```python
        # Cache for production queries
        if not timestamp:
            self.cache.setex(cache_key, 3600, json.dumps(features))  # 1 hour TTL

    return features
```

Benefits:

- **Consistency**: Training and production use same feature logic
- **Time-travel**: Can recreate features as they existed at training time
- **Efficiency**: Caching reduces redundant computation
- **Debugging**: Easy to inspect what features were used for a prediction

7.4 Model Registry and Versioning

Track all model versions systematically:

python

```yaml
# models/registry.yaml
models:
  - version: "1.0"
    algorithm: "XGBoost"
    training_date: "2024-06-01"
    training_data: "s3://data/training_2024_q1_q2.parquet"
    hyperparameters:
      n_estimators: 200
      max_depth: 8
      learning_rate: 0.05
    performance:
      rmse: 5.2
      mae: 3.8
      r2: 0.62
    status: "retired"

  - version: "1.1"
    algorithm: "XGBoost"
    training_date: "2024-09-01"
    training_data: "s3://data/training_2024_q2_q3.parquet"
    hyperparameters:
      n_estimators: 300
      max_depth: 10
      learning_rate: 0.03
    performance:
```

```yaml
    rmse: 4.8
    mae: 3.5
    r2: 0.65
  features_added:
    - "text_features_tfidf"
    - "synopsis_topics_lda"
  status: "production"

- version: "2.0"
  algorithm: "LightGBM"
  training_date: "2025-01-15"
  training_data: "s3://data/training_2024_q3_q4.parquet"
  hyperparameters:
    n_estimators: 500
    max_depth: 12
    learning_rate: 0.02
  performance:
    rmse: 4.5
    mae: 3.2
    r2: 0.68
  features_added:
    - "competitive_pricing"
    - "bert_synopsis_embeddings"
```

status: "canary" *# Testing on 10% of traffic*

Automated model comparison:

python

```python
def compare_models(baseline_version, candidate_version, test_data):
    baseline = load_model(baseline_version)
    candidate = load_model(candidate_version)

    baseline_preds = baseline.predict(test_data[features])
    candidate_preds = candidate.predict(test_data[features])

    results = {
        'baseline': {
            'rmse': rmse(test_data['price'], baseline_preds),
            'mae': mae(test_data['price'], baseline_preds),
            'r2': r2_score(test_data['price'], baseline_preds)
        },
        'candidate': {
            'rmse': rmse(test_data['price'], candidate_preds),
```

```
            'mae': mae(test_data['price'], candidate_preds),
            'r2': r2_score(test_data['price'], candidate_preds)
        }
    }

    # Statistical significance test
    baseline_errors = np.abs(test_data['price'] - baseline_preds)
    candidate_errors = np.abs(test_data['price'] - candidate_preds)

    t_stat, p_value = stats.ttest_rel(baseline_errors, candidate_errors)

    results['improvement'] = {
        'rmse_delta': results['baseline']['rmse'] - results['candidate']['rmse'],
        'mae_delta': results['baseline']['mae'] - results['candidate']['mae'],
        'statistically_significant': p_value < 0.05,
        'p_value': p_value
    }

return results
```

7.5 Documentation and Knowledge Transfer

Critical for long-term maintenance:

1. Model Card (inspired by Google's Model Cards framework):

markdown

```markdown
# Book Price Prediction Model v1.1

## Model Details
- **Developed by**: Data Science Team
- **Model date**: September 2024
- **Model type**: Gradient Boosted Trees (XGBoost)
- **Model version**: 1.1
- **License**: Proprietary

## Intended Use
- **Primary use**: Suggest listing prices for used books on marketplace
- **Primary users**: Individual sellers, small resellers
- **Out-of-scope**: Rare collectibles >$500, damaged books

## Training Data
- **Source**: eBay sold listings, January-August 2024
```

○ - **Size**: 150,000 transactions
○ - **Geographic scope**: United States
○ - **Filters applied**:
○   - Removed outliers (<$1 or >$300)
○   - Excluded auctions with <2 bids
○   - Required complete condition information
○
○ ## Performance
○ - **Test RMSE**: $4.80
○ - **Test MAE**: $3.50
○ - **R^2**: 0.65
○
○ ### Subgroup Performance
○ | Category | RMSE | MAE | N |
○ |----------|------|-----|---|
○ | Textbooks | $3.20 | $2.40 | 45k |
○ | Fiction | $2.10 | $1.60 | 62k |
○ | Non-Fiction | $5.50 | $4.10 | 38k |
○ | Collectibles | $18.00 | $12.50 | 5k |
○
○ ## Limitations
○ - Struggles with books that have <5 historical sales
○ - Does not account for signed copies or special editions well
○ - Performance degrades for books >10 years old
○ - May overpredict prices during market downturns
○
○ ## Ethical Considerations
○ - Does not use user demographics for pricing (no discrimination)
○ - Includes caps to prevent surge pricing during emergencies
○ - Provides uncertainty estimates to prevent overconfidence
○
○ ## Monitoring
○ - Retrained quarterly with fresh data
○ - Performance monitored weekly

- Alerts if RMSE exceeds $6.00 on validation set

2. Runbook for on-call engineers:

markdown

○ # Price Prediction Service Runbook
○
○ ## Common Issues
○

- ### Issue: Prediction latency >2 seconds
- **Symptoms**: Users report slow price suggestions
- **Diagnosis**:
- ```bash
- # Check API response times
- curl -w "@curl-format.txt" https://api.bookprice.com/predict
-
- # Check model inference time
- docker logs price-prediction-service | grep "inference_time"
- ```
- **Resolution**:
- - If >500ms: Check if model file is on slow storage (should be in memory)
- - If database slow: Check if Redis cache is hit (should be >80%)
- - Escalate if issue persists >30 min
-
- ### Issue: Predictions seem wrong (user reports)
- **Symptoms**: Multiple user complaints about inaccurate prices
- **Diagnosis**:
- ```bash
- # Check recent prediction distribution
- SELECT AVG(predicted_price), STDDEV(predicted_price)
- FROM predictions
- WHERE timestamp > NOW() - INTERVAL '1 hour';
-
- # Compare to historical baseline
- # Alert if mean shifted >20%
- ```
- **Resolution**:
- - Check if model version changed recently (rollback if needed)
- - Check data freshness (stale competitive data?)
- - Review recent A/B test deployments
-
- ### Issue: Missing features error
- **Symptoms**: Predictions fail with "KeyError: 'avg_rating'"
- **Diagnosis**:
- ```python
- # Check feature store
- features = feature_store.get_features(isbn="978-0134685991")
- print(features.keys())
- ```
- **Resolution**:
- - If API call to Goodreads failed: Use fallback (median rating)
- - If book not in database: Return "insufficient data" response

- Log ISBNs that frequently fail (may need better fallback logic)

3. Feature documentation:

python

```python
# features/documentation.py

FEATURE_DEFINITIONS = {
    'book_age': {
        'description': 'Number of years since publication',
        'type': 'numeric',
        'range': [0, 150],
        'source': 'Calculated from pub_year in books table',
        'importance': 0.12,  # From SHAP analysis
        'notes': 'Very old books (>50 years) may be collectibles; consider nonlinear effects'
    },

    'condition_ordinal': {
        'description': 'Numeric encoding of condition',
        'type': 'ordinal',
        'mapping': {'New': 5, 'Like New': 4, 'Very Good': 3, 'Good': 2, 'Acceptable': 1},
        'source': 'User-provided condition at listing time',
        'importance': 0.35,
        'notes': 'Most important feature; ensure consistent grading across platforms'
    },

    'has_signed': {
        'description': 'Boolean indicating if book is signed by author',
        'type': 'boolean',
        'source': 'Keyword extraction from title/description',
        'keywords': ['signed', 'autographed', 'inscribed'],
        'importance': 0.08,
        'notes': 'Can increase price 50-200%; verify authenticity in production'
    },

    # ... all features documented
}
```

---

8. Conclusion and Key Takeaways

8.1 Summary of Best Approaches

For most used book pricing applications, the winning combination is:

1. **Algorithm**: Gradient Boosting (XGBoost or LightGBM)
   - Best accuracy-complexity trade-off
   - Handles mixed data types naturally
   - Provides feature importance
2. **Features** (in order of importance):
   - Condition (ordinal encoding)
   - Text features from title/description (TF-IDF or keywords)
   - Edition/publication year
   - Format (hardcover vs. paperback)
   - Popularity metrics (ratings, reviews)
   - Competitive landscape (current offers, lowest price)
3. **Target transformation**: Log-price
   - Handles skewed distribution
   - Optimizes relative error
   - Use RMSLE as primary metric
4. **Uncertainty quantification**: Quantile regression or conformal prediction
   - Provide price ranges, not just point estimates
   - Build user trust with honest uncertainty
5. **Production considerations**:
   - Feature store for consistency
   - Automated retraining (monthly/quarterly)
   - A/B testing for model updates
   - Monitoring for drift and performance degradation

8.2 Common Pitfalls to Avoid

❌ **Using only structured features** → Missing 30-50% of predictive power from text

❌ **Training on mean/RMSE without log transform** → Poor performance on expensive books

❌ **Feature leakage** → Overly optimistic validation, fails in production

❌ **Ignoring temporal drift** → Old model becomes stale, accuracy degrades

❌ **Overfitting to text** → High-dimensional sparse features without regularization

❌ **Point estimates without uncertainty** → Users don't know when to trust predictions

❌ **Treating all books identically** → Textbooks, fiction, and collectibles need different approaches

❌ **Ignoring market dynamics** → Competitive pricing, seasonality affect real outcomes

8.3 When to Use Different Approaches

Linear Regression:

- Baseline only, or when interpretability is paramount
- Expected performance:
  -