Lesson 1: PyMC Fundamentals and Model Setup

Learning Objectives

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

- Understand PyMC architecture and ecosystem
- Learn basic model specification syntax
- Master probability distributions in PyMC
- Implement your first Bayesian model

Prerequisites

- Python programming experience
- Basic Bayesian statistics knowledge
- NumPy and matplotlib familiarity

1. Introduction to PyMC

PyMC is a probabilistic programming framework that allows you to build Bayesian models using Python syntax. The current version (PyMC 5.x) is built on top of PyTensor for automatic differentiation and uses advanced MCMC samplers like NUTS.

Key Components:

- **Model**: Container for your probabilistic model
- **Distributions**: Building blocks for random variables
- **Samplers**: Algorithms for posterior inference
- **ArviZ**: Analysis and visualization toolkit

Installation

```
pip install pymc arviz matplotlib seaborn
```

2. Basic Model Structure

Every PyMC model follows this basic structure:

```
import pymc as pm
import numpy as np
import matplotlib.pyplot as plt
import arviz as az
```

```
# Basic model template
with pm.Model() as model:
    # Define priors
    parameter = pm.Normal('parameter', mu=0, sigma=1)

    # Define likelihood
    obs = pm.Normal('obs', mu=parameter, sigma=1, observed=data)

    # Sample from posterior
    trace = pm.sample(1000, return_inferencedata=True)
```

3. Common Distributions in PyMC

Continuous Distributions

```
# Normal distribution
x = pm.Normal('x', mu=0, sigma=1)

# Uniform distribution
y = pm.Uniform('y', lower=0, upper=10)

# Exponential distribution
z = pm.Exponential('z', lam=1.0)

# Beta distribution
w = pm.Beta('w', alpha=2, beta=3)
```

Discrete Distributions

```
# Bernoulli distribution
success = pm.Bernoulli('success', p=0.5)

# Binomial distribution
count = pm.Binomial('count', n=10, p=0.3)

# Poisson distribution
events = pm.Poisson('events', mu=5)
```

4. Practical Example: Simple Linear Regression

Let's build a complete Bayesian linear regression model:

```
import pymc as pm
import numpy as np
import matplotlib.pyplot as plt
import arviz as az

# Set random seed for reproducibility
np.random.seed(42)
```

```
# Generate synthetic data
n_points = 100
true_slope = 2.5
true_intercept = 1.0
true_sigma = 0.5

x = np.linspace(0, 10, n_points)
y = true_intercept + true_slope * x + np.random.normal(0, true_sigma, n_points)

# Build Bayesian linear regression model
with pm.Model() as linear_model:
    # Priors
    intercept = pm.Normal('intercept', mu=0, sigma=10)
    slope = pm.Normal('slope', mu=0, sigma=10)
    sigma = pm.HalfNormal('sigma', sigma=1)

    # Expected value
    mu = intercept + slope * x

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

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

# Analyze results
print(az.summary(trace, hdi_prob=0.95))
```

5. Model Visualization and Diagnostics

```
# Plot trace plots
az.plot_trace(trace, var_names=['intercept', 'slope', 'sigma'])
plt.show()

# Plot posterior distributions
az.plot_posterior(trace, var_names=['intercept', 'slope', 'sigma'],
                  ref_val=[true_intercept, true_slope, true_sigma])
plt.show()

# Check convergence diagnostics
print(f"R-hat values: {az.rhat(trace)}")
print(f"Effective sample size: {az.ess(trace)}")
```

6. Exercises

Exercise 1: Basic Distribution Exploration

Create a model with different prior distributions and sample from them:

```
with pm.Model() as prior_model:
    # Create various priors
```

```
    normal_prior = pm.Normal('normal', mu=0, sigma=1)
    uniform_prior = pm.Uniform('uniform', lower=-2, upper=2)
    exponential_prior = pm.Exponential('exponential', lam=1)

    # Sample from priors only
    prior_samples = pm.sample_prior_predictive(samples=1000)

# Plot and compare the prior distributions
# [Your implementation here]
```

Exercise 2: Polynomial Regression

Extend the linear regression to a quadratic model:

```
# Generate quadratic data
x_quad = np.linspace(-3, 3, 100)
y_quad = 2 + 0.5 * x_quad + 1.5 * x_quad**2 + np.random.normal(0, 0.5, 100)

with pm.Model() as quadratic_model:
    # Define priors for intercept, linear, and quadratic terms
    # [Your implementation here]

    # Define likelihood
    # [Your implementation here]

    # Sample
    trace_quad = pm.sample(2000, tune=1000, return_inferencedata=True)
```

Exercise 3: Model with Multiple Observations

Create a model that handles multiple dependent variables:

```
# Generate data with two related outcomes
n = 50
x = np.random.normal(0, 1, n)
y1 = 2 + 3 * x + np.random.normal(0, 0.5, n)
y2 = 1 + 1.5 * x + np.random.normal(0, 0.3, n)

with pm.Model() as multi_outcome_model:
    # Create model with shared predictor but different intercepts/slopes
    # [Your implementation here]
    pass
```

7. Key Takeaways

1. **Model Context**: Always use `with pm.Model() as model:` to define your model
2. **Naming**: Give descriptive names to your random variables
3. **Priors**: Choose appropriate priors based on domain knowledge
4. **Observed Data**: Use the `observed` parameter to condition on data

5. **Sampling**: Use `pm.sample()` for posterior inference
6. **Diagnostics**: Always check convergence using R-hat and effective sample size

8. Common Pitfalls

1. **Forgetting the model context**: Always define variables within the model context
2. **Poor prior choices**: Avoid overly vague or restrictive priors
3. **Ignoring convergence**: Check diagnostics before interpreting results
4. **Shape mismatches**: Ensure data shapes match model expectations

9. Next Steps

In the next lesson, we'll dive deeper into MCMC sampling, learn about different samplers, and master convergence diagnostics. You'll also learn how to handle sampling issues and optimize performance.

10. Further Reading

- PyMC Documentation
- Bayesian Analysis with Python by Osvaldo Martin
- PyMC Examples Gallery

Time to complete: 2-3 hours
Difficulty: Beginner
Prerequisites: Python, Basic Bayesian Statistics