Module 7: Hierarchical Models for Multiple Commodities

Learning Objectives

By the end of this module, you will be able to:

1. Understand hierarchical/multilevel model structures
2. Distinguish between complete pooling, no pooling, and partial pooling
3. Build hierarchical models with varying intercepts by commodity
4. Implement varying slopes for commodity-specific relationships
5. Model cross-commodity correlation structures
6. Apply shrinkage to improve estimates for thinly-traded commodities
7. Use hierarchical models for agricultural commodities with shared weather impacts

Why This Matters for Trading

Commodities don't exist in isolation. They form interconnected complexes:

Energy Complex:

- WTI (West Texas Intermediate) crude oil
- Brent crude oil
- Natural gas
- Share supply/demand drivers, but have distinct regional factors

Agricultural Complex:

- Corn, Wheat, Soybeans
- Compete for farmland
- Share weather risks (drought affects all)
- Different demand patterns (food, feed, fuel)

Metals Complex:

- Gold, Silver, Copper
- Economic growth affects all, but to varying degrees

Trading Advantages of Hierarchical Models

1. **Information Borrowing**: Thinly-traded commodities (e.g., oats) borrow strength from liquid (e.g., corn)
2. **Pairs Trading**: Identify when a commodity deviates from its complex (e.g., WTI-Brent sprea
3. **Risk Management**: Model correlations for portfolio optimization
4. **Shrinkage**: Regularize estimates to prevent overfitting
5. **Cross-Sectional Analysis**: Which commodity in a complex is most/least sensitive to a drive

6 . **Missing Data**: Forecast one commodity using information from others

Example: If drought affects corn, wheat, and soybeans, but you only have recent corn data, the hierarchical model uses corn's drought response to update beliefs about wheat and soybean res

This is **statistical arbitrage at the structural level**.

In [ ]:
```
# Standard imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import pymc as pm
import arviz as az
import warnings
warnings.filterwarnings('ignore')

np.random.seed(42)
plt.style.use('seaborn-v0_8-whitegrid')

print(f"PyMC version: {pm.__version__}")
print(f"ArviZ version: {az.__version__}")
```

1 . Introduction: Pooling Strategies

The Pooling Spectrum

Suppose we have price data for 3 commodities and want to estimate each commodity's m price.

Complete Pooling (ignore differences): $$ y_{ij} \sim \text{Normal}(\mu, \sigma) $$

- One μ for all commodities
- Assumes all commodities are identical
- **Problem**: Ignores commodity-specific factors

No Pooling (independent estimates): $$ y_{ij} \sim \text{Normal}(\mu_j, \sigma_j) $$

- Separate μ_j for each commodity j
- No information sharing
- **Problem**: Overfits with small samples, ignores commonalities

Partial Pooling (hierarchical): $$ \begin{align} y_{ij} &\sim \text{Normal}(\mu_j, \sigma) \\ \mu_j \text{Normal}(\mu_{\text{global}}, \tau) \end{align} $$

- Commodity-specific μ_j, but they share a common prior
- **Shrinkage**: Estimates pulled toward group mean
- **Best of both worlds**: Commodity differences + information borrowing

Trading Interpretation

- **Complete pooling**: "All energy commodities are the same" (wrong)
- **No pooling**: "WTI and Brent are unrelated" (misses correlation)

• **Partial pooling**: "They're different, but share common drivers" (realistic)

In [ ]:
```
# Generate synthetic data: 3 commodities with different means
np.random.seed(42)

# True parameters
true_global_mean = 60.0
true_tau = 8.0  # Between-commodity variation
true_sigma = 5.0  # Within-commodity variation

# 3 commodities: WTI, Brent, Natural Gas (converted to oil-equivalent)
commodities = ['WTI', 'Brent', 'NatGas']
n_commodities = len(commodities)
n_obs_per_commodity = [50, 60, 30]  # Different sample sizes

# Generate commodity-specific means
np.random.seed(42)
true_mu = np.array([58, 62, 55])  # WTI slightly below Brent, NatGas lowe

# Generate data
data_list = []
for j, (commodity, n_obs) in enumerate(zip(commodities, n_obs_per_commodi
    prices = np.random.normal(true_mu[j], true_sigma, n_obs)
    for price in prices:
        data_list.append({'commodity': commodity, 'price': price})

df_pooling = pd.DataFrame(data_list)

# Visualize
fig, ax = plt.subplots(figsize=(12, 6))

positions = [1, 2, 3]
for j, commodity in enumerate(commodities):
    subset = df_pooling[df_pooling['commodity'] == commodity]['price']
    ax.scatter([positions[j]] * len(subset), subset, alpha=0.5, s=50, lab
    ax.plot([positions[j] - 0.3, positions[j] + 0.3], [true_mu[j], true_m
            'r-', linewidth=3, label='True Mean' if j == 0 else '')

ax.axhline(true_global_mean, color='black', linestyle='--', linewidth=2,
           label=f'Global Mean = {true_global_mean}')
ax.set_xticks(positions)
ax.set_xticklabels(commodities)
ax.set_ylabel('Price ($/barrel equivalent)', fontsize=12)
ax.set_title('Energy Commodity Prices: Different Means, Shared Drivers',
ax.legend()
ax.grid(True, alpha=0.3, axis='y')
plt.tight_layout()
plt.show()

print("Data Summary:")
print(df_pooling.groupby('commodity')['price'].describe())
```

In [ ]:
```
# Encode commodities as integers for PyMC
commodity_idx = pd.Categorical(df_pooling['commodity'], categories=commod
y_pooling = df_pooling['price'].values

# Model 1: Complete Pooling
with pm.Model() as complete_pooling:
    mu_global = pm.Normal('mu_global', mu=60, sigma=20)
```

```
    sigma = pm.HalfNormal('sigma', sigma=10)

    y = pm.Normal('y', mu=mu_global, sigma=sigma, observed=y_pooling)

    trace_complete = pm.sample(2000, tune=1000, return_inferencedata=True

# Model 2: No Pooling
with pm.Model() as no_pooling:
    mu = pm.Normal('mu', mu=60, sigma=20, shape=n_commodities)
    sigma = pm.HalfNormal('sigma', sigma=10)

    y = pm.Normal('y', mu=mu[commodity_idx], sigma=sigma, observed=y_pool

    trace_no_pooling = pm.sample(2000, tune=1000, return_inferencedata=Tr

# Model 3: Partial Pooling (Hierarchical)
with pm.Model() as partial_pooling:
    # Hyperpriors (group-level)
    mu_global = pm.Normal('mu_global', mu=60, sigma=20)
    tau = pm.HalfNormal('tau', sigma=10)  # Between-commodity std

    # Commodity-specific means
    mu = pm.Normal('mu', mu=mu_global, sigma=tau, shape=n_commodities)

    # Observation model
    sigma = pm.HalfNormal('sigma', sigma=10)
    y = pm.Normal('y', mu=mu[commodity_idx], sigma=sigma, observed=y_pool

    trace_partial = pm.sample(2000, tune=1000, return_inferencedata=True,

print("\n=== COMPLETE POOLING ===")
print(az.summary(trace_complete, var_names=['mu_global', 'sigma']))

print("\n=== NO POOLING ===")
print(az.summary(trace_no_pooling, var_names=['mu', 'sigma']))

print("\n=== PARTIAL POOLING ===")
print(az.summary(trace_partial, var_names=['mu_global', 'tau', 'mu', 'sig
```

In [ ]:

```
# Compare estimates
complete_est = trace_complete.posterior['mu_global'].mean().item()
no_pooling_est = trace_no_pooling.posterior['mu'].mean(dim=['chain', 'dra
partial_pooling_est = trace_partial.posterior['mu'].mean(dim=['chain', 'd

# Visualization
fig, ax = plt.subplots(figsize=(12, 6))

x = np.arange(n_commodities)
width = 0.2

# True means
ax.bar(x - 1.5*width, true_mu, width, label='True', alpha=0.8, color='bla

# Complete pooling
ax.bar(x - 0.5*width, [complete_est]*n_commodities, width,
       label='Complete Pooling', alpha=0.7, color='red')

# No pooling
ax.bar(x + 0.5*width, no_pooling_est, width,
       label='No Pooling', alpha=0.7, color='blue')
```

```python
# Partial pooling
ax.bar(x + 1.5*width, partial_pooling_est, width,
       label='Partial Pooling', alpha=0.7, color='green')

ax.set_xlabel('Commodity', fontsize=12)
ax.set_ylabel('Estimated Mean Price ($/barrel)', fontsize=12)
ax.set_title('Pooling Strategies Comparison', fontsize=14, fontweight='bo
ax.set_xticks(x)
ax.set_xticklabels(commodities)
ax.legend()
ax.grid(True, alpha=0.3, axis='y')
plt.tight_layout()
plt.show()

# Quantify errors
print("\n=== ESTIMATION ERRORS (True - Estimated) ===")
print(f"\nComplete Pooling:")
for j, commodity in enumerate(commodities):
    error = true_mu[j] - complete_est
    print(f"  {commodity}: {error:+.2f}")
print(f"  RMSE: {np.sqrt(np.mean((true_mu - complete_est)**2)):.3f}")

print(f"\nNo Pooling:")
for j, commodity in enumerate(commodities):
    error = true_mu[j] - no_pooling_est[j]
    print(f"  {commodity}: {error:+.2f}")
print(f"  RMSE: {np.sqrt(np.mean((true_mu - no_pooling_est)**2)):.3f}")

print(f"\nPartial Pooling (Best):")
for j, commodity in enumerate(commodities):
    error = true_mu[j] - partial_pooling_est[j]
    print(f"  {commodity}: {error:+.2f}")
print(f"  RMSE: {np.sqrt(np.mean((true_mu - partial_pooling_est)**2)):.3f

print("\n→ Partial pooling has lowest RMSE, especially for small-sample c
```

2 . Shrinkage Effect

Theory

Partial pooling **shrinks** commodity-specific estimates toward the group mean:

$$ \hat{\mu}_j \approx \lambda \bar{y}_j + (1 - \lambda) \mu_{\text{global}} $$

Where:

- λ: Shrinkage factor (depends on sample size and τ/σ ratio)
- Small sample → low λ → strong shrinkage
- Large sample → high λ → weak shrinkage

Trading Application

- **Thinly-traded commodities**: Borrow strength from liquid ones
- **New contracts**: Use existing commodity data to initialize forecasts
- **Regularization**: Prevent overfitting on noisy data

In [ ]:
```
# Visualize shrinkage
fig, ax = plt.subplots(figsize=(12, 7))

# Sample means (raw data)
sample_means = df_pooling.groupby('commodity')['price'].mean().values

# Global mean
global_mean = partial_pooling_est.mean()

for j, commodity in enumerate(commodities):
    # Arrow from sample mean to hierarchical estimate
    ax.annotate('', xy=(j, partial_pooling_est[j]), xytext=(j, sample_mea
                arrowprops=dict(arrowstyle='->', lw=2, color='red', alpha

    # Sample mean
    ax.scatter(j, sample_means[j], s=200, color='blue', marker='o',
               edgecolors='black', linewidths=2, zorder=5, label='Sample

    # Hierarchical estimate
    ax.scatter(j, partial_pooling_est[j], s=200, color='green', marker='s
               edgecolors='black', linewidths=2, zorder=5, label='Hierarc

    # True mean
    ax.scatter(j, true_mu[j], s=200, color='red', marker='^',
               edgecolors='black', linewidths=2, zorder=5, label='True Me

    # Sample size annotation
    ax.text(j, sample_means[j] + 1.5, f'n={n_obs_per_commodity[j]}',
            ha='center', fontsize=10, fontweight='bold')

# Global mean line
ax.axhline(global_mean, color='purple', linestyle='--', linewidth=2,
           label=f'Global Mean = {global_mean:.2f}', alpha=0.7)

ax.set_xticks(range(n_commodities))
```

```
ax.set_xticklabels(commodities)
ax.set_ylabel('Price ($/barrel)', fontsize=12)
ax.set_title('Shrinkage Effect: Sample Means → Hierarchical Estimates', f
ax.legend(loc='upper right')
ax.grid(True, alpha=0.3, axis='y')
plt.tight_layout()
plt.show()

print("\n=== SHRINKAGE ANALYSIS ===")
for j, commodity in enumerate(commodities):
    shrinkage = sample_means[j] - partial_pooling_est[j]
    print(f"\n{commodity} (n={n_obs_per_commodity[j]}):")
    print(f"  Sample mean: {sample_means[j]:.2f}")
    print(f"  Hierarchical estimate: {partial_pooling_est[j]:.2f}")
    print(f"  Shrinkage: {shrinkage:+.2f} (toward global mean)")
    print(f"  → {'STRONG' if abs(shrinkage) > 1 else 'WEAK'} shrinkage ({
```

3 . Varying Intercepts and Slopes

Theory

Extend hierarchical models to regression with commodity-specific intercepts AND slopes:

$$ \begin{align} y_{ij} &\sim \text{Normal}(\mu_{ij}, \sigma) \\ \mu_{ij} &= \alpha_j + \beta_j x_i \\ \
&\sim \text{Normal}(\mu_\alpha, \tau_\alpha) \\ \beta_j &\sim \text{Normal}(\mu_\beta, \tau_\beta
\end{align} $$

Where:

- α_j: Commodity-specific intercept (base price)
- β_j: Commodity-specific slope (sensitivity to predictor)
- μ_α, τ_α: Group-level intercept parameters
- μ_β, τ_β: Group-level slope parameters

Trading Application

Question: How much does each energy commodity respond to USD changes?

- WTI might have $\beta_{\text{WTI}} = -1.2$ (highly sensitive)
- Brent might have $\beta_{\text{Brent}} = -0.9$ (less sensitive, European market)
- Natural gas might have $\beta_{\text{NatGas}} = -0.5$ (regional market, less USD-driver

Hierarchical model estimates these while preventing overfitting.

In [ ]:
```
# Generate data with varying intercepts and slopes
np.random.seed(42)
n_per_commodity = 80

# Predictor: USD index (standardized)
usd_values = np.random.normal(0, 1, n_per_commodity * n_commodities)

# True parameters
true_alpha = np.array([60, 64, 56])  # Different base prices
true_beta = np.array([-1.2, -0.9, -0.5])  # Different USD sensitivities
```

```
true_sigma_y = 3.0

# Generate prices
data_varying = []
for j, commodity in enumerate(commodities):
    idx_start = j * n_per_commodity
    idx_end = (j + 1) * n_per_commodity

    usd_subset = usd_values[idx_start:idx_end]
    prices = true_alpha[j] + true_beta[j] * usd_subset + np.random.normal

    for i, (usd, price) in enumerate(zip(usd_subset, prices)):
        data_varying.append({
            'commodity': commodity,
            'usd': usd,
            'price': price
        })

df_varying = pd.DataFrame(data_varying)

# Visualize
fig, axes = plt.subplots(1, 3, figsize=(18, 5), sharey=True)

colors = ['steelblue', 'darkorange', 'green']
for j, commodity in enumerate(commodities):
    subset = df_varying[df_varying['commodity'] == commodity]

    axes[j].scatter(subset['usd'], subset['price'], alpha=0.5, s=30, colo

    # True regression line
    usd_range = np.linspace(-3, 3, 100)
    axes[j].plot(usd_range, true_alpha[j] + true_beta[j] * usd_range,
                 'r--', linewidth=2, label=f'True: α={true_alpha[j]:.1f},

    axes[j].set_xlabel('USD Index (standardized)', fontsize=11)
    if j == 0:
        axes[j].set_ylabel('Price ($/barrel)', fontsize=11)
    axes[j].set_title(f'{commodity}', fontsize=12, fontweight='bold')
    axes[j].legend(loc='upper right')
    axes[j].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print("\nNote: Different slopes show varying USD sensitivity across commo
```

In [ ]:

```
# Build hierarchical model with varying intercepts and slopes
commodity_idx_varying = pd.Categorical(df_varying['commodity'], categorie
usd_varying = df_varying['usd'].values
y_varying = df_varying['price'].values

with pm.Model() as varying_model:
    # Hyperpriors for intercepts
    mu_alpha = pm.Normal('mu_alpha', mu=60, sigma=10)
    tau_alpha = pm.HalfNormal('tau_alpha', sigma=10)

    # Hyperpriors for slopes
    mu_beta = pm.Normal('mu_beta', mu=-1, sigma=2)
    tau_beta = pm.HalfNormal('tau_beta', sigma=1)
```

```
    # Commodity-specific intercepts and slopes
    alpha = pm.Normal('alpha', mu=mu_alpha, sigma=tau_alpha, shape=n_comm
    beta = pm.Normal('beta', mu=mu_beta, sigma=tau_beta, shape=n_commodit

    # Observation noise
    sigma_y = pm.HalfNormal('sigma_y', sigma=10)

    # Regression
    mu = alpha[commodity_idx_varying] + beta[commodity_idx_varying] * usd

    # Likelihood
    y = pm.Normal('y', mu=mu, sigma=sigma_y, observed=y_varying)

    # Sample
    trace_varying = pm.sample(2000, tune=1000, return_inferencedata=True,

print("\nPosterior Summary:")
print(az.summary(trace_varying, var_names=['mu_alpha', 'tau_alpha', 'mu_b
                                           'alpha', 'beta', 'sigma_y']))
```

In [ ]:

```
# Visualize estimates
alpha_est = trace_varying.posterior['alpha'].mean(dim=['chain', 'draw']).
beta_est = trace_varying.posterior['beta'].mean(dim=['chain', 'draw']).va

fig, axes = plt.subplots(1, 3, figsize=(18, 5), sharey=True)

for j, commodity in enumerate(commodities):
    subset = df_varying[df_varying['commodity'] == commodity]

    axes[j].scatter(subset['usd'], subset['price'], alpha=0.4, s=30, colo

    # True line
    usd_range = np.linspace(-3, 3, 100)
    axes[j].plot(usd_range, true_alpha[j] + true_beta[j] * usd_range,
                 'r--', linewidth=2, alpha=0.7, label=f'True: α={true_alp

    # Estimated line
    axes[j].plot(usd_range, alpha_est[j] + beta_est[j] * usd_range,
                 'b-', linewidth=2.5, label=f'Est: α={alpha_est[j]:.1f},

    axes[j].set_xlabel('USD Index (standardized)', fontsize=11)
    if j == 0:
        axes[j].set_ylabel('Price ($/barrel)', fontsize=11)
    axes[j].set_title(f'{commodity}', fontsize=12, fontweight='bold')
    axes[j].legend(loc='upper right', fontsize=9)
    axes[j].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Trading analysis
print("\n=== USD SENSITIVITY ANALYSIS ===")
for j, commodity in enumerate(commodities):
    beta_samples = trace_varying.posterior['beta'].values[:, :, j].flatte
    beta_mean = beta_samples.mean()
    beta_ci = np.percentile(beta_samples, [2.5, 97.5])

    print(f"\n{commodity}:")
    print(f"  β (USD): {beta_mean:.3f} (95% CI: [{beta_ci[0]:.3f}, {beta_
    print(f"  True β: {true_beta[j]:.3f}")
```

```
    if abs(beta_mean) > 1.0:
        print(f"  → HIGH SENSITIVITY: 1% USD move → {abs(beta_mean):.2f}%
    elif abs(beta_mean) > 0.7:
        print(f"  → MODERATE SENSITIVITY: 1% USD move → {abs(beta_mean):.
    else:
        print(f"  → LOW SENSITIVITY: USD less important driver")

# Group-level parameters
mu_beta_samples = trace_varying.posterior['mu_beta'].values.flatten()
print(f"\n\nGroup-level β (average across commodities): {mu_beta_samples.
print(f"Between-commodity variation (τ_β): {trace_varying.posterior['tau_
```

4. Cross-Commodity Correlation Structure

Theory

So far, we've assumed commodities are independent conditional on group parameters. But ener commodities are correlated due to shared shocks.

Multivariate hierarchical model:

$$ \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} \sim \text{MVNormal (\boldsymbol{\mu}_\alpha, \boldsymbol{\Sigma}_\alpha) $$

Where $\boldsymbol{\Sigma}_\alpha$ is a covariance matrix capturing correlations between con intercepts.

Trading Application

- **Pairs trading**: If WTI and Brent have correlation 0.95, large deviations signal arbitrag
- **Portfolio construction**: Use correlation matrix for optimal hedging
- **Risk management**: Diversification benefits depend on cross-commodity correlations

In [ ]:
```
# Generate correlated commodity returns
np.random.seed(42)
T_corr = 200

# True correlation matrix
# WTI-Brent: 0.95 (highly correlated)
# WTI-NatGas: 0.60
# Brent-NatGas: 0.55
true_corr = np.array([
    [1.00, 0.95, 0.60],
    [0.95, 1.00, 0.55],
    [0.60, 0.55, 1.00]
])

# Standard deviations
true_stds = np.array([2.0, 2.1, 3.5])  # NatGas more volatile

# Covariance matrix
true_cov = np.outer(true_stds, true_stds) * true_corr

# Generate multivariate normal returns
```

```
mean_returns = np.array([0.05, 0.06, 0.03])  # Daily returns (%)
returns = np.random.multivariate_normal(mean_returns, true_cov, T_corr)

# Convert to prices
prices_corr = np.zeros((T_corr, n_commodities))
prices_corr[0] = [60, 64, 56]
for t in range(1, T_corr):
    prices_corr[t] = prices_corr[t-1] * (1 + returns[t] / 100)

df_corr = pd.DataFrame(prices_corr, columns=commodities)

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

# Price series
for j, commodity in enumerate(commodities):
    axes[0, 0].plot(df_corr[commodity], label=commodity, linewidth=1.5, c
axes[0, 0].set_xlabel('Time (days)', fontsize=11)
axes[0, 0].set_ylabel('Price ($/barrel)', fontsize=11)
axes[0, 0].set_title('Correlated Commodity Prices', fontsize=12, fontweig
axes[0, 0].legend()
axes[0, 0].grid(True, alpha=0.3)

# Scatter: WTI vs Brent
axes[0, 1].scatter(df_corr['WTI'], df_corr['Brent'], alpha=0.5, s=20)
axes[0, 1].set_xlabel('WTI Price', fontsize=11)
axes[0, 1].set_ylabel('Brent Price', fontsize=11)
axes[0, 1].set_title(f'WTI vs Brent (ρ = {true_corr[0, 1]:.2f})', fontsiz
axes[0, 1].grid(True, alpha=0.3)

# Scatter: WTI vs NatGas
axes[1, 0].scatter(df_corr['WTI'], df_corr['NatGas'], alpha=0.5, s=20, co
axes[1, 0].set_xlabel('WTI Price', fontsize=11)
axes[1, 0].set_ylabel('NatGas Price', fontsize=11)
axes[1, 0].set_title(f'WTI vs NatGas (ρ = {true_corr[0, 2]:.2f})', fontsi
axes[1, 0].grid(True, alpha=0.3)

# Correlation heatmap
im = axes[1, 1].imshow(true_corr, cmap='RdYlGn', vmin=-1, vmax=1)
axes[1, 1].set_xticks(range(n_commodities))
axes[1, 1].set_yticks(range(n_commodities))
axes[1, 1].set_xticklabels(commodities)
axes[1, 1].set_yticklabels(commodities)
axes[1, 1].set_title('True Correlation Matrix', fontsize=12, fontweight='
for i in range(n_commodities):
    for j in range(n_commodities):
        axes[1, 1].text(j, i, f'{true_corr[i, j]:.2f}', ha='center', va='
plt.colorbar(im, ax=axes[1, 1])

plt.tight_layout()
plt.show()

print("Sample correlation matrix:")
print(df_corr.corr())
```

In [ ]:

```
# Build hierarchical model with correlated effects
# For simplicity, we'll estimate correlations from returns rather than fu

# Calculate returns
returns_df = df_corr.pct_change().dropna() * 100  # Percentage returns
```

```
# Stack data for PyMC
returns_stacked = returns_df.values.flatten()
commodity_idx_corr = np.repeat(np.arange(n_commodities), len(returns_df))

with pm.Model() as corr_model:
    # Mean returns by commodity
    mu_return = pm.Normal('mu_return', mu=0, sigma=1, shape=n_commodities

    # Cholesky decomposition for covariance
    sd_dist = pm.HalfNormal.dist(sigma=5, shape=n_commodities)
    chol, corr, stds = pm.LKJCholeskyCov('chol', n=n_commodities, eta=2.0

    # Observation model (simplified: independent given parameters)
    # In practice, you'd use multivariate normal
    sigma_obs = pm.Deterministic('sigma_obs', stds)

    # For computational efficiency, use independent normal per commodity
    # (Full multivariate model would be pm.MvNormal)
    y = pm.Normal('y', mu=mu_return[commodity_idx_corr],
                  sigma=sigma_obs[commodity_idx_corr],
                  observed=returns_stacked)

    # Sample
    trace_corr = pm.sample(2000, tune=1000, return_inferencedata=True,
                           target_accept=0.95, random_seed=42, chains=2)

print("\nPosterior Summary:")
print(az.summary(trace_corr, var_names=['mu_return', 'sigma_obs', 'corr']
```

In [ ]:

```
# Extract estimated correlation matrix
corr_est = trace_corr.posterior['corr'].mean(dim=['chain', 'draw']).value

# Visualize comparison
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# True correlations
im1 = axes[0].imshow(true_corr, cmap='RdYlGn', vmin=-1, vmax=1)
axes[0].set_xticks(range(n_commodities))
axes[0].set_yticks(range(n_commodities))
axes[0].set_xticklabels(commodities)
axes[0].set_yticklabels(commodities)
axes[0].set_title('True Correlations', fontsize=12, fontweight='bold')
for i in range(n_commodities):
    for j in range(n_commodities):
        axes[0].text(j, i, f'{true_corr[i, j]:.2f}', ha='center', va='cen
plt.colorbar(im1, ax=axes[0])

# Estimated correlations
im2 = axes[1].imshow(corr_est, cmap='RdYlGn', vmin=-1, vmax=1)
axes[1].set_xticks(range(n_commodities))
axes[1].set_yticks(range(n_commodities))
axes[1].set_xticklabels(commodities)
axes[1].set_yticklabels(commodities)
axes[1].set_title('Estimated Correlations (Bayesian)', fontsize=12, fontw
for i in range(n_commodities):
    for j in range(n_commodities):
        axes[1].text(j, i, f'{corr_est[i, j]:.2f}', ha='center', va='cent
plt.colorbar(im2, ax=axes[1])
```

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

print("\n=== TRADING INSIGHTS ===")
print(f"\nWTI-Brent Correlation: {corr_est[0, 1]:.3f}")
if corr_est[0, 1] > 0.9:
    print("  → HIGHLY CORRELATED: WTI-Brent spread trades require small d
    print("     Strategy: Mean-reversion on spread, tight stop-loss")

print(f"\nWTI-NatGas Correlation: {corr_est[0, 2]:.3f}")
print(f"Brent-NatGas Correlation: {corr_est[1, 2]:.3f}")
if corr_est[0, 2] < 0.7:
    print("  → MODERATE CORRELATION: NatGas provides diversification")
    print("     Strategy: Use NatGas to hedge crude oil exposure (imperfe
```

5 . Practical Application: Agricultural Commodities with Shared Weather

Trading Context

Corn, Wheat, and Soybeans:

- **Compete for farmland**: Farmers allocate acres based on expected profits
- **Share weather risks**: Drought in the Midwest affects all three
- **Different markets**: Corn (feed, ethanol), Wheat (food), Soybeans (feed, oil)

We'll model:

- Varying intercepts (different base prices)
- Varying slopes for rainfall impact
- Shared seasonality (harvest depression)

In [ ]:
```
# Generate agricultural commodity data
np.random.seed(42)
T_ag = 120  # 10 years of monthly data

ag_commodities = ['Corn', 'Wheat', 'Soybeans']
n_ag = len(ag_commodities)

# Rainfall index (standardized)
rainfall = np.random.normal(0, 1, T_ag)

# Month of year (for seasonality)
month = np.arange(T_ag) % 12

# Harvest months: September-October (months 8-9)
harvest_effect = np.where((month == 8) | (month == 9), -20, 0)

# True parameters
true_alpha_ag = np.array([400, 600, 1100])  # Base prices (cents/bushel)
true_beta_rain = np.array([-15, -12, -18])  # Rainfall effect (drought →

# Generate prices
prices_ag = np.zeros((T_ag, n_ag))
for j in range(n_ag):
    trend = true_alpha_ag[j] + 0.5 * np.arange(T_ag)  # Slight uptrend
```

```
    prices_ag[:, j] = (trend +
                       true_beta_rain[j] * rainfall +
                       harvest_effect +
                       np.random.normal(0, 25, T_ag))

df_ag = pd.DataFrame(prices_ag, columns=ag_commodities)
df_ag['rainfall'] = rainfall
df_ag['month'] = month
df_ag['time'] = np.arange(T_ag)

# Visualize
fig, axes = plt.subplots(3, 1, figsize=(14, 10), sharex=True)

# Prices
for commodity in ag_commodities:
    axes[0].plot(df_ag['time'], df_ag[commodity], label=commodity, linewi
axes[0].set_ylabel('Price (cents/bushel)', fontsize=11)
axes[0].set_title('Agricultural Commodity Prices', fontsize=12, fontweigh
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Rainfall
axes[1].bar(df_ag['time'], df_ag['rainfall'], alpha=0.6, color='blue')
axes[1].axhline(0, color='black', linestyle='--')
axes[1].set_ylabel('Rainfall Index', fontsize=11)
axes[1].set_title('Rainfall (negative = drought)', fontsize=12, fontweigh
axes[1].grid(True, alpha=0.3, axis='y')

# Harvest seasonality
axes[2].bar(df_ag['time'], harvest_effect, alpha=0.6, color='orange')
axes[2].set_xlabel('Time (months)', fontsize=11)
axes[2].set_ylabel('Harvest Effect (cents)', fontsize=11)
axes[2].set_title('Seasonal Harvest Pressure', fontsize=12, fontweight='b
axes[2].grid(True, alpha=0.3, axis='y')

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

In [ ]:

```
# Prepare data for modeling
# Stack prices
prices_stacked = df_ag[ag_commodities].values.flatten()
rainfall_stacked = np.tile(df_ag['rainfall'].values, n_ag)
month_stacked = np.tile(df_ag['month'].values, n_ag)
commodity_idx_ag = np.repeat(np.arange(n_ag), T_ag)

# Build hierarchical model
with pm.Model() as ag_model:
    # Hyperpriors for base prices
    mu_alpha = pm.Normal('mu_alpha', mu=700, sigma=500)
    tau_alpha = pm.HalfNormal('tau_alpha', sigma=500)

    # Hyperpriors for rainfall sensitivity
    mu_beta_rain = pm.Normal('mu_beta_rain', mu=-15, sigma=10)
    tau_beta_rain = pm.HalfNormal('tau_beta_rain', sigma=10)

    # Commodity-specific parameters
    alpha = pm.Normal('alpha', mu=mu_alpha, sigma=tau_alpha, shape=n_ag)
    beta_rain = pm.Normal('beta_rain', mu=mu_beta_rain, sigma=tau_beta_ra

    # Shared harvest seasonality (2 months)
```

```
    harvest_sep = pm.Normal('harvest_sep', mu=-20, sigma=10)  # September
    harvest_oct = pm.Normal('harvest_oct', mu=-20, sigma=10)  # October

    # Map harvest effects to months
    harvest_effects = pm.math.switch(
        pm.math.eq(month_stacked, 8), harvest_sep,
        pm.math.switch(pm.math.eq(month_stacked, 9), harvest_oct, 0)
    )

    # Observation noise
    sigma_y = pm.HalfNormal('sigma_y', sigma=50)

    # Combined mean
    mu = (alpha[commodity_idx_ag] +
          beta_rain[commodity_idx_ag] * rainfall_stacked +
          harvest_effects)

    # Likelihood
    y = pm.Normal('y', mu=mu, sigma=sigma_y, observed=prices_stacked)

    # Sample
    trace_ag = pm.sample(2000, tune=1000, return_inferencedata=True,
                         target_accept=0.95, random_seed=42, chains=2)

print("\nPosterior Summary:")
print(az.summary(trace_ag, var_names=['mu_alpha', 'tau_alpha', 'mu_beta_r
                                      'alpha', 'beta_rain', 'harvest_sep
```

In [ ]:

```
# Extract estimates
alpha_ag_est = trace_ag.posterior['alpha'].mean(dim=['chain', 'draw']).va
beta_rain_est = trace_ag.posterior['beta_rain'].mean(dim=['chain', 'draw'
harvest_sep_est = trace_ag.posterior['harvest_sep'].mean().item()
harvest_oct_est = trace_ag.posterior['harvest_oct'].mean().item()

# Comparison
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# Base prices
x_pos = np.arange(n_ag)
axes[0].bar(x_pos - 0.2, true_alpha_ag, width=0.4, label='True', alpha=0.
axes[0].bar(x_pos + 0.2, alpha_ag_est, width=0.4, label='Estimated', alph
axes[0].set_xticks(x_pos)
axes[0].set_xticklabels(ag_commodities)
axes[0].set_ylabel('Base Price (cents/bushel)', fontsize=11)
axes[0].set_title('Commodity Base Prices (α)', fontsize=12, fontweight='b
axes[0].legend()
axes[0].grid(True, alpha=0.3, axis='y')

# Rainfall sensitivity
axes[1].bar(x_pos - 0.2, true_beta_rain, width=0.4, label='True', alpha=0
axes[1].bar(x_pos + 0.2, beta_rain_est, width=0.4, label='Estimated', alp
axes[1].axhline(0, color='black', linestyle='--')
axes[1].set_xticks(x_pos)
axes[1].set_xticklabels(ag_commodities)
axes[1].set_ylabel('Rainfall Sensitivity (β)', fontsize=11)
axes[1].set_title('Drought Impact (negative rainfall → higher prices)', f
axes[1].legend()
axes[1].grid(True, alpha=0.3, axis='y')

plt.tight_layout()
```

```
plt.show()

# Trading dashboard
print("\n" + "="*70)
print("AGRICULTURAL COMMODITY TRADING DASHBOARD")
print("="*70)

for j, commodity in enumerate(ag_commodities):
    beta_samples = trace_ag.posterior['beta_rain'].values[:, :, j].flatte
    prob_drought_bullish = np.mean(beta_samples < 0)

    print(f"\n{commodity}:")
    print(f"  Base price: {alpha_ag_est[j]:.1f} cents/bushel")
    print(f"  Rainfall β: {beta_rain_est[j]:.2f} (95% CI: [{np.percentile
    print(f"  Prob(drought increases prices): {prob_drought_bullish:.2%}"

    if abs(beta_rain_est[j]) > 15:
        print(f"  → HIGH DROUGHT SENSITIVITY: Weather derivatives highly
    else:
        print(f"  → MODERATE DROUGHT SENSITIVITY")

print(f"\n\nShared Harvest Effects:")
print(f"  September: {harvest_sep_est:.1f} cents (harvest pressure)")
print(f"  October: {harvest_oct_est:.1f} cents (harvest pressure)")
print(f"\n  → Strategy: Short ag futures ahead of harvest, cover in Novem

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

Knowledge Check Quiz

Question 1

What is the main advantage of partial pooling over no pooling?

A) Faster computation
B) Borrows strength across groups, reducing overfitting
C) Assumes all groups are identical
D) Eliminates the need for priors

Answer: B - Partial pooling shares information between related groups while allowing group-spe parameters, reducing overfitting especially with small samples.

Question 2

When does shrinkage have the strongest effect in hierarchical models?

A) Large sample sizes
B) Small sample sizes
C) Equal sample sizes
D) Shrinkage is always the same

Answer: B - Groups with small samples are shrunk more toward the group mean, borrowing str from better-estimated groups.

Question 3

What does a varying slopes model estimate?

A) Only intercepts differ by group
B) Only slopes differ by group
C) Both intercepts and slopes can differ by group
D) All parameters are identical

Answer: C - Varying slopes models allow both intercepts (base levels) and slopes (sensitivities) across groups.

Question 4

Why is modeling cross-commodity correlations important for trading?

A) It makes models more complex
B) It's required by regulators
C) For portfolio construction, hedging, and pairs trading
D) It eliminates all risk

Answer: C - Correlations determine diversification benefits, hedging effectiveness, and spread t opportunities.

Question 5

In the agricultural model, what does a negative β (rainfall) coefficient indicate?

A) More rainfall increases prices
B) Drought (low rainfall) increases prices
C) Rainfall has no effect
D) The model is wrong

Answer: B - Negative coefficient means low rainfall (drought) is associated with higher prices du reduced yields.

Exercises

Exercise 1 : Energy Spread Trading

Using the WTI-Brent correlation model:

1. Define the spread as: Brent - WTI
2. Model the spread as a mean-reverting process
3. Calculate the equilibrium spread (mean)
4. Design a pairs trading rule: long spread when 2 σ below mean, short when 2 σ above

5. Backtest on the generated data

Exercise 2: Portfolio Optimization

Given estimated correlations for energy commodities:

1. Assume equal expected returns
2. Use correlation matrix to compute minimum variance portfolio weights
3. Compare diversified portfolio volatility to single-commodity volatility
4. Calculate diversification benefit

Exercise 3: Drought Scenario Analysis

For agricultural commodities:

1. Simulate a severe drought: rainfall = -2 (2 std below normal) for 6 months
2. Forecast price impact for each commodity
3. Which commodity benefits most from drought?
4. Calculate portfolio P&L if long all three commodities equally

Exercise 4: Hierarchical Volatility Model

Build a model where volatility (σ) varies by commodity: $$ \sigma_j \sim \text{HalfNormal}(\mu_\
\tau_\sigma) $$

1. Estimate commodity-specific volatilities
2. Rank commodities by volatility
3. Design position sizing rule: allocate inversely to volatility
4. Compare to equal-weighted portfolio

Exercise 5: Forecasting with Missing Data

Simulate missing data scenario:

1. Remove last 30 days of Soybeans prices
2. Re-estimate hierarchical model using only Corn and Wheat
3. Use group-level parameters to forecast Soybeans
4. Compare forecast accuracy to model fit on complete data
5. Quantify information borrowing benefit

Summary

In this module, you learned:

1. **Pooling Strategies**: Complete, none, and partial pooling for multi-group data
2. **Shrinkage**: How hierarchical models regularize estimates toward group means
3. **Varying Intercepts & Slopes**: Commodity-specific parameters with shared priors
4. **Cross-Commodity Correlations**: Modeling dependencies for portfolio decisions

5. **Agricultural Application**: Weather impacts across related commodities

Key Takeaways for Trading

- **Information Borrowing**: Improve estimates for thinly-traded commodities
- **Regularization**: Prevent overfitting via shrinkage to group mean
- **Pairs Trading**: Identify relative mispricings within commodity complexes
- **Risk Management**: Model correlations for optimal portfolio construction
- **Cross-Sectional Analysis**: Which commodity is most/least sensitive to shared drivers?
- **Scenario Analysis**: Forecast impact of shocks (drought, OPEC cuts) across complex

Hierarchical models are essential when:

- You have **multiple related assets** (energy complex, grains, metals)
- **Sample sizes vary** (some commodities thinly traded)
- You want to **share information** without assuming homogeneity
- **Correlations matter** for hedging and diversification

This framework scales from 3 commodities (as shown) to dozens, enabling systematic cro sectional strategies.

Course Summary & Next Steps

What You've Accomplished

Across Modules 1 - 7, you've built a complete Bayesian toolkit for commodity trading:

1. **Foundations**: Bayesian inference, prior/posterior, MCMC
2. **Prior Selection**: Informative, weakly informative, and regularizing priors
3. **MCMC Inference**: Sampling, convergence diagnostics, trace analysis
4. **Time Series**: Autocorrelation, stationarity, seasonality
5. **Bayesian Linear Regression**: Uncertainty quantification, robust models
6. **BSTS**: Trend, seasonality, dynamic regression, state-space models
7. **Hierarchical Models**: Multi-commodity analysis, shrinkage, correlations

Remaining Modules (8 - 10)

- **Module 8: Gaussian Processes** - Non-parametric flexible curves for price discovery
- **Module 9: Volatility Modeling** - GARCH, stochastic volatility, VIX forecasting
- **Module 10: Trading Strategies** - Complete systematic strategies with backtesting

Practice Recommendations

1. **Real data**: Apply to actual commodity prices (FRED, Quandl, Yahoo Finance)
2. **Daily practice**: Build one small model per day
3. **Paper trading**: Test forecasts in real-time before risking capital
4. **Community**: Share models, get feedback, iterate

Resources

- PyMC Documentation: https://www.pymc.io/
- ArviZ Gallery: https://arviz-devs.github.io/arviz/
- Statistical Rethinking (McElreath): Excellent Bayesian textbook
- BDA 3 (Gelman et al.): Bayesian Data Analysis reference

Congratulations on completing Module 7 ! You now have the statistical tools to build sophisticated multi-commodity trading models. Keep practicing, and see you in Module