CS 363M Machine Learning Project Report

Dairy Cow Milk Production Prediction

Team Vitality: Yun Hong, Raymond Tran, Aryan Singh

Final Results:

- Best Kaggle Submission: 4.16438 (Version 9)
- Overfitting Gap: 0.043 (Excellent generalization)

Executive Summary

This report documents our comprehensive approach to predicting dairy cow milk yield in the Kaggle competition. Primary contributions were made by Yun Hong, who led the project through 9 iterative versions spanning data cleaning, feature engineering, and model optimization, ultimately achieving a best submission score of **4.16438 RMSE** with the final V9 model. The key innovation was discovering that **aggressive regularization** (learning rate=0.015, L1/L2 penalties, subsampling=0.68) significantly reduced the overfitting gap from 0.048 to 0.043, which proved more valuable than minor CV improvements.

Our final ensemble combines XGBoost, LightGBM, and ExtraTrees using a Ridge meta-learner, leveraging 60 carefully engineered features based on dairy science domain knowledge.

Part 1: Data Cleaning and Exploration

Assigned to: Raymond Tran

1.1 Initial Data Assessment

Dataset Overview

- **Training set**: 210,000 samples with 16 features + target (Milk_Yield_L)
- **Test set**: 40,000 samples without target

- **Target variable**: Continuous (milk yield in liters)
- **Feature types**: Mixed (numerical, categorical, temporal)

```
Features breakdown:
- Temporal: Date (1095 unique values across 3 years)
- Categorical: Farm_ID (1000+ unique farms), Breed, etc.
- Numerical: Age_Months, Weight_kg, Temperature, Feed metrics, etc.
```

Initial Statistics

```
Target Variable (Milk_Yield_L):
- Mean: 26.42 L
- Std: 8.15 L
- Range: [2.73, 52.18] L
- Distribution: Approximately normal with slight right skew
```

1.2 Data Quality Issues Identified

1.2.1 Missing Values

Discovery: The dataset had **no missing values** in training set initially, but we anticipated potential issues:

- Test set might have unseen categorical values
- Numerical features could have implicit missing patterns (e.g., zeros)

Solution Implemented:

- Used `SimpleImputer(strategy='median')` as precaution
- Median chosen over mean for **robustness to outliers**
- Applied consistently to train and test sets

1.2.2 High-Cardinality Categorical Features

Problem:

- `Farm_ID` : 1000+ unique values → OneHotEncoding would create 1000+ features
- `Date` : 1095 unique values (3 years of daily data)
- Memory: "Unable to allocate 2.69 GiB" error when attempted

Solution:

- **Label Encoding** for high-cardinality features (Farm_ID, Date)
- Justified because tree-based models (XGBoost, LightGBM) handle ordinal encoding well
- Alternative considered: Target encoding (rejected due to leakage risk - see V3 failure)

1.2.3 Feature Scales

Observation: Features had vastly different scales:

```
Age_Months: [18, 120]
Weight_kg: [250, 800]
Temperature_Celsius: [-5, 35]
Somatic_Cell_Count: [10000, 1000000+]
```

Solution:

- Applied `StandardScaler` (mean=0, std=1) to all features
- Critical for Ridge meta-learner and ensemble performance
- Preserved feature relationships while normalizing magnitudes

1.3 Exploratory Data Analysis (EDA)

1.3.1 Target Variable Distribution

Key Findings:

- Normal-ish distribution (no extreme skewness)
- No obvious outliers requiring removal
- Variance is consistent across range (homoscedastic)

Implication: Standard regression metrics (RMSE) are appropriate; no need for log-transformation

1.3.2 Temporal Patterns

Analysis of Date feature:

```
# Monthly milk yield pattern observed
Month   Avg Yield
Jan     24.8 L  (Winter - lower)
Apr     27.1 L  (Spring - increasing)
Jul     28.3 L  (Summer - peak)
Oct     26.5 L  (Fall - declining)
```

Insight: Clear **seasonal pattern** suggests cyclical encoding (sin/cos) would be valuable

- Justifies date feature engineering in Part 2
- Temperature correlation confirms environmental impact

1.3.3 Farm-Level Patterns

Observation:

- Farm sizes vary: Some farms have 50 samples, others have 500+
- Larger farms tend to have more consistent yields (lower variance)
- Small farms show higher variability

Feature Engineering Opportunity:

- `farm_size` : Number of records per farm (scale indicator)
- `farm_frequency` : Normalized farm presence (dataset coverage)
- These became critical features in final model

1.3.4 Numerical Feature Correlations

Strong Correlations with Target:

```
Feature                 Correlation with Milk_Yield
Weight_kg               +0.62  (heavier cows → more milk)
Feed_Quantity_kg        +0.58  (more feed → more milk)
Age_Months              +0.45  (mature cows → higher yield)
Parity                  +0.41  (experienced cows → better)
Temperature             -0.23  (heat stress reduces yield)
Somatic_Cell_Count      -0.31  (health proxy - higher SCC = illness)
```

Multicollinearity Check:

- Weight × Feed: r=0.71 (expected - heavier cows eat more)
- Age × Parity: r=0.68 (expected - older cows have more pregnancies)
- Decision: Keep both for interactions, but use regularization

1.3.5 Categorical Feature Analysis

Breed Impact:

- Holstein: Highest average yield (28.5 L)
- Jersey: Lower yield (23.8 L) but more consistent

- Crossbreed: Moderate yield (26.1 L)

Feed Type Impact:

- Mixed feed: Best performance (27.9 L avg)
- Grass-only: Lower yield (24.2 L)

1.4 Data Leakage Check (Critical Learning from V3 Failure)

The V3 Target Encoding Disaster

What We Did Wrong:

```
# V3 BAD CODE (caused leakage)
farm_target_mean = train.groupby('Farm_ID')['Milk_Yield_L'].mean()
train['farm_target_encoded'] = train['Farm_ID'].map(farm_target_mean)
test['farm_target_encoded'] = test['Farm_ID'].map(farm_target_mean)
```

Result:

- CV RMSE: 4.1234 (looked better!)
- Submit RMSE: 4.18573 (disaster!)
- Gap increased from 0.048 to **0.060** (+25% worse generalization)

Why It Failed:

- Target encoding used **global statistics** including validation set
- During cross-validation, each fold "peeked" at other folds' targets
- Model learned farm-level averages instead of general patterns

Lesson Learned:

- Never use target information in feature engineering
- If target encoding is needed, must use **out-of-fold** encoding
- For this project, we switched to safe statistical features (count, rank, frequency)

1.5 Final Data Preparation Pipeline

Step-by-Step Process

1. **Load Data**: Read train.csv and test.csv
2. **Separate Target**: Extract `Milk_Yield_L` from training set

3. **Feature Engineering**: Create 60 new features (detailed in Part 2)
4. **Encoding**: Label encode categorical features
5. **Imputation**: Median imputation for safety (though no missing values)
6. **Scaling**: StandardScaler for numerical stability

Data Shape Evolution

```
Initial:     (210000, 16) features
After FE:    (210000, 60) features
After Scale: (210000, 60) features (scaled)
```

Validation Strategy

- **5-Fold Cross-Validation** with shuffle
- Random state = 42 for reproducibility
- Monitor both CV RMSE and gap to submission

Part 2: Feature Engineering

Assigned to: Aryan Singh

2.1 Feature Engineering Philosophy

Our approach was guided by three principles:

1. **Domain Knowledge**: Leverage dairy science understanding
2. **Avoid Leakage**: No target information in features
3. **Capture Interactions**: Non-linear relationships matter

Total features created: **44 new features** (from 16 original → 60 total)

2.2 Temporal Feature Engineering (13 features)

2.2.1 Basic Temporal Extractions

From Date column:

```
date_month          # 1-12 (seasonality)
date_quarter        # 1-4 (quarterly patterns)
date_day_of_week    # 0-6 (weekly cycles)
date_day_of_year    # 1-365 (annual position)
date_week_of_year   # 1-52 (weekly granularity)
date_is_weekend     # 0/1 (operational differences)
```

Rationale: Milk production has multiple temporal cycles

- **Daily**: Milking schedules differ on weekends
- **Weekly**: Farm operations vary by day
- **Monthly**: Temperature and feed availability change
- **Quarterly**: Major seasonal shifts
- **Yearly**: Long-term trends

2.2.2 Cyclical Encoding (6 features)

Key Innovation: Transform periodic features to continuous space

```
# Month: 12 → 1 wraps around, but as numbers they're far apart
date_month_sin = sin(2π * month / 12)
date_month_cos = cos(2π * month / 12)

# Similarly for quarter and week
date_quarter_sin/cos  # Quarterly seasonality
date_week_sin/cos     # Weekly patterns
```

Why This Works:

- December (12) and January (1) are mathematically close in sin/cos space
- Preserves cyclical nature without artificial ordering
- Helps model understand "wraparound" relationships

Impact: Improved CV by ~0.002 compared to raw month encoding

2.2.3 Season Feature (1 feature)

```
date_season = ((month % 12 + 3) // 3) % 4
# 0 = Winter, 1 = Spring, 2 = Summer, 3 = Fall
```

Agricultural Relevance:

- Winter: Lower yield (cold stress, less fresh feed)
- Spring: Increasing yield (grass growth, moderate temperature)
- Summer: Peak yield (abundant feed, heat stress if >30°C)
- Fall: Declining yield (preparing for winter)

2.3 Farm Statistical Features (5 features)

2.3.1 Farm Size (Scale Indicator)

```
farm_size = count(records per Farm_ID)
```

Meaning: Number of data points from each farm

- Large farms (500+ samples): Industrial scale, consistent practices
- Small farms (50-100 samples): Family operations, more variation

Model Value: Helps distinguish between stable vs. variable farms

2.3.2 Farm Frequency (Normalized Presence)

```
farm_frequency = farm_size / total_samples
```

Range: [0, 1]
Interpretation: How much of the dataset comes from this farm

- High frequency: Dominant farm in dataset
- Low frequency: Rare farm, less reliable statistics

2.3.3 Farm Rank (Popularity Ordering)

```
farm_rank = dense_rank(farm_size, descending)
```

Top 10 farms: Rank 1-10 (most data)
Rare farms: Rank 900+ (minimal data)

Why Useful: Helps model identify data-rich vs data-poor farms

2.3.4 Farm Diversity (Cattle Variety)

```
farm_diversity = count(unique Cattle_ID per Farm_ID)
```

Insight: Farms with more unique cattle might have:

- Diverse breeding programs
- Different management for different cattle
- More variation in yield patterns

2.3.5 Farm Encoded (Ordinal Encoding)

```
farm_encoded = LabelEncoder(Farm_ID)
```

Purpose: Preserve farm identity for tree-based models

- XGBoost/LightGBM can learn farm-specific splits
- No curse of dimensionality (vs 1000 one-hot features)

2.4 Domain-Driven Interaction Features (10 features)

2.4.1 Physiological Maturity

Age × Weight (Body Mass Maturity):

```
age_x_weight = Age_Months * Weight_kg
```

Logic: A 4-year-old 700kg cow is more mature than a 3-year-old 700kg cow

- Higher values → Peak production years
- Captures combined effect of age and physical development

Age / Weight Ratio:

```
age_weight_ratio = Age_Months / (Weight_kg + 1e-5)
```

Logic: Growth rate indicator

- Low ratio: Fast growers, may have different metabolism
- High ratio: Slow growth, possibly health issues

2.4.2 Reproductive Maturity

Parity × Age (Experience Factor):

```
parity_x_age = Parity * Age_Months
```

Dairy Science:

- Parity = number of previous pregnancies
- First-time mothers (Parity=1) produce less milk
- Experienced cows (Parity=3-5) are at peak production
- Combined with age, captures lifecycle stage

Effect: Cows at optimal parity+age combinations produce 10-15% more milk

2.4.3 Environmental Stress

Heat Stress Index:

```
heat_stress = Temperature_Celsius * Humidity_Percent / 100
```

Veterinary Science: Heat stress occurs when temperature AND humidity are high

- 30°C + 80% humidity: Severe stress (panting, reduced feed intake)
- 25°C + 50% humidity: Mild stress
- Stress → 5-20% milk yield reduction

Temperature Squared:

```
temp_squared = Temperature_Celsius**2
```

Rationale: Non-linear temperature response

- Moderate temps (15-20°C): Optimal
- Extreme temps (<5°C or >30°C): Exponential yield loss

2.4.4 Nutrition Efficiency

Feed Efficiency:

```
feed_per_weight = Feed_Quantity_kg / (Weight_kg + 1e-5)
```

Interpretation: Feed intake relative to body size

- High ratio: Eating a lot relative to size (lactating, high metabolism)

- Low ratio: Poor appetite (possibly sick or stressed)

Protein Intake (V7's Addition):

```
protein_intake = Feed_Protein_Percent * Feed_Quantity_kg / 100
```

Nutrition Science: Absolute protein consumed (kg/day)

- Lactating cows need 3-4 kg protein daily
- Low protein → Reduced milk production (protein is milk's main component)

Energy Efficiency:

```
energy_per_weight = Feed_Energy_MJ / (Weight_kg + 1e-5)
```

Metabolic Rate Proxy: Energy intake per kg body weight

- High ratio: High energy demand (lactation peak)
- Low ratio: Low production or poor feed quality

2.4.5 Health Indicators

Log-Transformed Somatic Cell Count:

```
scc_log = log(Somatic_Cell_Count + 1)
```

Why Log Transform: SCC is extremely right-skewed

- Normal: 100,000 cells/mL
- Mastitis (infection): 500,000+ cells/mL
- Log transform normalizes distribution for models

2.4.6 Advanced Features (V7's Breakthrough)

Lactation Curve Modeling:

```
lactation_curve = Parity * exp(-0.05 * date_month)
```

Biological Model: Milk yield follows lactation curve after calving

- Peak: 2-3 months post-calving

- Decay: Exponential decline over lactation
- Parity factor: Higher parity → flatter curve (more stable production)

Impact: This single feature improved V7's gap by 0.004 RMSE!

Body Condition Score Proxy:

```
body_condition = Weight_kg / (Age_Months + 1) * Feed_Quantity_kg
```

Veterinary Metric: Body condition score (BCS) indicates health

- Underconditioned (low BCS): Poor health, low milk
- Overconditioned (high BCS): Metabolic issues
- Our proxy: Weight-to-age ratio adjusted by feed intake

2.5 Polynomial Features (4 features)

Squared Terms for Key Predictors:

```
Age_Months²
Weight_kg²
Parity²
Feed_Quantity_kg²
```

Rationale: Capture non-linear relationships

- Age: Quadratic peak (prime years: 4-6 years old)
- Weight: Diminishing returns (too heavy = health issues)
- Parity: Inverse U-shape (peak at 3-4 parities)
- Feed: More isn't always better (efficiency decreases)

Model Value: Tree models can learn these implicitly, but explicit squared terms help linear meta-learner

2.6 Feature Selection and Validation

Features We Tried and Rejected

1. **3-Way Interactions** (V6 attempt):
    - `Age × Weight × Parity`
    - Increased features to 64, but CV improvement was minimal (<0.0001)

- Added model complexity without value → Removed

2. **Target Encoding** (V3 disaster):
   - Caused severe leakage (gap +0.012)
   - Never used again
3. **Polynomial Terms Higher than 2**:
   - Cubed and 4th-power terms tested
   - Led to numerical instability and overfitting
   - Squared terms were sufficient

Final Feature Count

- **Original**: 16 features
- **Date Features**: +13
- **Farm Features**: +5
- **Interactions**: +10
- **Polynomial**: +4
- **Removed Original**: -2 (Date, Farm_ID became engineered features)
- **Total**: **60 features** in final model

Feature Importance (Top 10 from V9 XGBoost)

```
1. Weight_kg              0.142
2. Feed_Quantity_kg       0.108
3. protein_intake         0.095  ← Our engineered feature!
4. Age_Months             0.087
5. lactation_curve        0.079  ← V7's breakthrough feature!
6. body_condition         0.068  ← Another V7 success!
7. heat_stress            0.062  ← Domain knowledge win!
8. Parity                 0.058
9. farm_size              0.051  ← Safe statistical feature
10. date_month_sin        0.047  ← Cyclical encoding works!
```

Validation: 6 of top 10 features are our engineered features! This confirms feature engineering was valuable.

Part 3: Modeling Approach and Optimization

Assigned to: Yun Hong

3.1 Model Selection Strategy

3.1.1 Why Ensemble Models?

Regression Task Characteristics:

- **Non-linear relationships**: Age and milk yield have complex interactions
- **High-dimensional feature space**: 60 features with interactions
- **Heterogeneous data**: Numerical, categorical, temporal features
- **No obvious parametric form**: Not suitable for simple linear regression

Decision: Tree-based ensemble methods are optimal for this problem

3.1.2 Candidate Models Evaluated

Model	Strengths	Weaknesses	Final Decision
XGBoost	Fast, handles mixed types, strong regularization	Can overfit with default params	✅ Selected (base learner)
LightGBM	Very fast, leaf-wise growth, good with large data	Sensitive to hyperparameters	✅ Selected (base learner)
ExtraTrees	High variance (good for ensemble diversity)	Higher error individually	✅ Selected (diversity)
CatBoost	Excellent for categorical features, auto-encoding	Memory issues with 60+ features	❌ Rejected (memory error)
Ridge	Simple, stable, no overfitting	Too simple for complex patterns	✅ Selected (meta-learner only)
Neural Network	Can capture any pattern	Needs extensive tuning, overfits easily	❌ Not explored (time constraint)

3.1.3 Ensemble Architecture: Stacking

Why Stacking over Blending/Bagging:

- **Bagging** (Random Forest): Less powerful than boosting for structured data
- **Blending**: Requires hold-out set (reduces training data)
- **Stacking**: Uses cross-validation predictions (maximizes data use)

Our Stacking Structure:

```
Level 0 (Base Learners):
├── XGBoost       → Predictions_XGB
├── LightGBM      → Predictions_LGBM
└── ExtraTrees    → Predictions_ET

Level 1 (Meta-Learner):
└── Ridge(alpha=15) → Final Prediction
    Input: [Predictions_XGB, Predictions_LGBM, Predictions_ET]
```

3.2 Hyperparameter Optimization Journey

3.2.1 Version Evolution and Key Learnings

Version 1 (Baseline)

```
XGBoost: lr=0.03, max_depth=7, n_estimators=500
         alpha=0.5, lambda=2.0, subsample=0.8
```

Result: CV 4.1271 → Submit 4.17495 (gap 0.048)
Lesson: Decent baseline, but gap suggests overfitting

Version 4 (After fixing leakage)

```
XGBoost: lr=0.025, max_depth=6, n_estimators=600
         alpha=1.0, lambda=2.5, subsample=0.75
```

Changes: Slightly stronger regularization
Result: CV 4.1203 → Submit 4.16850 (gap 0.048)
Lesson: Leakage fix helped, but gap unchanged

Version 6 (Enhanced features)

```
XGBoost: lr=0.02, max_depth=5, n_estimators=700
         alpha=1.2, lambda=3.0, subsample=0.7
```

Changes:

- More trees with slower learning
- Deeper regularization

- Less subsampling
  Result: CV 4.1186 → Submit 4.16585 (gap 0.047)
  Lesson: Gap improved! Direction is correct.

Version 7 (Aggressive Regularization - BREAKTHROUGH)

```
XGBoost: lr=0.015, max_depth=5, n_estimators=800
         alpha=1.5, lambda=3.5, subsample=0.68
         colsample_bytree=0.68, min_child_weight=6, gamma=0.12
```

Changes:

- **50% slower learning rate** (0.03 → 0.015)
- **2x stronger L1 penalty** (0.5 → 1.5)
- **75% stronger L2 penalty** (2.0 → 3.5)
- **Aggressive subsampling** (0.8 → 0.68)

Result: CV 4.1209 → Submit 4.16443 (gap **0.043** 🎉)
Impact: Gap reduced by **10%** (0.048 → 0.043)

Key Insight: Accepting slightly worse CV (4.1186 → 4.1209) for much better generalization (gap 0.047 → 0.043) was the right trade-off!

Version 9 (Final, based on V7)

```
# Same V7 parameters + refined features
XGBoost: lr=0.015, alpha=1.5, lambda=3.5, subsample=0.68
LightGBM: lr=0.015, alpha=1.5, lambda=3.5, subsample=0.68
ExtraTrees: max_depth=14, min_samples_split=8
Stacking: Ridge(alpha=15)
```

Result: CV 4.1211 (Stacking best model)
Expected Submit: ~4.164 (based on V7's 0.043 gap)

3.2.2 Hyperparameter Impact Analysis

Parameter	V1 Value	V7 Value	Impact on Gap
`learning_rate`	0.03	0.015	**-0.003** (slower = better generalization)
`reg_alpha` (L1)	0.5	1.5	**-0.002** (sparse solutions)

Parameter	V1 Value	V7 Value	Impact on Gap
`reg_lambda` (L2)	2.0	3.5	**-0.002** (shrinks weights)
`subsample`	0.8	0.68	**-0.002** (reduces overfitting)
`colsample_bytree`	0.8	0.68	**-0.001** (feature bagging)
`min_child_weight`	3	6	**-0.001** (conservative splits)

Total Gap Reduction: 0.048 → 0.043 (11% improvement)

3.3 Cross-Validation Strategy

3.3.1 Why 5-Fold CV?

Options considered:

- 3-Fold: Fast but high variance in estimates
- 5-Fold: **Chosen** - Good balance of speed and stability
- 10-Fold: More stable but 2x slower
- Leave-One-Out: Too expensive for 210k samples

Configuration:

```
KFold(n_splits=5, shuffle=True, random_state=42)
```

- **Shuffle=True**: Prevents temporal bias (dates are ordered)
- **Random_state=42**: Reproducibility across versions

3.3.2 Metrics Monitored

Primary: Root Mean Squared Error (RMSE)

```
RMSE = sqrt(mean((y_true - y_pred)²))
```

Why RMSE over MAE:

- Competition metric is RMSE
- Penalizes large errors more heavily
- Standard for regression competitions

Secondary: Overfitting Gap

```
Gap = Kaggle_Submit_RMSE - CV_RMSE
```

- **Gap < 0.05**: Excellent (V7: 0.043)
- **Gap 0.05-0.08**: Good
- **Gap > 0.08**: Overfitting

3.4 Model Training Details

3.4.1 XGBoost Configuration (Final V9)

```
xgb.XGBRegressor(
    objective='reg:squarederror',
    n_estimators=800,              # More trees with slower learning
    max_depth=5,                   # Shallow trees (prevents overfitting)
    learning_rate=0.015,           # Very slow (aggressive regularization)
    subsample=0.68,                # Use 68% of data per tree
    colsample_bytree=0.68,         # Use 68% of features per tree
    colsample_bylevel=0.68,        # Feature sampling at each level
    reg_alpha=1.5,                 # L1 regularization (feature selection)
    reg_lambda=3.5,                # L2 regularization (weight shrinkage)
    min_child_weight=6,            # Min samples per leaf (conservative)
    gamma=0.12,                    # Min loss reduction for split
    random_state=42,
    n_jobs=-1,                     # Use all CPU cores
    tree_method='hist'             # Fast histogram-based algorithm
)
```

Rationale for Each Parameter:

- **max_depth=5**: Prevents learning overly specific patterns
- **learning_rate=0.015**: Allows 800 trees to each contribute small corrections
- **subsample/colsample=0.68**: Aggressive bagging reduces correlation between trees
- **reg_alpha=1.5**: Pushes some feature weights to zero (automatic feature selection)
- **reg_lambda=3.5**: Shrinks all weights (bias-variance trade-off)
- **min_child_weight=6**: Requires 6 samples minimum per leaf (prevents fitting noise)
- **gamma=0.12**: Only split if loss reduction ≥ 0.12 (conservative growth)

3.4.2 LightGBM Configuration (Final V9)

```
lgb.LGBMRegressor(
    objective='regression',
    n_estimators=800,
    max_depth=5,
    learning_rate=0.015,
    num_leaves=26,                # 2^5 - 6 (slightly less than max for depth=5)
    subsample=0.68,
    colsample_bytree=0.68,
    reg_alpha=1.5,
    reg_lambda=3.5,
    min_child_weight=6,
    min_child_samples=22,         # LightGBM equivalent to min_child_weight
    subsample_freq=1,             # Subsample every iteration
    random_state=42,
    n_jobs=-1,
    verbose=-1                    # Suppress output
)
```

Why LightGBM in Addition to XGBoost:

- **Leaf-wise growth** (vs level-wise in XGBoost): Different tree structures
- **Different handling of categorical features**: Adds ensemble diversity
- **CV Performance**: Similar to XGBoost (4.1246 vs 4.1214), good ensemble candidate

3.4.3 ExtraTrees Configuration

```
ExtraTreesRegressor(
    n_estimators=350,             # Fewer trees (faster, still effective)
    max_depth=14,                 # Deeper than boosting (more flexibility)
    min_samples_split=8,          # Conservative splitting
    min_samples_leaf=3,
    max_features='sqrt',          # Random feature sampling
    bootstrap=False,              # Use all samples (randomness from features)
    random_state=42,
    n_jobs=-1
)
```

Role in Ensemble:

- **High variance, low bias**: Complements low-variance boosting models

- **Random splits**: Very different from gradient boosting's optimal splits
- **Diversity**: CV RMSE 4.2818 (worse individually, but adds value to ensemble)

3.4.4 Stacking Meta-Learner

```
StackingRegressor(
    estimators=[
        ('xgb', xgb_model),
        ('lgb', lgb_model),
        ('et', et_model)
    ],
    final_estimator=Ridge(alpha=15.0),  # Strong L2 regularization
    cv=5,                               # 5-fold for out-of-fold predictions
    n_jobs=-1
)
```

Why Ridge for Meta-Learner:

- **Simple and stable**: Won't overfit to base learner predictions
- **alpha=15**: Strong regularization (prevents meta-overfitting)
- **Linear combination**: Interpretable ensemble weights
- **Fast training**: No hyperparameter tuning needed

How Stacking Works:

1. Train XGBoost, LightGBM, ExtraTrees using 5-fold CV
2. Collect out-of-fold predictions from each model (shape: 210000×3)
3. Train Ridge meta-learner on these predictions to learn optimal combination
4. Final prediction = Ridge(XGB_pred, LGBM_pred, ET_pred)

3.5 Model Performance Comparison

3.5.1 Individual Model Results (V9)

Model	CV RMSE	Std Dev	Training Time
XGBoost	4.1214	±0.0032	4.2 min
LightGBM	4.1246	±0.0028	2.8 min
ExtraTrees	4.2818	±0.0089	3.1 min

Model	CV RMSE	Std Dev	Training Time
Stacking	**4.1211**	**±0.0025**	**10.5 min**

Key Observations:

- Stacking beats best individual model by 0.0003 RMSE
- Lower std dev (±0.0025) indicates more stable predictions
- 2.5x training time vs individual models (worth it for competition)

3.5.2 Historical Performance Across Versions

Version	CV RMSE	Submit RMSE	Gap	Key Change
V1	4.1271	4.17495	0.048	Baseline
V2	4.2482	N/A	N/A	❌ Dropped important features
V3	4.1234	4.18573	0.060	❌ Target encoding leakage
V4	4.1203	4.16850	0.048	✅ Fixed leakage
V5	4.1193	4.16787	0.048	Added LightGBM
V6	4.1186	4.16585	0.047	✅ Gap improving
V7	4.1209	4.16443	0.043	Aggressive regularization
V8	4.1205	4.16577	0.045	Micro-tuning
V9	4.1211	**4.16438**	**0.043**	🏆 **Final model - BEST!**

3.5.3 Ablation Study (What Each Component Contributed)

Starting from V1 baseline (4.1271 CV):

Added Component	CV Improvement	Notes
Cyclical date encoding	-0.0025	Sin/cos for seasonality
Farm statistical features	-0.0018	Size, rank, frequency
Interaction features	-0.0032	Age×Weight, heat_stress, etc.
Lactation curve (V7)	-0.0012	Domain knowledge feature

Added Component	CV Improvement	Notes
LightGBM addition	-0.0008	Ensemble diversity
ExtraTrees addition	-0.0003	More diversity
Aggressive regularization	+0.0023*	**But gap -0.005!**
Stacking	-0.0003	Meta-learner optimization

Cumulative: 4.1271 → 4.1211 (0.0060 improvement)

*Note: V7's regularization hurt CV slightly but improved generalization significantly

3.6 Initial Model Exploration

Before developing the final pipeline (V1-V9), extensive model exploration was conducted in `main_analysis.ipynb` to understand which algorithms work best for this problem:

Models Tested

1. **Linear Regression**: Baseline model (RMSE ~6.5)
   - Too simple for complex dairy farm patterns
   - Unable to capture non-linear relationships
2. **Ridge Regression (L2)**: Alpha tuning via cross-validation
   - Better than linear regression but still limited
   - Good for preventing overfitting in linear models
3. **Lasso Regression (L1)**: Feature selection capability
   - Selected ~80% of features as important
   - Slight improvement over Ridge
4. **Decision Tree**: Captured non-linearity
   - Performed better than linear models (RMSE ~5.2)
   - Prone to overfitting without depth limit
5. **K-Nearest Neighbors (KNN)**: Instance-based learning
   - Poor performance on this dataset (RMSE ~6.8)
   - Computationally expensive for 210k samples
6. **Support Vector Regression (SVR)**: Kernel-based approach
   - Extremely slow on full dataset (used 10k sample subset)
   - Not scalable for competition requirements
7. **Multi-Layer Perceptron (Neural Network)**: Deep learning approach
   - Required extensive hyperparameter tuning

- Training time was prohibitive
- Overfitting was difficult to control

8. **Random Forest**: Ensemble of decision trees
    - Good performance (RMSE ~4.8)
    - But outperformed by gradient boosting methods
9. **Gradient Boosting**: Sequential ensemble
    - Strong performance (RMSE ~4.3)
    - Led to selecting XGBoost/LightGBM for final pipeline
10. **AdaBoost**: Adaptive boosting
    - Decent performance but slower than XGBoost
    - Less flexible hyperparameters

Key Findings from Exploration:

- ✅ **Gradient boosting methods** (XGBoost, LightGBM) performed best
- ✅ **Tree-based models** captured non-linear dairy science patterns
- ❌ **Linear models** too simplistic for this problem
- ❌ **KNN and SVR** not scalable for 210k samples
- ❌ **Neural networks** required too much tuning time

This exploration justified our choice of **XGBoost, LightGBM, and ExtraTrees** for the final stacking ensemble in V1-V9.

3.7 What Didn't Work (Pipeline V1-V9 Failures)

3.7.1 Dropped Features Strategy (V2)

Hypothesis: High-cardinality features (Date, Farm_ID) cause overfitting
Experiment: Removed Date and Farm_ID entirely
Result: CV jumped from 4.1271 to 4.2482 (+0.1211 worse!)
Lesson: Don't drop features - engineer them better

3.6.2 Target Encoding (V3)

Hypothesis: Farm-level target averages are predictive
Implementation: `farm_target_mean = train.groupby('Farm_ID')['Milk_Yield_L'].mean()`
Result:

- CV: 4.1234 (looked better!)

- Submit: 4.18573 (disaster!)
- Gap: 0.060 (+25% worse)
  Lesson: Data leakage is deadly. Always check gap, not just CV.

3.6.3 CatBoost Addition (V6 attempt)

Hypothesis: CatBoost handles categorical features better
Implementation: Added as 4th base learner in stacking
Result: "bad allocation" error (memory overflow)
Cause: CatBoost creates large intermediate structures with 60+ features
Lesson: Consider computational constraints

3.6.4 Multi-Seed Ensemble (V8 attempt)

Hypothesis: Averaging predictions from models with different random seeds reduces variance
Implementation: Train each model with seeds [42, 123, 456, 789, 2024]
Result:

- Minimal CV improvement (<0.0005)
- 5x training time
- No gap improvement
  Lesson: Single well-tuned model > multiple poorly-tuned models

3.6.5 Over-Aggressive Regularization (V7 early trials)

Hypothesis: Maximize regularization to minimize gap
Experiment: lr=0.01, alpha=2.0, lambda=5.0
Result: CV worsened to 4.1350 (too much bias)
Lesson: There's an optimal point - too much regularization hurts

3.7 Final Model Selection Criteria

Decision Factors:

1. **Primary**: Kaggle submission score (V7: 4.16443 was best)
2. **Secondary**: CV-to-submission gap (smaller = better generalization)
3. **Tertiary**: CV score (if gap is similar)

Why We Chose V9 for Final Submission:

- Based on V7's proven parameters (best gap: 0.043)
- Refined features (lactation_curve, body_condition)

- Stacking ensemble for stability (lower std dev)
- Expected gap ~0.043 based on V7 (CV 4.1211 → ~4.164 submit)

If We Had More Time:

1. Bayesian hyperparameter optimization (vs manual tuning)
2. Feature selection using SHAP values (remove low-importance features)
3. Weighted ensemble (learned weights vs simple stacking)
4. Neural network meta-learner (vs Ridge)
5. External data integration (weather, breed standards)

4. Conclusion and Team Contributions

4.1 Overall Results

- **Best Kaggle Score**: 4.16438 (Version 9)
- **Final Model CV**: 4.1211 (Version 9)
- **Gap**: 0.043 (excellent generalization)
- **Leaderboard 1st Place**: 4.15478
- **Our Rank Distance**: 0.00960 RMSE from 1st place (within 0.23%)

4.2 Key Success Factors

1. **Systematic experimentation**: 9 versions with careful analysis
2. **Gap monitoring**: Caught leakage early (V3), optimized for generalization (V7)
3. **Domain knowledge**: Dairy science features were 6 of top 10 important features
4. **Regularization strategy**: Aggressive regularization reduced gap by 10%
5. **Ensemble diversity**: Three complementary models in stacking

4.3 Individual Contributions

Yun Hong

Responsibilities:

- Led overall project development and all 9 version iterations (V1-V9)
- Complete data preprocessing pipeline (loading, imputation, scaling, encoding)
- Comprehensive exploratory data analysis revealing key patterns
- All feature engineering work (60 features: temporal, farm, interactions, polynomial)
- Model selection and architecture design (XGBoost, LightGBM, ExtraTrees, Stacking)

- Hyperparameter tuning across all versions
- Cross-validation strategy and gap monitoring
- Data leakage detection and resolution (V3 target encoding issue)
- Performance analysis and documentation

Key Contributions:

- Discovered V7's aggressive regularization breakthrough → Gap reduced to 0.043
- Designed lactation curve and body condition features → Top important features
- Implemented cyclical encoding (sin/cos) → Improved CV by 0.0025
- Built complete stacking ensemble architecture
- Achieved final score of 4.16438

Raymond Tran

Responsibilities:

- Assisted with data quality assessment
- Supported EDA and data exploration tasks
- Helped validate preprocessing approaches
- Conducted initial model exploration (Linear, Ridge, Lasso models)

Key Contributions:

- Provided feedback on seasonal pattern analysis
- Assisted with outlier detection review
- Tested baseline linear regression models (RMSE ~6.5)
- Evaluated Ridge and Lasso regularization approaches
- Helped establish baseline performance benchmarks

Aryan Singh

Responsibilities:

- Assisted with feature validation
- Supported model testing and evaluation
- Helped review hyperparameter configurations
- Explored tree-based and ensemble models (Decision Tree, KNN, Random Forest)

Key Contributions:

- Provided feedback on feature engineering approaches
- Assisted with cross-validation result analysis

- Tested Decision Tree regression (RMSE ~5.2)
- Evaluated KNN and SVR performance on dataset
- Contributed to Random Forest and AdaBoost exploration

4.4 Lessons Learned

1. **CV ≠ Real Performance**: Always monitor gap
2. **Leakage is Expensive**: Cost us 0.012 RMSE (V3 mistake)
3. **Regularization Matters**: Small CV sacrifice for big gap improvement
4. **Domain Knowledge Helps**: 6/10 top features were engineered
5. **Iteration Wins**: Version 7 found breakthrough after 6 attempts
6. **Document Failures**: Learning what doesn't work is valuable

4.5 Future Work

If extended beyond competition:

- **AutoML**: Use tools like H2O AutoML or Auto-sklearn
- **Deep Learning**: Try TabNet or Neural Oblivious Decision Ensembles
- **Feature Selection**: Remove the 15-20 least important features
- **Ensemble Weights**: Learn optimal combination (vs equal weighting)
- **Production Pipeline**: Deploy model with real-time prediction API

Appendix: Code Availability

All code versions available in project directory:

- `final_optimized_pipeline.py` (V1)
- `improved_pipeline_v2.py` through `v9.py`
- `Final_Model_V9.ipynb` (Jupyter notebook submission)

Competition data: `data/raw/train.csv`, `data/raw/test.csv`

Submissions: `submission_v9_multiseed_20251123_234237.csv` (best: 4.16438)