Module 5 : Bayesian Linear Regression for Price Prediction

Learning Objectives

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

1. Build simple and multiple Bayesian linear regression models using PyMC
2. Interpret posterior distributions for trading decisions
3. Distinguish between credible intervals and confidence intervals
4. Compare models using WAIC and LOO cross-validation
5. Implement robust regression with Student-t likelihood
6. Apply Bayesian linear regression to natural gas price forecasting with weather data

Why This Matters for Trading

Bayesian linear regression provides several advantages over classical approaches for commodi trading:

- **Uncertainty Quantification**: Full posterior distributions allow you to quantify prediction unc critical for risk management
- **Incorporating Prior Knowledge**: Include market beliefs (e.g., "oil prices tend to mean-reve directly in the model
- **Credible Intervals**: Unlike frequentist confidence intervals, Bayesian credible intervals have probability interpretations
- **Model Comparison**: WAIC/LOO provide principled ways to select between competing mod
- **Robust to Outliers**: Student-t likelihood handles price spikes and crashes better than norm likelihood
- **Sequential Updating**: As new data arrives, update beliefs without re-estimating from scratc

In commodity markets, where supply shocks, weather events, and geopolitical risks create extre volatility, these features translate directly to better risk-adjusted returns.

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 . Simple Bayesian Linear Regression: Price ~ Time

Theory

A simple linear regression model relates a response variable y (e.g., commodity price) to a sir predictor x (e.g., time):

$$ \begin{align} y_i &\sim \text{Normal}(\mu_i, \sigma) \\ \mu_i &= \alpha + \beta x_i \\ \alpha &\s \text{Normal}(\mu_\alpha, \sigma_\alpha) \\ \beta &\sim \text{Normal}(\mu_\beta, \sigma_\beta) \' &\sim \text{HalfNormal}(\sigma_s) \end{align} $$

Where:

- α is the intercept (baseline price)
- β is the slope (trend)
- σ is the observation noise (volatility)

Trading Interpretation

- **$\beta >** 0 **$**: Upward trend (bullish signal)
- **$\beta <** 0 **$**: Downward trend (bearish signal)
- **Wide posterior for β**: High uncertainty about trend direction
- **Large σ**: High volatility, wider position sizing needed

In [ ]:

```
# Generate synthetic crude oil price data with trend
np.random.seed(42)
n_days = 120
time = np.arange(n_days)

# True parameters
true_alpha = 65.0  # Starting price
true_beta = 0.15   # Upward trend of $0.15/day
true_sigma = 3.5   # Daily volatility

# Generate prices
crude_prices = true_alpha + true_beta * time + np.random.normal(0, true_s

# Create DataFrame
df_simple = pd.DataFrame({
    'day': time,
    'price': crude_prices
})

# Visualize
fig, ax = plt.subplots(figsize=(12, 5))
ax.plot(df_simple['day'], df_simple['price'], 'o-', alpha=0.6, label='Obs
ax.axhline(true_alpha, color='red', linestyle='--', alpha=0.5, label=f'Tr
ax.plot(time, true_alpha + true_beta * time, 'g--', linewidth=2, label=f'
ax.set_xlabel('Days', fontsize=12)
ax.set_ylabel('Crude Oil Price ($/barrel)', fontsize=12)
ax.set_title('Crude Oil Prices: Simple Linear Trend', fontsize=14, fontwe
```

```python
ax.legend()
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

print(f"Data summary:")
print(df_simple.describe())
```

In [ ]:
```python
# Build Bayesian linear regression model
with pm.Model() as model_simple:
    # Data
    x = pm.Data('x', df_simple['day'].values)
    y_obs = pm.Data('y_obs', df_simple['price'].values)

    # Priors
    # Weakly informative prior: we expect crude oil around $60-$80
    alpha = pm.Normal('alpha', mu=70, sigma=20)

    # Prior on trend: we don't expect huge daily changes
    # Could be negative (bear market) or positive (bull market)
    beta = pm.Normal('beta', mu=0, sigma=1)

    # Prior on volatility: crude oil typically has $2-$10 daily volatilit
    sigma = pm.HalfNormal('sigma', sigma=10)

    # Linear model
    mu = alpha + beta * x

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

    # Sample from posterior
    trace_simple = pm.sample(2000, tune=1000, return_inferencedata=True,

# Summary
print("\nPosterior Summary:")
print(az.summary(trace_simple, var_names=['alpha', 'beta', 'sigma']))
```

In [ ]:
```python
# Visualize posterior distributions
fig, axes = plt.subplots(1, 3, figsize=(15, 4))

# Alpha (intercept)
az.plot_posterior(trace_simple, var_names=['alpha'], ax=axes[0],
                  ref_val=true_alpha, color='steelblue')
axes[0].set_title('Posterior: Intercept (α)', fontsize=12, fontweight='bo
axes[0].axvline(true_alpha, color='red', linestyle='--', linewidth=2, lab

# Beta (slope)
az.plot_posterior(trace_simple, var_names=['beta'], ax=axes[1],
                  ref_val=true_beta, color='darkorange')
axes[1].set_title('Posterior: Slope (β)', fontsize=12, fontweight='bold')
axes[1].axvline(true_beta, color='red', linestyle='--', linewidth=2, labe
axes[1].axvline(0, color='black', linestyle=':', alpha=0.5, label='Zero (

# Sigma (noise)
az.plot_posterior(trace_simple, var_names=['sigma'], ax=axes[2],
                  ref_val=true_sigma, color='forestgreen')
axes[2].set_title('Posterior: Volatility (σ)', fontsize=12, fontweight='b
axes[2].axvline(true_sigma, color='red', linestyle='--', linewidth=2, lab
```

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

# Trading interpretation
beta_samples = trace_simple.posterior['beta'].values.flatten()
prob_uptrend = np.mean(beta_samples > 0)
print(f"\nTrading Signal:")
print(f"Probability of upward trend: {prob_uptrend:.2%}")
print(f"Expected daily price change: ${np.mean(beta_samples):.3f} ± ${np.
if prob_uptrend > 0.75:
    print("→ BULLISH: Strong evidence of uptrend")
elif prob_uptrend < 0.25:
    print("→ BEARISH: Strong evidence of downtrend")
else:
    print("→ NEUTRAL: Insufficient evidence for directional trade")
```

2 . Posterior Predictive Checks

Before using the model for trading, we must verify it captures the data-generating process.

In [ ]:
```
# Generate posterior predictive samples
with model_simple:
    ppc_simple = pm.sample_posterior_predictive(trace_simple, random_seed

# Visualize
fig, ax = plt.subplots(figsize=(12, 5))

# Plot 100 posterior predictive samples
ppc_samples = ppc_simple.posterior_predictive['y'].values
for i in range(100):
    chain_idx = np.random.randint(0, ppc_samples.shape[0])
    draw_idx = np.random.randint(0, ppc_samples.shape[1])
    ax.plot(df_simple['day'], ppc_samples[chain_idx, draw_idx, :],
            alpha=0.05, color='blue', linewidth=1)

# Plot observed data
ax.plot(df_simple['day'], df_simple['price'], 'ko', markersize=4, label='

ax.set_xlabel('Days', fontsize=12)
ax.set_ylabel('Price ($/barrel)', fontsize=12)
ax.set_title('Posterior Predictive Check: Model Fit', fontsize=14, fontwe
ax.legend()
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

print("If observed data (black dots) falls within the blue cloud, model c
```

3 . Credible Intervals vs Confidence Intervals

Key Difference

Bayesian Credible Interval (CI):

- "There is a 95 % probability that the true parameter lies in this interval"
- Direct probability statement about parameter

- What traders actually want to know

Frequentist Confidence Interval:

- "If we repeated the experiment many times, 95% of such intervals would contain the tr parameter"
- Statement about the procedure, not the parameter
- Difficult to interpret for one-time decisions

Trading Application

For position sizing, you want to know: "What's the probability my forecast is within X% of reality? Bayesian credible intervals answer this directly.

In [ ]:

```
# Forecast 30 days ahead with credible intervals
future_days = np.arange(n_days, n_days + 30)

# Update model data for predictions
with model_simple:
    pm.set_data({'x': future_days})
    forecast_simple = pm.sample_posterior_predictive(trace_simple, random

# Extract predictions
forecast_samples = forecast_simple.posterior_predictive['y'].values.resha
forecast_mean = forecast_samples.mean(axis=0)
forecast_lower = np.percentile(forecast_samples, 2.5, axis=0)
forecast_upper = np.percentile(forecast_samples, 97.5, axis=0)

# Visualize forecast
fig, ax = plt.subplots(figsize=(14, 6))

# Historical data
ax.plot(df_simple['day'], df_simple['price'], 'o-', color='black',
        label='Historical Prices', markersize=3, alpha=0.7)

# Forecast
ax.plot(future_days, forecast_mean, 'o-', color='red',
        label='Forecast (Mean)', markersize=4, linewidth=2)
ax.fill_between(future_days, forecast_lower, forecast_upper,
                alpha=0.3, color='red', label='95% Credible Interval')

ax.axvline(n_days - 0.5, color='gray', linestyle='--', linewidth=2, label
ax.set_xlabel('Days', fontsize=12)
ax.set_ylabel('Price ($/barrel)', fontsize=12)
ax.set_title('30-Day Price Forecast with 95% Credible Interval', fontsize
ax.legend(loc='upper left')
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# Trading interpretation
print(f"\nForecast for Day {n_days + 30}:")
print(f"Expected price: ${forecast_mean[-1]:.2f}")
print(f"95% Credible Interval: [${forecast_lower[-1]:.2f}, ${forecast_upp
print(f"\nInterpretation: There is a 95% probability the price will be be
print(f"Uncertainty range: ${forecast_upper[-1] - forecast_lower[-1]:.2f}
```

4 . Multiple Regression: Price ~ Supply + Demand + US
Index

Theory

Multiple regression extends to multiple predictors:

$$ \begin{align} y_i &\sim \text{Normal}(\mu_i, \sigma) \\ \mu_i &= \alpha + \beta_ 1 x_{ 1 i} + \beta_ 2 x_{ 2 i} + \beta_ 3 x_{ 3 i} \\ \alpha &\sim \text{Normal}(\mu_\alpha, \sigma_\alpha \beta_j &\sim \text{Normal}(0 , \sigma_\beta) \quad \text{for } j = 1 , 2 , 3 \\ \sigma & \text{HalfNormal}(\sigma_s) \end{align} $$

Trading Application

For crude oil:

- $x_ 1 $: Global supply (million barrels/day) → $\beta_ 1 < 0 $ (more supply, lower pri
- $x_ 2 $: Global demand (million barrels/day) → $\beta_ 2 > 0 $ (more demand, highe
- $x_ 3 $: USD index → $\beta_ 3 < 0 $ (stronger dollar, lower commodity prices)

In [ ]:
```python
# Generate synthetic multiple regression data
np.random.seed(42)
n = 150

# Predictors
supply = np.random.normal(100, 5, n)  # Million barrels/day
demand = np.random.normal(98, 4, n)   # Million barrels/day
usd_index = np.random.normal(95, 3, n)  # USD strength index

# True coefficients
true_alpha_multi = 200.0
true_beta_supply = -1.2   # Negative: more supply → lower price
true_beta_demand = 1.8    # Positive: more demand → higher price
true_beta_usd = -0.5      # Negative: stronger USD → lower price
true_sigma_multi = 4.0

# Generate prices
price_multi = (true_alpha_multi +
               true_beta_supply * supply +
               true_beta_demand * demand +
               true_beta_usd * usd_index +
               np.random.normal(0, true_sigma_multi, n))

# Create DataFrame
df_multi = pd.DataFrame({
    'price': price_multi,
    'supply': supply,
    'demand': demand,
    'usd_index': usd_index
})

print("Data Summary:")
print(df_multi.describe())
```

```python
print(f"\nCorrelation with price:")
print(df_multi.corr()['price'].sort_values(ascending=False))
```

In [ ]:

```python
# Standardize predictors for better sampling
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_multi[['supply', 'demand', 'usd_index'

# Build multiple regression model
with pm.Model() as model_multi:
    # Data
    X = pm.Data('X', X_scaled)
    y_obs = pm.Data('y_obs', df_multi['price'].values)

    # Priors
    alpha = pm.Normal('alpha', mu=70, sigma=20)

    # Coefficients: using regularizing priors
    beta_supply = pm.Normal('beta_supply', mu=0, sigma=10)
    beta_demand = pm.Normal('beta_demand', mu=0, sigma=10)
    beta_usd = pm.Normal('beta_usd', mu=0, sigma=10)

    # Stack coefficients
    beta = pm.math.stack([beta_supply, beta_demand, beta_usd])

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

    # Linear model
    mu = alpha + pm.math.dot(X, beta)

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

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

# Summary
print("\nPosterior Summary:")
print(az.summary(trace_multi, var_names=['alpha', 'beta_supply', 'beta_de
```

In [ ]:

```python
# Visualize coefficient posteriors
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Supply coefficient
az.plot_posterior(trace_multi, var_names=['beta_supply'], ax=axes[0, 0],
axes[0, 0].set_title('β (Supply): Standardized Effect', fontsize=12, font
axes[0, 0].axvline(0, color='black', linestyle='--', alpha=0.5)

# Demand coefficient
az.plot_posterior(trace_multi, var_names=['beta_demand'], ax=axes[0, 1],
axes[0, 1].set_title('β (Demand): Standardized Effect', fontsize=12, font
axes[0, 1].axvline(0, color='black', linestyle='--', alpha=0.5)

# USD Index coefficient
az.plot_posterior(trace_multi, var_names=['beta_usd'], ax=axes[1, 0], col
axes[1, 0].set_title('β (USD Index): Standardized Effect', fontsize=12, f
axes[1, 0].axvline(0, color='black', linestyle='--', alpha=0.5)
```

```
# Sigma
az.plot_posterior(trace_multi, var_names=['sigma'], ax=axes[1, 1], color=
axes[1, 1].set_title('σ (Residual Volatility)', fontsize=12, fontweight='

plt.tight_layout()
plt.show()

# Trading interpretation
beta_supply_samples = trace_multi.posterior['beta_supply'].values.flatten
beta_demand_samples = trace_multi.posterior['beta_demand'].values.flatten
beta_usd_samples = trace_multi.posterior['beta_usd'].values.flatten()

print("\nTrading Signals:")
print(f"Supply coefficient: {np.mean(beta_supply_samples):.3f} (95% CI: [
print(f"  → Prob(β < 0): {np.mean(beta_supply_samples < 0):.2%} - {'CONFI

print(f"\nDemand coefficient: {np.mean(beta_demand_samples):.3f} (95% CI:
print(f"  → Prob(β > 0): {np.mean(beta_demand_samples > 0):.2%} - {'CONFI

print(f"\nUSD Index coefficient: {np.mean(beta_usd_samples):.3f} (95% CI:
print(f"  → Prob(β < 0): {np.mean(beta_usd_samples < 0):.2%} - {'CONFIRME
```

5 . Model Comparison: WAIC and LOO

Theory

WAIC (Watanabe-Akaike Information Criterion):

- Bayesian generalization of AIC
- Estimates out-of-sample predictive accuracy
- Lower is better
- Accounts for both fit and complexity

LOO (Leave-One-Out Cross-Validation):

- Estimates predictive accuracy by leaving each observation out
- Approximated efficiently using Pareto-smoothed importance sampling
- More robust than WAIC for small samples

Trading Application

Compare simple trend model vs multiple regression model to decide which to use for trading.

In [ ]:
```
# Build comparable simple model on same data
with pm.Model() as model_simple_multi:
    # Use only time as predictor
    x = pm.Data('x', np.arange(len(df_multi)))
    y_obs = pm.Data('y_obs', df_multi['price'].values)

    alpha = pm.Normal('alpha', mu=70, sigma=20)
    beta = pm.Normal('beta', mu=0, sigma=1)
    sigma = pm.HalfNormal('sigma', sigma=10)

    mu = alpha + beta * x
    y = pm.Normal('y', mu=mu, sigma=sigma, observed=y_obs)
```

```
    trace_simple_multi = pm.sample(2000, tune=1000, return_inferencedata=

# Compute model comparison metrics
comparison = az.compare({
    'Simple (Time Only)': trace_simple_multi,
    'Multiple Regression': trace_multi
})

print("\nModel Comparison (LOO):")
print(comparison)
print("\nInterpretation:")
print("- 'elpd_loo': Expected log pointwise predictive density (higher is
print("- 'p_loo': Effective number of parameters")
print("- 'loo': LOO information criterion (lower is better)")
print("- 'se': Standard error of the difference")
print("- 'dse': Standard error of the LOO difference")
print("- 'weight': Pseudo-BMA weight (model probability)")
print(f"\nBest model: {comparison.index[0]}")
print(f"Weight: {comparison.loc[comparison.index[0], 'weight']:.2%}")
```

In [ ]:
```
# Visualize model comparison
az.plot_compare(comparison, insample_dev=False)
plt.title('Model Comparison: LOO Cross-Validation', fontsize=14, fontweig
plt.tight_layout()
plt.show()

print("\nTrading Decision:")
if comparison.loc['Multiple Regression', 'weight'] > 0.75:
    print("→ USE MULTIPLE REGRESSION: Strong evidence it outperforms simp
    print("  Action: Incorporate supply, demand, and USD data into tradin
elif comparison.loc['Simple (Time Only)', 'weight'] > 0.75:
    print("→ USE SIMPLE TREND: Sufficient for prediction, avoid overfitti
    print("  Action: Stick with simple time-based model")
else:
    print("→ MODEL AVERAGING: Combine predictions from both models")
    print("  Action: Weight predictions by model probabilities")
```

6 . Robust Regression with Student-t Likelihood

Theory

Normal likelihood assumes Gaussian noise, which underestimates tail risk. Student-t likelihood h heavier tails:

$$ \begin{align} y_i &\sim \text{StudentT}(\nu, \mu_i, \sigma) \\ \mu_i &= \alpha + \beta x_i \\ \nu \text{Gamma}(2 , 0 . 1) \end{align} $$

Where ν is the degrees of freedom:

- $\nu = 1 $: Cauchy distribution (very heavy tails)
- $\nu = 3 0 +$: Approximately Normal
- $\nu \in [3 , 1 0]$: Typical for commodity data

Trading Application

Commodity markets experience:

- Supply shocks (hurricanes, OPEC cuts)
- Demand spikes (cold snaps, geopolitical events)
- "Flash crashes" and fat-tail events

Student-t likelihood prevents a few extreme days from distorting the entire model.

In [ ]:
```python
# Generate data with outliers (simulating supply shocks)
np.random.seed(42)
n_robust = 100
time_robust = np.arange(n_robust)

# Base trend
price_robust = 60 + 0.1 * time_robust + np.random.normal(0, 2, n_robust)

# Add 5 extreme shocks (fat tails)
shock_days = [20, 35, 50, 68, 85]
for day in shock_days:
    price_robust[day] += np.random.choice([-15, -12, 12, 15])  # Large pr

df_robust = pd.DataFrame({
    'day': time_robust,
    'price': price_robust
})

# Visualize
fig, ax = plt.subplots(figsize=(12, 5))
ax.plot(df_robust['day'], df_robust['price'], 'o-', alpha=0.7)
ax.scatter([shock_days], df_robust.loc[shock_days, 'price'],
           color='red', s=100, zorder=5, label='Supply Shocks', edgecolor
ax.set_xlabel('Days', fontsize=12)
ax.set_ylabel('Price ($/barrel)', fontsize=12)
ax.set_title('Crude Oil Prices with Supply Shocks (Fat Tails)', fontsize=
ax.legend()
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

In [ ]:
```python
# Normal likelihood model (non-robust)
with pm.Model() as model_normal:
    x = pm.Data('x', df_robust['day'].values)
    y_obs = pm.Data('y_obs', df_robust['price'].values)

    alpha = pm.Normal('alpha', mu=60, sigma=10)
    beta = pm.Normal('beta', mu=0, sigma=1)
    sigma = pm.HalfNormal('sigma', sigma=10)

    mu = alpha + beta * x
    y = pm.Normal('y', mu=mu, sigma=sigma, observed=y_obs)

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

# Student-t likelihood model (robust)
```

```python
with pm.Model() as model_robust:
    x = pm.Data('x', df_robust['day'].values)
    y_obs = pm.Data('y_obs', df_robust['price'].values)

    alpha = pm.Normal('alpha', mu=60, sigma=10)
    beta = pm.Normal('beta', mu=0, sigma=1)
    sigma = pm.HalfNormal('sigma', sigma=10)

    # Degrees of freedom: lower = heavier tails
    nu = pm.Gamma('nu', alpha=2, beta=0.1)

    mu = alpha + beta * x
    y = pm.StudentT('y', nu=nu, mu=mu, sigma=sigma, observed=y_obs)

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

print("\nNormal Likelihood Model:")
print(az.summary(trace_normal, var_names=['alpha', 'beta', 'sigma']))

print("\nStudent-t Likelihood Model:")
print(az.summary(trace_robust, var_names=['alpha', 'beta', 'sigma', 'nu']
```

In [ ]:

```python
# Compare fits
fig, axes = plt.subplots(1, 2, figsize=(16, 5))

# Normal model
alpha_normal = trace_normal.posterior['alpha'].values.flatten().mean()
beta_normal = trace_normal.posterior['beta'].values.flatten().mean()
axes[0].plot(df_robust['day'], df_robust['price'], 'o', alpha=0.5, label=
axes[0].scatter([shock_days], df_robust.loc[shock_days, 'price'],
                color='red', s=100, zorder=5, edgecolors='black', linewid
axes[0].plot(df_robust['day'], alpha_normal + beta_normal * df_robust['da
             'b-', linewidth=2, label='Normal Fit')
axes[0].set_title('Normal Likelihood (Non-Robust)', fontsize=12, fontweig
axes[0].set_xlabel('Days')
axes[0].set_ylabel('Price ($/barrel)')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Robust model
alpha_robust = trace_robust.posterior['alpha'].values.flatten().mean()
beta_robust = trace_robust.posterior['beta'].values.flatten().mean()
axes[1].plot(df_robust['day'], df_robust['price'], 'o', alpha=0.5, label=
axes[1].scatter([shock_days], df_robust.loc[shock_days, 'price'],
                color='red', s=100, zorder=5, label='Supply Shocks', edge
axes[1].plot(df_robust['day'], alpha_robust + beta_robust * df_robust['da
             'g-', linewidth=2, label='Student-t Fit')
axes[1].set_title('Student-t Likelihood (Robust)', fontsize=12, fontweigh
axes[1].set_xlabel('Days')
axes[1].set_ylabel('Price ($/barrel)')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print("\nComparison:")
print(f"Normal model slope: {beta_normal:.4f}")
print(f"Student-t model slope: {beta_robust:.4f}")
print(f"\nThe Student-t model is less influenced by extreme shocks.")
```

```
print(f"Estimated degrees of freedom: {trace_robust.posterior['nu'].value
print(f"(Lower ν = heavier tails; ν ≈ 5-10 is typical for commodities)")
```

In [ ]:
```
# Model comparison
comparison_robust = az.compare({
    'Normal Likelihood': trace_normal,
    'Student-t Likelihood': trace_robust
})

print("\nModel Comparison:")
print(comparison_robust)
print(f"\nBest model: {comparison_robust.index[0]}")
print(f"\nTrading Recommendation: Use Student-t likelihood for commodity
```

7 . Practical Example: Natural Gas Prices with Weathe Data

Natural gas prices are highly sensitive to weather:

- **Winter (heating demand)**: Cold temperatures → Higher prices
- **Summer (cooling demand)**: Hot temperatures → Higher prices
- **Storage levels**: Lower inventory → Higher prices

We'll build a Bayesian regression model incorporating these factors.

In [ ]:
```
# Generate synthetic natural gas data
np.random.seed(42)
n_gas = 365  # 1 year of daily data

# Day of year (for seasonality)
day_of_year = np.arange(1, n_gas + 1)

# Temperature (Fahrenheit): cold in winter, hot in summer
# Sinusoidal pattern
temperature = 50 + 30 * np.sin(2 * np.pi * (day_of_year - 80) / 365) + np

# Heating Degree Days (HDD): higher when cold
hdd = np.maximum(65 - temperature, 0)

# Cooling Degree Days (CDD): higher when hot
cdd = np.maximum(temperature - 65, 0)

# Storage levels (billion cubic feet): starts high, depletes in winter
storage = 3500 - 1000 * np.sin(2 * np.pi * (day_of_year - 80) / 365) + np

# Price model:
# Base price + HDD effect (heating) + CDD effect (cooling) - storage effe
true_base = 3.0
true_hdd_coef = 0.05    # Higher heating demand → Higher price
true_cdd_coef = 0.03    # Higher cooling demand → Higher price
true_storage_coef = -0.0008  # More storage → Lower price

natgas_price = (true_base +
                true_hdd_coef * hdd +
                true_cdd_coef * cdd +
                true_storage_coef * storage +
                np.random.normal(0, 0.3, n_gas))
```

```
# Ensure prices are positive
natgas_price = np.maximum(natgas_price, 1.5)

df_natgas = pd.DataFrame({
    'day': day_of_year,
    'temperature': temperature,
    'hdd': hdd,
    'cdd': cdd,
    'storage': storage,
    'price': natgas_price
})

print("Natural Gas Data Summary:")
print(df_natgas.describe())
```

In [ ]:
```
# Visualize data
fig, axes = plt.subplots(3, 1, figsize=(14, 10), sharex=True)

# Price
axes[0].plot(df_natgas['day'], df_natgas['price'], linewidth=1.5, color='
axes[0].set_ylabel('Price ($/MMBtu)', fontsize=11)
axes[0].set_title('Natural Gas Prices (Daily)', fontsize=12, fontweight='
axes[0].grid(True, alpha=0.3)

# Temperature & Degree Days
ax2 = axes[1]
ax2.plot(df_natgas['day'], df_natgas['temperature'], label='Temperature (
ax2.axhline(65, color='black', linestyle='--', alpha=0.5, label='Base Tem
ax2.set_ylabel('Temperature (°F)', fontsize=11)
ax2.legend(loc='upper left')
ax2.grid(True, alpha=0.3)

ax2b = ax2.twinx()
ax2b.fill_between(df_natgas['day'], 0, df_natgas['hdd'], alpha=0.3, color
ax2b.fill_between(df_natgas['day'], 0, df_natgas['cdd'], alpha=0.3, color
ax2b.set_ylabel('Degree Days', fontsize=11)
ax2b.legend(loc='upper right')

# Storage
axes[2].plot(df_natgas['day'], df_natgas['storage'], linewidth=1.5, color
axes[2].set_xlabel('Day of Year', fontsize=11)
axes[2].set_ylabel('Storage (Bcf)', fontsize=11)
axes[2].set_title('Natural Gas Storage Levels', fontsize=12, fontweight='
axes[2].grid(True, alpha=0.3)

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

In [ ]:
```
# Standardize predictors
scaler_natgas = StandardScaler()
X_natgas_scaled = scaler_natgas.fit_transform(df_natgas[['hdd', 'cdd', 's

# Build Bayesian regression model
with pm.Model() as model_natgas:
    # Data
    X = pm.Data('X', X_natgas_scaled)
    y_obs = pm.Data('y_obs', df_natgas['price'].values)
```

```
    # Priors
    alpha = pm.Normal('alpha', mu=3, sigma=2)  # Base price around $3/MMB

    # Coefficients
    beta_hdd = pm.Normal('beta_hdd', mu=0, sigma=1)  # Heating effect
    beta_cdd = pm.Normal('beta_cdd', mu=0, sigma=1)  # Cooling effect
    beta_storage = pm.Normal('beta_storage', mu=0, sigma=1)  # Storage ef

    beta = pm.math.stack([beta_hdd, beta_cdd, beta_storage])

    # Use Student-t for robustness
    nu = pm.Gamma('nu', alpha=2, beta=0.1)
    sigma = pm.HalfNormal('sigma', sigma=2)

    # Linear model
    mu = alpha + pm.math.dot(X, beta)

    # Likelihood
    y = pm.StudentT('y', nu=nu, mu=mu, sigma=sigma, observed=y_obs)

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

print("\nPosterior Summary:")
print(az.summary(trace_natgas, var_names=['alpha', 'beta_hdd', 'beta_cdd'
```

In [ ]:

```
# Visualize posteriors
fig, axes = plt.subplots(2, 3, figsize=(16, 8))

az.plot_posterior(trace_natgas, var_names=['alpha'], ax=axes[0, 0], color
axes[0, 0].set_title('Base Price (α)', fontsize=11, fontweight='bold')

az.plot_posterior(trace_natgas, var_names=['beta_hdd'], ax=axes[0, 1], co
axes[0, 1].set_title('HDD Effect (β_hdd)', fontsize=11, fontweight='bold'
axes[0, 1].axvline(0, color='black', linestyle='--', alpha=0.5)

az.plot_posterior(trace_natgas, var_names=['beta_cdd'], ax=axes[0, 2], co
axes[0, 2].set_title('CDD Effect (β_cdd)', fontsize=11, fontweight='bold'
axes[0, 2].axvline(0, color='black', linestyle='--', alpha=0.5)

az.plot_posterior(trace_natgas, var_names=['beta_storage'], ax=axes[1, 0]
axes[1, 0].set_title('Storage Effect (β_storage)', fontsize=11, fontweigh
axes[1, 0].axvline(0, color='black', linestyle='--', alpha=0.5)

az.plot_posterior(trace_natgas, var_names=['sigma'], ax=axes[1, 1], color
axes[1, 1].set_title('Volatility (σ)', fontsize=11, fontweight='bold')

az.plot_posterior(trace_natgas, var_names=['nu'], ax=axes[1, 2], color='p
axes[1, 2].set_title('Degrees of Freedom (ν)', fontsize=11, fontweight='b

plt.tight_layout()
plt.show()

# Trading interpretation
beta_hdd_samples = trace_natgas.posterior['beta_hdd'].values.flatten()
beta_cdd_samples = trace_natgas.posterior['beta_cdd'].values.flatten()
beta_storage_samples = trace_natgas.posterior['beta_storage'].values.flat

print("\n=== TRADING SIGNALS ===")
print(f"\nHeating Demand (HDD):")
```

```python
print(f"  Effect: {np.mean(beta_hdd_samples):.4f} (95% CI: [{np.percentil
print(f"  Prob(β > 0): {np.mean(beta_hdd_samples > 0):.2%}")
if np.mean(beta_hdd_samples > 0) > 0.95:
    print("  → CONFIRMED: Cold weather increases prices (go long ahead of

print(f"\nCooling Demand (CDD):")
print(f"  Effect: {np.mean(beta_cdd_samples):.4f} (95% CI: [{np.percentil
print(f"  Prob(β > 0): {np.mean(beta_cdd_samples > 0):.2%}")
if np.mean(beta_cdd_samples > 0) > 0.95:
    print("  → CONFIRMED: Hot weather increases prices (go long ahead of

print(f"\nStorage Levels:")
print(f"  Effect: {np.mean(beta_storage_samples):.4f} (95% CI: [{np.perce
print(f"  Prob(β < 0): {np.mean(beta_storage_samples < 0):.2%}")
if np.mean(beta_storage_samples < 0) > 0.95:
    print("  → CONFIRMED: Low storage increases prices (monitor weekly EI
```

In [ ]:

```python
# In-sample fit
with model_natgas:
    ppc_natgas = pm.sample_posterior_predictive(trace_natgas, random_seed

ppc_mean = ppc_natgas.posterior_predictive['y'].mean(dim=['chain', 'draw'

fig, axes = plt.subplots(2, 1, figsize=(14, 8), sharex=True)

# Actual vs Fitted
axes[0].plot(df_natgas['day'], df_natgas['price'], 'o', alpha=0.5, label=
axes[0].plot(df_natgas['day'], ppc_mean, linewidth=2, color='red', label=
axes[0].set_ylabel('Price ($/MMBtu)', fontsize=11)
axes[0].set_title('Natural Gas: Actual vs Fitted Prices', fontsize=12, fo
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Residuals
residuals = df_natgas['price'].values - ppc_mean
axes[1].plot(df_natgas['day'], residuals, 'o', alpha=0.6, markersize=3)
axes[1].axhline(0, color='black', linestyle='--', linewidth=1.5)
axes[1].fill_between(df_natgas['day'], -2*residuals.std(), 2*residuals.st
                     alpha=0.2, color='gray', label='±2σ')
axes[1].set_xlabel('Day of Year', fontsize=11)
axes[1].set_ylabel('Residuals', fontsize=11)
axes[1].set_title('Residuals (Actual - Fitted)', fontsize=12, fontweight=
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

print(f"\nModel Performance:")
print(f"RMSE: ${np.sqrt(np.mean(residuals**2)):.4f}/MMBtu")
print(f"MAE: ${np.mean(np.abs(residuals)):.4f}/MMBtu")
print(f"R²: {1 - np.var(residuals) / np.var(df_natgas['price']):.4f}")
```

Knowledge Check Quiz

Test your understanding with these questions:

Question 1

What is the key difference between a Bayesian credible interval and a frequentist confidence int

A) Credible intervals are always wider
B) Credible intervals allow direct probability statements about parameters
C) Confidence intervals are more accurate
D) They are mathematically equivalent

Answer: B - Credible intervals provide the probability that the parameter lies in the interval, give data. Confidence intervals describe the long-run frequency of interval coverage.

Question 2

When should you use Student-t likelihood instead of Normal likelihood?

A) When you have large sample sizes
B) When data has heavy tails or outliers
C) When you want faster computation
D) Never, Normal is always better

Answer: B - Student-t likelihood is robust to outliers and heavy tails, common in commodity mar supply shocks.

Question 3

What does WAIC measure?

A) Model training accuracy
B) Out-of-sample predictive accuracy
C) Number of parameters
D) Computational efficiency

Answer: B - WAIC estimates how well the model predicts new, unseen data.

Question 4

If the 95% credible interval for β (slope) is [-0.5, 0.3], what trading signal does this

A) Strong bullish (go long)
B) Strong bearish (go short)
C) Neutral (insufficient evidence for direction)
D) Extremely volatile

Answer: C - The interval contains zero, indicating we cannot confidently determine if the trend is or negative.

Question 5

In the natural gas example, why did we standardize predictors before modeling?

A) Required by PyMC
B) Improves sampling efficiency and prior specification
C) Makes coefficients exactly equal
D) Reduces data size

Answer: B - Standardization puts all predictors on the same scale, improving MCMC sampling a allowing comparable priors on coefficients.

Exercises

Exercise 1: Gold Price Regression

Build a Bayesian linear regression model for gold prices using:

- USD index (negative relationship expected)
- Real interest rates (negative relationship expected)
- VIX (volatility index, positive relationship expected)

Compare Normal vs Student-t likelihood. Which performs better?

Exercise 2: Model Comparison

For the crude oil multiple regression example:

1. Build a model with only supply
2. Build a model with only demand
3. Build a model with only USD index
4. Compare all models using WAIC
5. Which single predictor is most important?

Exercise 3: Forecast Uncertainty

Using the natural gas model:

1. Generate forecasts for the next 30 days
2. Assume a cold snap increases HDD by 50%
3. Quantify the impact on price forecasts
4. Calculate the probability that prices exceed $5/MMBtu

Exercise 4: Prior Sensitivity

For the simple linear regression:

1. Try a strongly informative prior: β ~ Normal(0.2, 0.05)

2. Try an uninformative prior: β ~ Normal(0, 100)
3. Compare posteriors and predictions
4. When does the prior matter most?

Exercise 5: Trading Strategy

Design a trading strategy for natural gas:

1. Use the model to forecast next week's average price
2. If P(price > current price) > 0.75, go long
3. If P(price < current price) > 0.75, go short
4. Otherwise, stay flat
5. Backtest on synthetic data

Summary

In this module, you learned:

1. **Simple Bayesian Linear Regression**: Model price trends with full uncertainty quantificatior
2. **Posterior Interpretation**: Extract trading signals from posterior distributions
3. **Credible Intervals**: Direct probability statements for risk management
4. **Multiple Regression**: Incorporate fundamental drivers (supply, demand, FX)
5. **Model Comparison**: Use WAIC/LOO to select the best model
6. **Robust Regression**: Handle outliers and fat tails with Student-t likelihood
7. **Natural Gas Application**: Real-world example with weather and storage data

Key Takeaways for Trading

- Bayesian regression provides **actionable uncertainty** for position sizing
- **Student-t likelihood** is essential for commodity data with extreme events
- **Model comparison** prevents overfitting and improves out-of-sample performance
- **Incorporate fundamentals** (supply, demand, weather) for better forecasts
- **Credible intervals** directly answer "What's the probability of X?"

Preview of Next Module

Module 6: Bayesian Structural Time Series (BSTS)

Linear regression assumes constant relationships. But commodity markets have:

- **Trends** that change over time
- **Seasonality** (winter heating, summer cooling)
- **Dynamic relationships** (correlations that evolve)

In Module 6, we'll build Bayesian Structural Time Series models that decompose prices into:

- Local level (random walk)

- Local linear trend (changing slopes)
- Seasonal components (daily, weekly, annual)
- Dynamic regression coefficients

You'll learn to model **time-varying volatility** and **structural breaks** crucial for commodity tradin

See you in Module `6`!