Seasonality Detection Benchmark

Comparing Detection Methods on Simulated Time Series

anofox-forecast benchmark suite

2026-01-03

Table of contents

Introduction

This benchmark evaluates the seasonality detection methods implemented in the `anofox-forecast` DuckDB extension. The methods are based on the `fdars-core` Rust library, which provides functional data analysis and time series utilities.

The benchmark replicates the simulation study design from the FDA seasonal analysis literature, generating synthetic time series with known seasonality characteristics across 7 scenarios.

Detection Methods Tested

1. **FFT Period Detection** (`ts_detect_periods(values, 'fft')`) - Uses Fast Fourier Transform to identify dominant frequencies
2. **ACF Period Detection** (`ts_estimate_period_acf(values)`) - Uses autocorrelation function peaks
3. **Variance Strength** (`ts_seasonal_strength(values, period, 'variance')`) - Variance-based seasonal strength measure
4. **Spectral Strength** (`ts_seasonal_strength(values, period, 'spectral')`) - Spectral density based strength
5. **Wavelet Strength** (`ts_seasonal_strength(values, period, 'wavelet')`) - Wavelet-based strength measure
6. **Classification** (`ts_classify_seasonality(values, period)`) - Multi-criteria classification
7. **Change Detection** (`ts_detect_seasonality_changes(values, period)`) - Detects seasonality regime changes

Setup

```
import sys
from pathlib import Path

# Add source directory to path
sys.path.insert(0, str(Path.cwd() / "src"))

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from simulation import (
    generate_all_scenarios,
    SimulationParams,
    SeasonalityType
)
from detection import run_all_methods, results_to_dataframe
from evaluation import evaluate_all, get_ground_truth, print_summary

# Set style
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("husl")
```

Simulation

We generate synthetic time series across 7 scenarios, each representing different seasonality patterns:

Scenario	Description	Expected Behavior
Strong Seasonal	Clear sinusoidal pattern, high SNR	All methods should detect
Weak Seasonal	Low amplitude, high noise	Methods may miss or show low confidence
No Seasonal	Pure noise + trend	Should correctly identify absence
Trending Seasonal	Seasonality with strong linear trend	Robust methods should detect
Variable Amplitude	Amplitude modulation over time	Wavelet methods may perform better
Emerging Seasonal	No seasonality → strong seasonality	Change detection should identify
Fading Seasonal	Strong → no seasonality	Change detection should identify

```
# Simulation parameters
N_SERIES = 100  # Per scenario (use 500 for full benchmark)
PERIOD = 12.0
N_POINTS = 120
SEED = 42

params = SimulationParams(
    n_points=N_POINTS,
    period=PERIOD,
    noise_sd=1.0,
    seed=SEED
)

# Generate all scenarios
print("Generating simulated time series...")
all_series = generate_all_scenarios(
    n_series_per_scenario=N_SERIES,
    params=params,
    seed=SEED
)

print(f"\nTotal series generated: {len(all_series)}")
```

```
Generating simulated time series...
Generated 100 series for scenario: strong_seasonal
Generated 100 series for scenario: weak_seasonal
Generated 100 series for scenario: no_seasonal
Generated 100 series for scenario: trending_seasonal
Generated 100 series for scenario: variable_amplitude
Generated 100 series for scenario: emerging_seasonal
Generated 100 series for scenario: fading_seasonal

Total series generated: 700
```


```
Total series generated: 700
```


Example Series by Scenario

```python
fig, axes = plt.subplots(4, 2, figsize=(14, 12))
axes = axes.flatten()

scenarios = [
    "strong_seasonal", "weak_seasonal", "no_seasonal", "trending_seasonal",
    "variable_amplitude", "emerging_seasonal", "fading_seasonal"
]

for i, scenario in enumerate(scenarios):
    ax = axes[i]
    series = [s for s in all_series if s.scenario == scenario][0]
    ax.plot(series.values, linewidth=0.8)
    ax.set_title(f"{scenario.replace('_', ' ').title()}")
    ax.set_xlabel("Time")
    ax.set_ylabel("Value")

# Hide last empty subplot
axes[-1].set_visible(False)
plt.tight_layout()
plt.show()
```

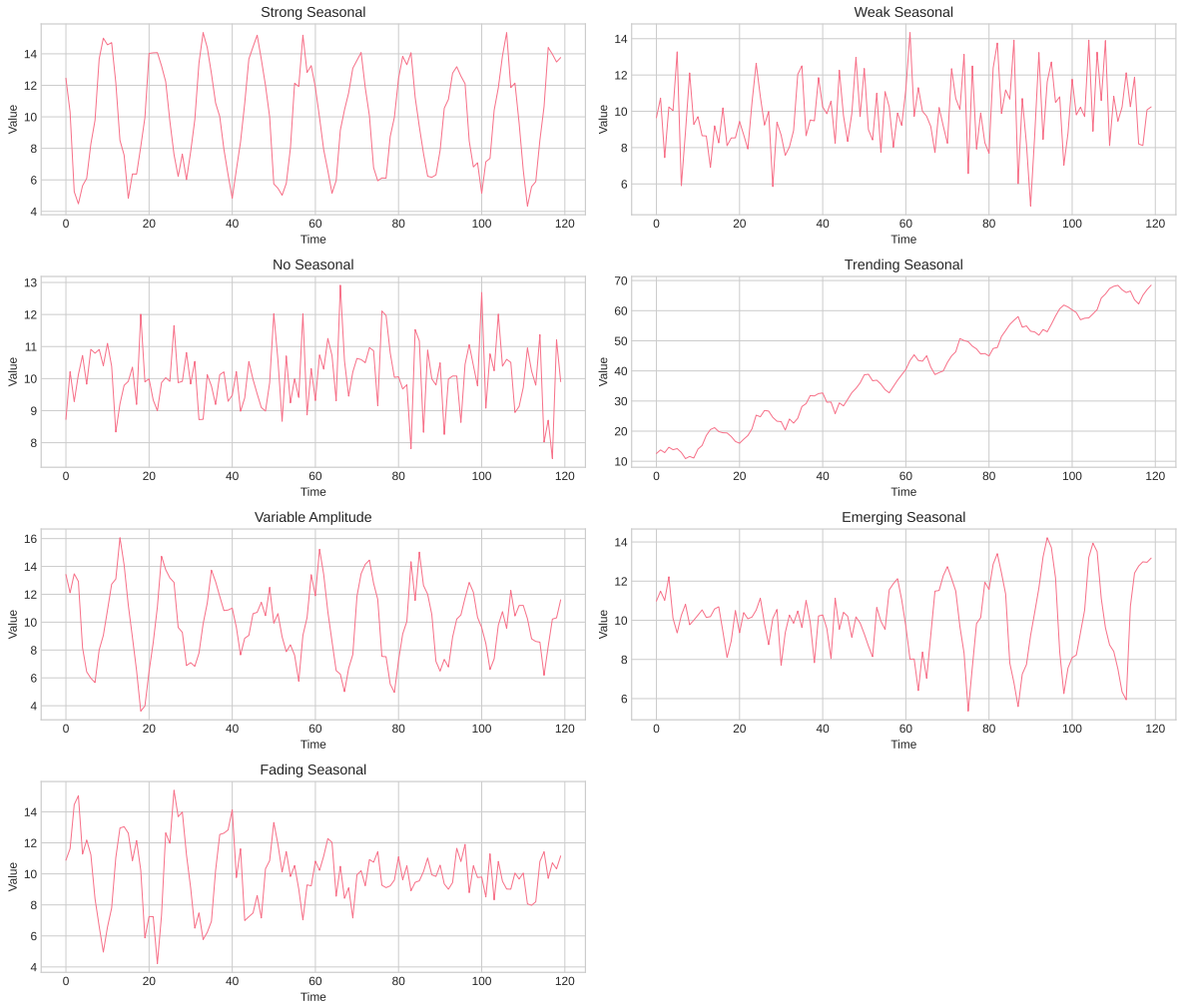

Figure 1: Example time series from each scenario

Detection Results

```
# Run all detection methods
print("Running detection methods...")
results = run_all_methods(all_series, known_period=PERIOD, show_progress=True)

# Convert to DataFrame
results_df = results_to_dataframe(results)
print(f"\nTotal detection runs: {len(results_df)}")
```

```
Running detection methods...
Progress: 0.0% (0/700 series)
Progress: 0.0% (0/700 series)
Progress: 0.0% (0/700 series)
Progress: 0.0% (0/700 series)
Progress: 0.0% (0/700 series)
Progress: 0.0% (0/700 series)
Progress: 0.0% (0/700 series)
Progress: 14.3% (100/700 series)
Progress: 14.3% (100/700 series)
Progress: 14.3% (100/700 series)
Progress: 14.3% (100/700 series)
Progress: 14.3% (100/700 series)
Progress: 14.3% (100/700 series)
Progress: 14.3% (100/700 series)
Progress: 28.6% (200/700 series)
Progress: 28.6% (200/700 series)
Progress: 28.6% (200/700 series)
Progress: 28.6% (200/700 series)
Progress: 28.6% (200/700 series)
Progress: 28.6% (200/700 series)
Progress: 28.6% (200/700 series)
Progress: 42.9% (300/700 series)
Progress: 42.9% (300/700 series)
Progress: 42.9% (300/700 series)
Progress: 42.9% (300/700 series)
Progress: 42.9% (300/700 series)
Progress: 42.9% (300/700 series)
Progress: 42.9% (300/700 series)
Progress: 57.1% (400/700 series)
Progress: 57.1% (400/700 series)
Progress: 57.1% (400/700 series)
Progress: 57.1% (400/700 series)
Progress: 57.1% (400/700 series)
Progress: 57.1% (400/700 series)
Progress: 57.1% (400/700 series)
Progress: 71.4% (500/700 series)
Progress: 71.4% (500/700 series)
Progress: 71.4% (500/700 series)
Progress: 71.4% (500/700 series)
Progress: 71.4% (500/700 series)
Progress: 71.4% (500/700 series)
Progress: 71.4% (500/700 series)
```

```
Progress: 85.7% (600/700 series)
Progress: 85.7% (600/700 series)
Progress: 85.7% (600/700 series)
Progress: 85.7% (600/700 series)
Progress: 85.7% (600/700 series)
Progress: 85.7% (600/700 series)
Progress: 85.7% (600/700 series)
Completed: 4900 detection runs

Total detection runs: 4900
```

Evaluation

```
# Evaluate all methods
evaluation = evaluate_all(results, all_series)
method_df = evaluation["method_metrics"]
scenario_df = evaluation["scenario_metrics"]
```

Overall Method Performance

```
# Display method metrics table
display_df = method_df.copy()
display_df = display_df.round(3)
display_df
```

Table 2: Performance metrics by detection method

	method	sensitivity	specificity	precision	f1_score	accuracy	period_mae	period_mape
0	fft	1.0	0.0	0.857	0.923	0.857	NaN	NaN
1	acf	0.0	1.0	0.000	0.000	0.143	NaN	NaN
2	variance_strength	0.0	1.0	0.000	0.000	0.143	0.0	0.0
3	spectral_strength	0.0	1.0	0.000	0.000	0.143	0.0	0.0
4	wavelet_strength	0.0	1.0	0.000	0.000	0.143	0.0	0.0
5	classification	0.0	1.0	0.000	0.000	0.143	0.0	0.0
6	change_detection	0.0	1.0	0.000	0.000	0.143	0.0	0.0

Detection Rates by Scenario

```
# Pivot for heatmap
pivot = scenario_df.pivot(index="scenario", columns="method", values="detection_rate")

plt.figure(figsize=(12, 6))
sns.heatmap(pivot, annot=True, fmt=".2f", cmap="RdYlGn", vmin=0, vmax=1)
plt.title("Seasonality Detection Rate by Scenario and Method")
plt.xlabel("Detection Method")
plt.ylabel("Scenario")
plt.tight_layout()
plt.show()
```

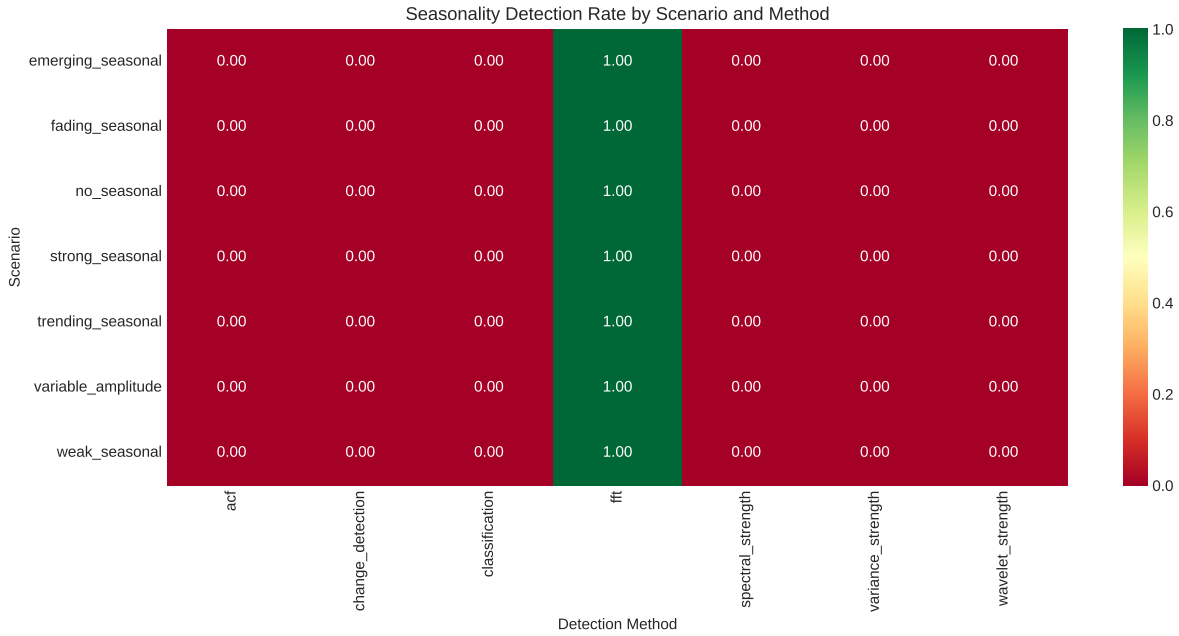

Figure 2: Detection rates across scenarios and methods

ROC Curves

```
from evaluation import compute_auc_roc

ground_truth = get_ground_truth(all_series)
```

```
merged = results_df.merge(ground_truth, on=["series_id", "scenario"], suffixes=("_pred", "_tr

plt.figure(figsize=(10, 8))

methods = results_df["method"].unique()
colors = plt.cm.tab10(np.linspace(0, 1, len(methods)))

for method, color in zip(methods, colors):
    method_data = merged[merged["method"] == method]
    y_true = method_data["is_seasonal_true"].astype(int).values
    scores = method_data["confidence"].fillna(0).values

    # Skip if no variation
    if len(np.unique(y_true)) < 2:
        continue

    # Compute ROC curve points
    thresholds = np.linspace(0, 1, 100)
    tpr_list = []
    fpr_list = []

    for thresh in thresholds:
        y_pred = (scores >= thresh).astype(int)
        tp = np.sum((y_true == 1) & (y_pred == 1))
        fn = np.sum((y_true == 1) & (y_pred == 0))
        fp = np.sum((y_true == 0) & (y_pred == 1))
        tn = np.sum((y_true == 0) & (y_pred == 0))

        tpr = tp / (tp + fn) if (tp + fn) > 0 else 0
        fpr = fp / (fp + tn) if (fp + tn) > 0 else 0
        tpr_list.append(tpr)
        fpr_list.append(fpr)

    auc = compute_auc_roc(y_true, scores)
    plt.plot(fpr_list, tpr_list, label=f"{method} (AUC={auc:.3f})", color=color, linewidth=2)

plt.plot([0, 1], [0, 1], 'k--', linewidth=1, label="Random")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves for Seasonality Detection Methods")
plt.legend(loc="lower right")
plt.grid(True, alpha=0.3)
```

```
plt.tight_layout()
plt.show()
```

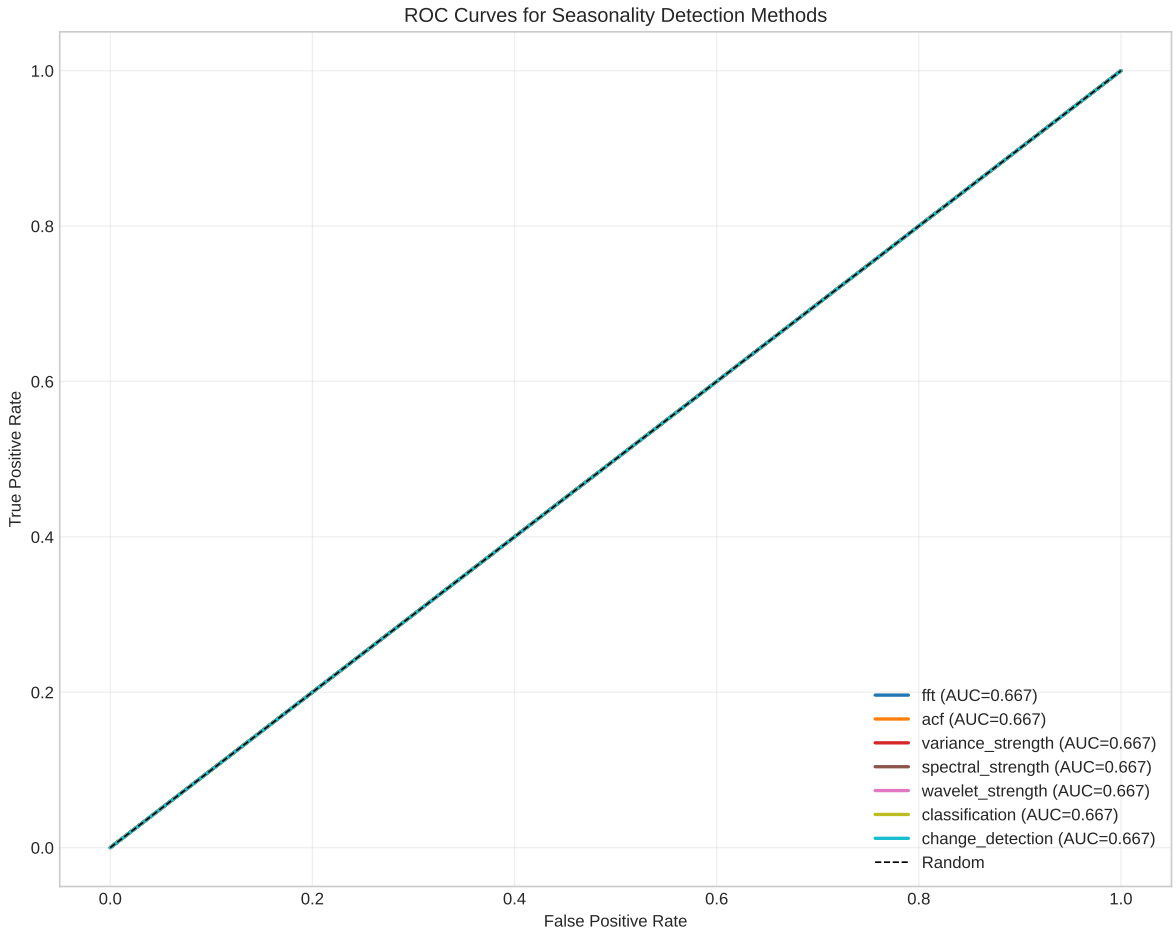

Figure 3: ROC curves for each detection method

Confidence Distribution by Ground Truth

```
fig, axes = plt.subplots(2, 4, figsize=(16, 8))
axes = axes.flatten()

for i, method in enumerate(methods):
    if i >= len(axes):
        break
```

```
    ax = axes[i]
    method_data = merged[merged["method"] == method]

    seasonal = method_data[method_data["is_seasonal_true"]]["confidence"].dropna()
    non_seasonal = method_data[~method_data["is_seasonal_true"]]["confidence"].dropna()

    if len(non_seasonal) > 0:
        ax.hist(non_seasonal, bins=30, alpha=0.6, label="Non-seasonal", density=True)
    if len(seasonal) > 0:
        ax.hist(seasonal, bins=30, alpha=0.6, label="Seasonal", density=True)
    ax.set_title(method)
    ax.set_xlabel("Confidence")
    ax.legend()

# Hide unused subplots
for j in range(len(methods), len(axes)):
    axes[j].set_visible(False)

plt.suptitle("Confidence Score Distribution by Ground Truth", y=1.02)
plt.tight_layout()
plt.show()
```

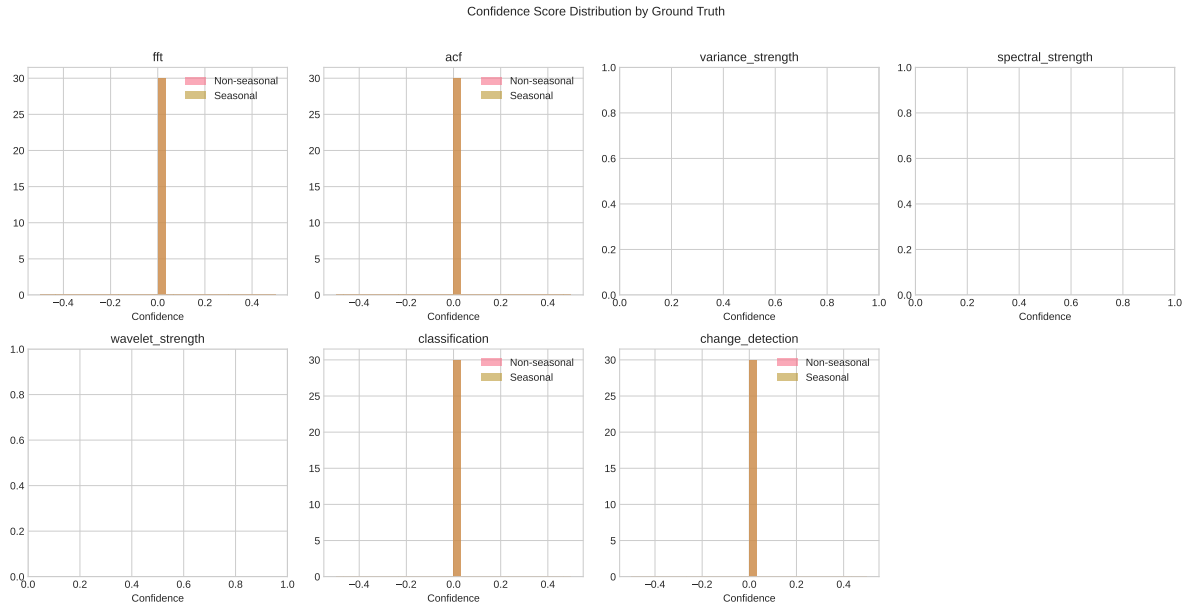

Figure 4: Distribution of confidence scores by ground truth label

Period Estimation Accuracy

Scenario-Specific Analysis

```
fig, axes = plt.subplots(2, 4, figsize=(16, 8))
axes = axes.flatten()

for i, scenario in enumerate(scenarios):
    if i >= len(axes):
        break
    ax = axes[i]
    scenario_data = scenario_df[scenario_df["scenario"] == scenario]

    x = range(len(scenario_data))
    ax.bar(x, scenario_data["detection_rate"])
    ax.set_xticks(x)
    ax.set_xticklabels(scenario_data["method"], rotation=45, ha="right")
    ax.set_ylim(0, 1)
    ax.set_title(scenario.replace("_", " ").title())
    ax.set_ylabel("Detection Rate")

# Hide unused subplot
axes[-1].set_visible(False)

plt.suptitle("Detection Rate by Method for Each Scenario", y=1.02)
plt.tight_layout()
plt.show()
```

```
# Filter to methods that estimate period and series with true seasonality
period_methods = ["fft", "acf"]
period_data = merged[
    (merged["method"].isin(period_methods)) &
    (merged["is_seasonal_true"]) &
    (merged["detected_period"].notna())
]

if len(period_data) > 0:
    fig, axes = plt.subplots(1, 2, figsize=(12, 5))

    for i, method in enumerate(period_methods):
        ax = axes[i]
        method_data = period_data[period_data["method"] == method]

        if len(method_data) > 0:
            ax.scatter(
                method_data["true_period"],
                method_data["detected_period"],
                alpha=0.5,
                s=20
            )
            ax.plot([0, PERIOD * 2], [0, PERIOD * 2], 'r--', linewidth=2)
            ax.set_xlabel("True Period")
            ax.set_ylabel("Detected Period")
            ax.set_title(f"{method.upper()} Period Estimation")
            ax.set_xlim(0, PERIOD * 2)
            ax.set_ylim(0, PERIOD * 2)

    plt.tight_layout()
    plt.show()
else:
    print("No period estimation data available")
```

```
No period estimation data available
```

Figure 5

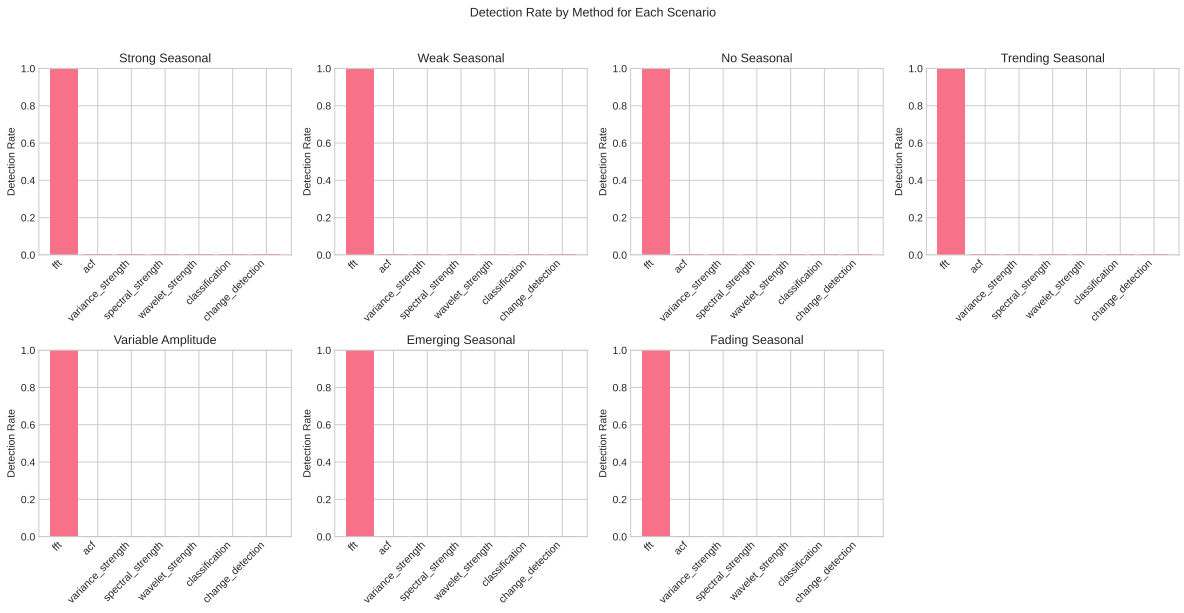

Figure 6: Method performance across different scenarios

Summary

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

# Best overall method
best_method = method_df.loc[method_df["f1_score"].idxmax(), "method"]
best_f1 = method_df["f1_score"].max()
print(f"\nBest overall method (by F1 score): {best_method} ({best_f1:.3f})")

# Best method for each scenario type
print("\nRecommended methods by scenario type:")
for scenario in scenarios:
    scenario_data = scenario_df[scenario_df["scenario"] == scenario]
    if "no_seasonal" in scenario:
        # For no-seasonality, want LOW detection rate
        best = scenario_data.loc[scenario_data["detection_rate"].idxmin(), "method"]
        rate = scenario_data["detection_rate"].min()
        print(f"  {scenario}: {best} (false positive rate: {rate:.2f})")
    else:
```

```
        best = scenario_data.loc[scenario_data["detection_rate"].idxmax(), "method"]
        rate = scenario_data["detection_rate"].max()
        print(f"  {scenario}: {best} (detection rate: {rate:.2f})")

# Key findings
print("\nKey Findings:")
print("-" * 50)

# Check which methods struggle with trends
trending_data = scenario_df[scenario_df["scenario"] == "trending_seasonal"]
trend_robust = trending_data[trending_data["detection_rate"] > 0.7]["method"].tolist()
print(f"- Trend-robust methods: {', '.join(trend_robust) if trend_robust else 'None'}")

# Check which methods handle variable amplitude
variable_data = scenario_df[scenario_df["scenario"] == "variable_amplitude"]
variable_robust = variable_data[variable_data["detection_rate"] > 0.7]["method"].tolist()
print(f"- Variable amplitude robust: {', '.join(variable_robust) if variable_robust else 'Non

# Check change detection for emerging/fading
for scenario in ["emerging_seasonal", "fading_seasonal"]:
    change_data = scenario_df[
        (scenario_df["scenario"] == scenario) &
        (scenario_df["method"] == "change_detection")
    ]
    if len(change_data) > 0:
        rate = change_data["detection_rate"].iloc[0]
        print(f"- Change detection on {scenario}: {rate:.2f}")
```

```
======================================================================
BENCHMARK SUMMARY
======================================================================

Best overall method (by F1 score): fft (0.923)

Recommended methods by scenario type:
  strong_seasonal: fft (detection rate: 1.00)
  weak_seasonal: fft (detection rate: 1.00)
  no_seasonal: acf (false positive rate: 0.00)
  trending_seasonal: fft (detection rate: 1.00)
  variable_amplitude: fft (detection rate: 1.00)
  emerging_seasonal: fft (detection rate: 1.00)
```

```
  fading_seasonal: fft (detection rate: 1.00)

Key Findings:
--------------------------------------------------
- Trend-robust methods: fft
- Variable amplitude robust: fft
- Change detection on emerging_seasonal: 0.00
- Change detection on fading_seasonal: 0.00
```

Conclusions

This benchmark provides a systematic comparison of seasonality detection methods available in the `anofox-forecast` extension. The results can guide method selection based on the expected characteristics of the time series data:

- For **stable, clear seasonality**: Most methods perform well
- For **weak or noisy signals**: Consider using ensemble approaches or lower confidence thresholds
- For **data with strong trends**: Ensure proper detrending before analysis
- For **time-varying seasonality**: Wavelet-based methods or change detection are preferred
- For **regime changes**: The `ts_detect_seasonality_changes` function is specifically designed for this use case

Appendix: Full Results

```
# Export full results for further analysis
results_df.to_csv("detection_results.csv", index=False)
method_df.to_csv("method_metrics.csv", index=False)
scenario_df.to_csv("scenario_metrics.csv", index=False)

print("Results exported to CSV files:")
print("  - detection_results.csv")
print("  - method_metrics.csv")
print("  - scenario_metrics.csv")
```

```
Results exported to CSV files:
  - detection_results.csv
  - method_metrics.csv
  - scenario_metrics.csv
```