_drift[f'{feature}_drift'] = p_value < 0.05  # Drift if p < 0.05

    feature_drift_scores.append(week_drift)

feature_drift_df = pd.DataFrame(feature_drift_scores)

print("Feature drift statistics:")
for feature in feature_cols:
    drift_count = feature_drift_df[f'{feature}_drift'].sum()
    drift_pct = (drift_count / len(feature_drift_df)) * 100
    print(f"  {feature:15s}: {drift_count:2d}/{len(feature_drift_df)} weeks ({drift_pct:5.1f}%)")

# Heatmap of p-values
pval_cols = [col for col in feature_drift_df.columns if col.endswith('_pval')]
pval_matrix = feature_drift_df[pval_cols].values.T

plt.figure(figsize=(14, 6))
sns.heatmap(
    pval_matrix,
    cmap='RdYlGn_r',
    vmin=0, vmax=0.1,
    yticklabels=[col.replace('_pval', '') for col in pval_cols],
    xticklabels=feature_drift_df['week'].values,
    cbar_kws={'label': 'P-value'}
)
plt.xlabel('Week')
plt.ylabel('Feature')
plt.title('Feature Drift P-values Over Time (Red = Drift)')
plt.tight_layout()
plt.savefig('feature_drift_heatmap.png', dpi=150)
plt.show()

# Save feature drift results
feature_drift_df.to_csv('feature_drift_scores.csv', index=False)
print("\n✓ Feature drift analysis complete")
```

Part 4: Create Monitoring Dashboard (40 min)

Goal: Build a comprehensive monitoring dashboard using Plotly.

Exercise 4.1: Build interactive dashboard (40 min)

```python
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Load all data
reference_data = pd.read_csv('reference_data.csv')
production_perf = pd.read_csv('production_performance.csv')
drift_results = pd.read_csv('drift_detection_results.csv')

# Create dashboard with multiple subplots
fig = make_subplots(
    rows=3, cols=2,
    subplot_titles=(
        'Model Performance (MAE)',
        'Model Performance (R²)',
        'Data Drift Detection',
        'Prediction vs Actual (Mean)',
        'Feature Drift Summary',
        'Alert Timeline'
    ),
    specs=[
        [{"type": "scatter"}, {"type": "scatter"}],
        [{"type": "scatter"}, {"type": "scatter"}],
        [{"type": "bar"}, {"type": "scatter"}]
    ],
    vertical_spacing=0.12,
    horizontal_spacing=0.12
)

# 1. MAE over time
fig.add_trace(
    go.Scatter(
        x=production_perf['week'],
        y=production_perf['mae'],
        mode='lines+markers',
        name='MAE',
        line=dict(color='blue', width=2),
        marker=dict(size=6)
    ),
    row=1, col=1
)
fig.add_hline(
    y=baseline_metrics['mae'],
    line_dash="dash",
    line_color="red",
    annotation_text="Baseline",
    row=1, col=1
)

# 2. R² over time
fig.add_trace(
    go.Scatter(
        x=production_perf['week'],
        y=production_perf['r2'],
        mode='lines+markers',
        name='R²',
        line=dict(color='green', width=2),
        marker=dict(size=6)
    ),
    row=1, col=2
)
fig.add_hline(
    y=baseline_metrics['r2'],
    line_dash="dash",
    line_color="red",
    annotation_text="Baseline",
    row=1, col=2
)

# 3. Drift detection
fig.add_trace(
    go.Scatter(
        x=drift_results['week'],
        y=drift_results['dataset_drift'].astype(int),
        mode='markers',
        name='Drift',
        marker=dict(
            size=12,
            color=drift_results['dataset_drift'].astype(int),
            colorscale=[[0, 'green'], [1, 'red']],
            showscale=False
        )
    ),
    row=2, col=1
)

# 4. Mean prediction vs actual
fig.add_trace(
    go.Scatter(
        x=production_perf['week'],
        y=production_perf['mean_actual'],
        mode='lines',
        name='Actual',
        line=dict(color='blue', width=2)
    ),
    row=2, col=2
)
fig.add_trace(
    go.Scatter(
        x=production_perf['week'],
        y=production_perf['mean_prediction'],
        mode='lines',
        name='Predicted',
        line=dict(color='orange', width=2, dash='dash')
    ),
    row=2, col=2
)

# 5. Feature drift summary (count of drifted features per week)
feature_drift_df = pd.read_csv('feature_drift_scores.csv')
drift_cols = [col for col in feature_drift_df.columns if col.endswith('_drift')]
drift_count_per_week = feature_drift_df[drift_cols].sum(axis=1)

fig.add_trace(
    go.Bar(
        x=feature_drift_df['week'],
        y=drift_count_per_week,
        name='# Drifted Features',
        marker=dict(color='orange')
    ),
    row=3, col=1
)

# 6. Alert timeline (combine performance and drift alerts)
alerts = []
alert_weeks = []
alert_types = []

# Performance alerts (MAE > baseline * 1.2)
perf_alerts = production_perf[production_perf['mae'] > baseline_metrics['mae'] * 1.2]
for _, row in perf_alerts.iterrows():
    alerts.append(1)
    alert_weeks.append(row['week'])
    alert_types.append('Performance')

# Drift alerts
drift_alerts = drift_results[drift_results['dataset_drift'] == True]
for _, row in drift_alerts.iterrows():
    alerts.append(0.5)
    alert_weeks.append(row['week'])
    alert_types.append('Drift')

fig.add_trace(
    go.Scatter(
        x=alert_weeks,
        y=alerts,
        mode='markers',
        name='Alerts',
        marker=dict(
            size=15,
            color=['red' if t == 'Performance' else 'orange' for t in alert_types],
            symbol=['circle' if t == 'Performance' else 'diamond' for t in alert_types]
        ),
        text=alert_types,
        hovertemplate='%{text}<br>Week: %{x}<extra></extra>'
    ),
    row=3, col=2
)

# Update layout
fig.update_xaxes(title_text="Week", row=3, col=1)
```

Part 5: Automated Alerting System (30 min)

Goal: Build an automated alerting system that triggers on drift/performance degradation.

Exercise 5.1: Create alert rules (15 min)

```
# Define alert rules
class MonitoringAlert:
    def __init__(self, name, severity, threshold, description):
        self.name = name
        self.severity = severity  # 'critical', 'warning', 'info'
        self.threshold = threshold
        self.description = description
        self.triggered = False
        self.trigger_week = None
        self.trigger_value = None

    def check(self, value, week):
        """Check if alert should trigger."""
        if self.severity in ['critical', 'warning']:
            if value > self.threshold:
                self.triggered = True
                self.trigger_week = week
                self.trigger_value = value
                return True
        return False

    def __repr__(self):
        status = "TRIGGERED" if self.triggered else "OK"
        return f"[{self.severity.upper()}] {self.name}: {status}"

# Define alert rules
alerts = [
    MonitoringAlert(
        name="High MAE",
        severity="critical",
        threshold=baseline_metrics['mae'] * 1.3,  # 30% worse
        description="Model MAE is 30% worse than baseline"
    ),
    MonitoringAlert(
        name="Moderate MAE",
        severity="warning",
        threshold=baseline_metrics['mae'] * 1.2,  # 20% worse
        description="Model MAE is 20% worse than baseline"
    ),
    MonitoringAlert(
        name="Low R²",
        severity="critical",
        threshold=baseline_metrics['r2'] * 0.9,  # 10% drop
        description="Model R² dropped more than 10%"
    ),
```

Exercise 5.2: Run alert checks (15 min)

```
# Run alert checks on production data
alert_log = []

for week_idx in range(len(production_perf)):
    week_mae = production_perf.iloc[week_idx]['mae']
    week_r2 = production_perf.iloc[week_idx]['r2']
    week_drift = drift_results.iloc[week_idx]['dataset_drift']

    week_alerts = []

    # Check MAE alerts
    for alert in alerts:
        if alert.name == "High MAE":
            if alert.check(week_mae, week_idx):
                week_alerts.append(alert)
        elif alert.name == "Moderate MAE":
            if alert.check(week_mae, week_idx):
                week_alerts.append(alert)
        elif alert.name == "Low R²":
            # Inverted logic for R² (lower is worse)
            if week_r2 < alert.threshold:
                if not alert.triggered:
                    alert.triggered = True
                    alert.trigger_week = week_idx
                    alert.trigger_value = week_r2
                    week_alerts.append(alert)
        elif alert.name == "Data Drift Detected":
            if week_drift:
                if alert.trigger_week != week_idx:  # Log each week separately
                    alert_copy = MonitoringAlert(
                        alert.name, alert.severity, alert.threshold, alert.description
                    )
                    alert_copy.triggered = True
                    alert_copy.trigger_week = week_idx
                    alert_copy.trigger_value = 1
                    week_alerts.append(alert_copy)

    # Log alerts for this week
    for triggered_alert in week_alerts:
        alert_log.append({
            'week': week_idx,
            'alert_name': triggered_alert.name,
            'severity': triggered_alert.severity,
            'description': triggered_alert.description,
            'value': triggered_alert.trigger_value,
            'timestamp': production_perf.iloc[week_idx]['start_date']
        })

# Convert to DataFrame
alert_log_df = pd.DataFrame(alert_log)

if len(alert_log_df) > 0:
    print(f"\n
⚠️
 {len(alert_log_df)} alerts triggered:")
    print(alert_log_df[['week', 'alert_name', 'severity', 'description']].head(20))

    # Save alert log
    alert_log_df.to_csv('alert_log.csv', index=False)
    print("\n✓ Alert log saved to alert_log.csv")

    # Summary by severity
    print("\nAlerts by severity:")
    print(alert_log_df['severity'].value_counts())
else:
    print("\n✓ No alerts triggered")
```

Exercise 5.2 (continued): Alert notification simulation

```
def send_alert_notification(alert_dict):
    """Simulate sending an alert (e.g., to Slack, email)."""
    severity_emoji = {
        'critical': '
🚨
',
        'warning': '
⚠️
',
        'info': '
ℹ️
'
    }

    emoji = severity_emoji.get(alert_dict['severity'], '
📢
')

    message = f"""
{emoji} Model Monitoring Alert

Alert: {alert_dict['alert_name']}
Severity: {alert_dict['severity'].upper()}
Week: {alert_dict['week']}
Timestamp: {alert_dict['timestamp']}

Description: {alert_dict['description']}
Value: {alert_dict['value']:.4f}

Action required: Review model performance and consider retraining.
    """

    return message

# Simulate sending alerts
if len(alert_log_df) > 0:
    print("\n" + "="*60)
    print("SIMULATED ALERT NOTIFICATIONS")
    print("="*60)

    # Send first 5 alerts as examples
    for idx, alert in alert_log_df.head(5).iterrows():
        notification = send_alert_notification(alert)
        print(notification)
        print("-"*60)
else:
    print("\nNo alerts to send")

print("\n✓ Alert notification system ready")
```

Part 6: Performance Degradation Analysis (35 min)

Goal: Analyze why the model degraded and identify root causes.

Exercise 6.1: Compare distributions (20 min)

```
# Compare reference vs production data distributions
import matplotlib.pyplot as plt
import seaborn as sns

# Aggregate all production data
all_production_data = pd.concat(batches, ignore_index=True)

# Compare distributions for key features
key_features = ['temp', 'atemp', 'hum', 'windspeed', 'weathersit']

fig, axes = plt.subplots(3, 2, figsize=(14, 12))
axes = axes.flatten()

for i, feature in enumerate(key_features):
    # Reference distribution
    axes[i].hist(
        reference_data[feature],
        bins=30,
        alpha=0.5,
        label='Reference (2011)',
        color='blue',
        density=True
    )

    # Production distribution
    axes[i].hist(
        all_production_data[feature],
        bins=30,
        alpha=0.5,
        label='Production (2012)',
        color='orange',
        density=True
    )

    axes[i].set_xlabel(feature)
    axes[i].set_ylabel('Density')
    axes[i].set_title(f'Distribution: {feature}')
    axes[i].legend()
    axes[i].grid(alpha=0.3)

# Target distribution
axes[5].hist(
    reference_data[target_col],
    bins=30,
    alpha=0.5,
    label='Reference (2011)',
    color='blue',
    density=True
)
axes[5].hist(
    all_production_data[target_col],
    bins=30,
    alpha=0.5,
    label='Production (2012)',
    color='orange',
    density=True
)
axes[5].set_xlabel('Bike Rentals (Target)')
axes[5].set_ylabel('Density')
axes[5].set_title('Target Distribution')
```

Exercise 6.2: Statistical drift analysis (15 min)

```python
from scipy.stats import ks_2samp, wasserstein_distance

# Perform statistical tests
statistical_analysis = []

for feature in feature_cols + [target_col]:
    ref_values = reference_data[feature].values
    prod_values = all_production_data[feature].values

    # KS test
    ks_stat, ks_pval = ks_2samp(ref_values, prod_values)

    # Wasserstein distance
    wasserstein_dist = wasserstein_distance(ref_values, prod_values)

    # Mean and std comparison
    ref_mean, ref_std = ref_values.mean(), ref_values.std()
    prod_mean, prod_std = prod_values.mean(), prod_values.std()

    mean_change_pct = ((prod_mean - ref_mean) / ref_mean) * 100

    statistical_analysis.append({
        'feature': feature,
        'ks_statistic': ks_stat,
        'ks_pvalue': ks_pval,
        'wasserstein_distance': wasserstein_dist,
        'ref_mean': ref_mean,
        'prod_mean': prod_mean,
        'mean_change_%': mean_change_pct,
        'ref_std': ref_std,
        'prod_std': prod_std,
        'drift_detected': ks_pval < 0.05
    })

# Convert to DataFrame
stats_df = pd.DataFrame(statistical_analysis)

print("\nStatistical Drift Analysis:")
print(stats_df[['feature', 'ks_pvalue', 'mean_change_%', 'drift_detected']].to_string(index=False))

# Identify most drifted features
drifted_features = stats_df[stats_df['drift_detected'] == True].sort_values(
    'wasserstein_distance', ascending=False
)

print(f"\n{len(drifted_features)} features with detected drift:")
print(drifted_features[['feature', 'ks_pvalue', 'mean_change_%']].to_string(index=False))

# Visualize drift scores
plt.figure(figsize=(12, 6))

# Sort by wasserstein distance
stats_df_sorted = stats_df.sort_values('wasserstein_distance', ascending=True)

colors = ['red' if drift else 'green' for drift in stats_df_sorted['drift_detected']]

plt.barh(stats_df_sorted['feature'], stats_df_sorted['wasserstein_distance'], color=colors, alpha=0.7)
plt.xlabel('Wasserstein Distance (Drift Score)')
plt.title('Feature Drift Scores (Red = Significant Drift)')
plt.grid(alpha=0.3)
plt.tight_layout()
plt.savefig('drift_scores.png', dpi=150)
plt.show()

# Save analysis
stats_df.to_csv('statistical_drift_analysis.csv', index=False)
print("\n✓ Statistical analysis saved to statistical_drift_analysis.csv")
```

Exercise 6.3: Root cause analysis

```python
# Analyze root causes of performance degradation

print("\n" + "="*70)
print("ROOT CAUSE ANALYSIS")
print("="*70)

# 1. Temporal analysis
print("\n1. TEMPORAL PATTERNS:")
print(f"   Reference period: {reference_data['dteday'].min()} to {reference_data['dteday'].max()}")
print(f"   Production period: {all_production_data['dteday'].min()} to {all_production_data['dteday'].max()}")

# 2. Feature drift impact
print("\n2. TOP 5 DRIFTED FEATURES:")
top_drifted = stats_df.nlargest(5, 'wasserstein_distance')
for idx, row in top_drifted.iterrows():
    print(f"   {row['feature']:15s}: "
          f"Wasserstein Distance = {row['wasserstein_distance']:.4f}, "
          f"Mean Change = {row['mean_change_%']:+.1f}%")

# 3. Performance correlation with drift
print("\n3. CORRELATION WITH PERFORMANCE:")

# Merge drift and performance data
merged = drift_results.merge(production_perf, on='week')

# Check if drift correlates with higher error
drifted_weeks_perf = merged[merged['dataset_drift'] == True]
no_drift_weeks_perf = merged[merged['dataset_drift'] == False]

if len(drifted_weeks_perf) > 0 and len(no_drift_weeks_perf) > 0:
    drift_mean_mae = drifted_weeks_perf['mae'].mean()
    no_drift_mean_mae = no_drift_weeks_perf['mae'].mean()

    print(f"   MAE (weeks with drift):    {drift_mean_mae:.2f}")
    print(f"   MAE (weeks without drift): {no_drift_mean_mae:.2f}")
    print(f"   Difference: {drift_mean_mae - no_drift_mean_mae:+.2f} "
          f"({((drift_mean_mae - no_drift_mean_mae) / no_drift_mean_mae * 100):+.1f}%)")

# 4. Recommendations
print("\n4. RECOMMENDATIONS:")
recommendations = []

# Check if target drifted
target_drift = stats_df[stats_df['feature'] == target_col].iloc[0]
if target_drift['drift_detected']:
    recommendations.append("
 Target distribution changed - model retraining recommended")

# Check if many features drifted
n_drifted = stats_df['drift_detected'].sum()
if n_drifted > len(feature_cols) / 2:
    recommendations.append(f"
 {n_drifted}/{len(feature_cols)} features drifted - consider data quality checks")

# Check performance degradation
mean_mae_increase = ((production_perf['mae'].mean() - baseline_metrics['mae']) / baseline_metrics['mae']) * 100
if mean_mae_increase > 15:
    recommendations.append(f"
 Performance degraded {mean_mae_increase:.1f}% - immediate retraining needed")

# Check if specific features drifted significantly
high_drift_features = stats_df[stats_df['wasserstein_distance'] > stats_df['wasserstein_distance'].quantile(0.75)]
if len(high_drift_features) > 0:
    feat_names = ', '.join(high_drift_features['feature'].values)
    recommendations.append(f"
 Focus retraining on features: {feat_names}")

for i, rec in enumerate(recommendations, 1):
    print(f"   {i}. {rec}")

if len(recommendations) == 0:
    print("   ✓ No critical issues detected - continue monitoring")

print("\n" + "="*70)
```

Deliverables Summary

Create a summary report:

```
# Create comprehensive summary report
summary_report = f"""
# MODEL MONITORING LAB - SUMMARY REPORT

## Executive Summary

**Model**: Random Forest Regressor for Bike Sharing Demand
**Training Period**: 2011 (Reference Data)
**Production Period**: 2012 (52 weeks)

## Baseline Performance
- MAE: {baseline_metrics['mae']:.2f}
- RMSE: {baseline_metrics['rmse']:.2f}
- R²: {baseline_metrics['r2']:.4f}

## Production Performance
- Mean MAE: {production_perf['mae'].mean():.2f} (±{production_perf['mae'].std():.2f})
- Mean RMSE: {production_perf['rmse'].mean():.2f} (±{production_perf['rmse'].std():.2f})
- Mean R²: {production_perf['r2'].mean():.4f} (±{production_perf['r2'].std():.4f})

## Degradation Analysis
- Performance Degradation: {((production_perf['mae'].mean() - baseline_metrics['mae']) / baseline_metrics['mae'] * 100):+.1f}%
- Weeks with Drift: {drift_results['dataset_drift'].sum()} / {len(drift_results)}
- Alerts Triggered: {len(alert_log_df) if len(alert_log_df) > 0 else 0}

## Key Findings
1. Data drift detected in {stats_df['drift_detected'].sum()}/{len(stats_df)} features
2. Most drifted features: {', '.join(top_drifted.head(3)['feature'].values)}
3. Performance degradation correlates with drift: {"Yes" if len(drifted_weeks_perf) > 0 and drift_mean_mae > no_drift_mean_mae else "No"}

## Files Generated
- monitoring_dashboard.html (Interactive dashboard)
- drift_reports/week_XX_drift_report.html (Per-week drift reports)
- alert_log.csv (Alert history)
- statistical_drift_analysis.csv (Detailed drift statistics)
- production_performance.csv (Performance metrics over time)

## Next Steps
{chr(10).join(['- ' + rec for rec in recommendations])}

---
Report Generated: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}
"""

# Save summary report
with open('MONITORING_SUMMARY_REPORT.md', 'w') as f:
    f.write(summary_report)

print(summary_report)
print("\n✓ Summary report saved to MONITORING_SUMMARY_REPORT.md")
```

Bonus: Automated Retraining Trigger

Optional exercise: Implement automated retraining logic

```
# Define retraining trigger logic
def should_retrain(performance_metrics, drift_metrics, alert_log):
    """Determine if model should be retrained."""

    retrain_reasons = []

    # Rule 1: Performance degradation > 20%
    recent_mae = performance_metrics.tail(5)['mae'].mean()
    if recent_mae > baseline_metrics['mae'] * 1.2:
        retrain_reasons.append(f"Recent MAE ({recent_mae:.2f}) > 20% worse than baseline")

    # Rule 2: Consistent drift for 4+ consecutive weeks
    recent_drift = drift_metrics.tail(10)['dataset_drift'].sum()
    if recent_drift >= 4:
        retrain_reasons.append(f"Drift detected in {recent_drift}/10 recent weeks")

    # Rule 3: Critical alerts
    if len(alert_log) > 0:
        critical_alerts = alert_log[alert_log['severity'] == 'critical']
        if len(critical_alerts) >= 3:
            retrain_reasons.append(f"{len(critical_alerts)} critical alerts triggered")

    # Rule 4: Time-based (optional)
    # weeks_since_training = len(performance_metrics)
    # if weeks_since_training > 26:  # 6 months
    #     retrain_reasons.append("6 months since last training")

    should_trigger = len(retrain_reasons) > 0

    return should_trigger, retrain_reasons

# Check if retraining should be triggered
should_retrain_flag, reasons = should_retrain(
    production_perf,
    drift_results,
    alert_log_df if len(alert_log_df) > 0 else pd.DataFrame()
)

print("\n" + "="*70)
print("RETRAINING DECISION")
print("="*70)

if should_retrain_flag:
    print("\n
🔄
 RETRAINING RECOMMENDED\n")
    print("Reasons:")
    for i, reason in enumerate(reasons, 1):
        print(f"  {i}. {reason}")

    print("\nAction items:")
    print("  1. Collect all production data with labels")
    print("  2. Combine with historical training data")
    print("  3. Retrain model with updated dataset")
    print("  4. Validate new model on holdout set")
    print("  5. Deploy if performance improves")
else:
    print("\n✓ NO RETRAINING NEEDED\n")
    print("Current model performance is acceptable.")
    print("Continue monitoring for future degradation.")

print("="*70)
```

Final Summary

Congratulations! You have completed the Model Monitoring lab.

What you accomplished:

1. ✅ Trained baseline model and established reference metrics
2. ✅ Simulated production environment with data drift
3. ✅ Implemented drift detection using Evidently AI
4. ✅ Created interactive monitoring dashboard
5. ✅ Built automated alerting system
6. ✅ Performed root cause analysis

Additional Challenges (Optional)

Challenge 1: Real-time monitoring (45 min)

- Simulate streaming data
- Implement rolling window drift detection
- Update dashboard in real-time

Challenge 2: Multi-model monitoring (60 min)

- Train multiple models (RF, GB, Linear)
- Track all models simultaneously
- Implement model selection logic

Challenge 3: Custom drift metrics (30 min)

- Implement Population Stability Index (PSI)
- Create custom drift score

Additional Resources

Documentation:

- Evidently AI: https://docs.evidentlyai.com/
- Alibi Detect: https://docs.seldon.io/projects/alibi-detect/
- Great Expectations: https://docs.greatexpectations.io/

Tutorials:

- Evidently Tutorials: https://github.com/evidentlyai/evidently/tree/main/examples
- Model Monitoring Best Practices: https://huyenchip.com/

Papers:

- "A Survey on Concept Drift Adaptation" (Gama et al., 2014)
- "Failing Loudly: An Empirical Study of Methods for Detecting Dataset Shift" (Rabanser et al., 2019)