Team Name: 1-Beta

Date: November 25th, 2025

STA314 Kaggle Competition

A journey from trials and errors to final model by Kien Nguyen

Agenda

1) Intro (~2 min)

2) 3 phrases of Trial-and-errors:

   Phase 1: Establishing a Baseline (~4 min)
   Phase 2: Fixing Target & Model Engineering (~4 min)
   Phase 3: Data Augmentation Approach (~5 min)

3) Key Takeaway (~1 min)

4) Feedbacks & Discuss (~2 min)

Topics not yet discussed will quickly be introduced as highlighted: XX

Phase 1: Establishing a Baseline

A Brute Force approach to establish a baseline

Goal: understand the dataset's characteristics for future implementation

3 Linear models:

- Linear Regression -> fits everything for baseline
- Ridge -> shrinks all coefficients (doesn't eliminate any)
- Lasso -> removes irrelevant or very weak features

1 Distance-based model:

- KNN -> helps reveal curse of dimensionality

2 Tree-based models:

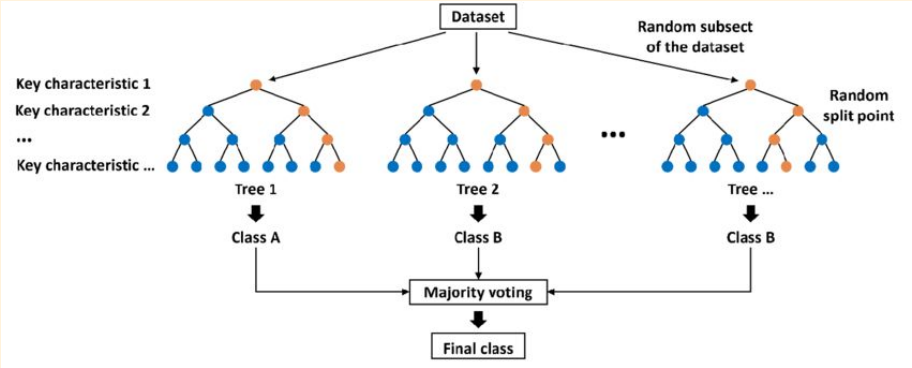
- Random Forest -> helps detect non-linear interactions
- ExtraTrees -> similar to Random Forest but instead of finding optimal split, we use random feature selection & random split/thresholds

10-Fold CV:

- Validation set 30 samples (N/10)
- Train set on 270 samples per fold
- Maximizing the signal in small datasets

Random_state = 42

- Fixed a seed gives consistent results for fair models comparison
- Prevents "lucky splits" risky with small datasets (n=300)



Phase 1: Result

Models	CV RMSE
Linear Reg	0.81950
Ridge	0.79830
Lasso	0.68971
KNN	0.92018
Random Forest	0.54434
Extra Trees	0.52402

Lasso < Ridge < OLS
=> **Only a few features** carry most of the linear signal (huge!!)

KNN = bleh (112 dimensions!!)

Extra Trees < Random Forest
=> The dataset is **moderately noisy** (112 features)

Tree models **better** than linear models
=> Relationships likely non-linear
=> Likely interaction effects linear models cannot capture

Action: submit ExtraTrees model yields RMSE **~0.58**
(Overfit, perhaps because of its high variance nature)

Phase 1: Appendix

```
N_FOLDS = 10
RANDOM_STATE = 42
TARGET_COL = 'y'
ID_COL = 'id'
```

```
def load_data():
    print("Loading data...")
    train_df = pd.read_csv('data/trainingdata.csv')
    test_df = pd.read_csv('data/test_predictors.csv')
    sample_submission = pd.read_csv('data/SampleSubmission.csv')
    return train_df, test_df, sample_submission
```

```
models = [
    ('Linear Reg', LinearRegression()),
    ('Ridge', Ridge(random_state=RANDOM_STATE)),
    ('Lasso', Lasso(random_state=RANDOM_STATE)),
    ('KNN', KNeighborsRegressor()),
    ('Random Forest', RandomForestRegressor(n_estimators=200, random_state=RANDOM_STATE)),
    ('Extra Trees', ExtraTreesRegressor(n_estimators=200, random_state=RANDOM_STATE))
]

print("\n[Comparing Models]")
results = []
kf = KFold(n_splits=N_FOLDS, shuffle=True, random_state=RANDOM_STATE)

for name, model in models:
    scores = cross_val_score(model, X, y, cv=kf, scoring='neg_root_mean_squared_error')
    rmse_score = -np.mean(scores)
    results.append((name, rmse_score))
    print(f"  {name:<15} CV RMSE: {rmse_score:.5f}")
```

Interesting fact:

Scikit-learn cross_val_score is designed so that higher scores is equal better. RMSE is a loss, and lower RMSE = better:
=> rmse_score = −RMSE

Transitioning from Phase 1 to 2

Recap from phase 1:

- Only a few features carry linear signal
- Relationships are likely non-linear
- The dataset is moderately noisy

Intuition: We can do better by building a system, not just picking a model

Phase 2 Components:

- Model: Stacking Ensemble 4 base models (Level 0) + meta-learner (Level 1)
- Feature Engineering: polynomials, stats, clusters
- Preprocessing: RobustScaler
- Feature Selection: 208 -> 108 (via SelectFromModel using Extratrees)
- Target Transformation: Standard (No transform), Yeo-Johnson, Quantile, Blend

Phase 2: Stacking Ensemble

Stacking (Diversity is key):

- Work best when different models make different errors
- Meta-learner learns to trust the more accurate model in each scenario.
- Can outperform single model.

Goal: Trial & error with ensembles by select diverse models to balance the Bias-Variance Tradeoff.

L0 Base Models:

- ExtraTreesRegressor: random splits.
- GradientBoostingRegressor: adds diversity
- XGBoost: reduce bias with regularization
- Ridge Regression: account for simple linear trends.

L1 Meta Learner:

- Ridge Regression: Combines predictions, controls collinearity with coefficient shrinkage.

XGBoost: Tree-based algorithm that builds trees sequentially, each fixing previous errors.

1) Train a base tree (often decision tree)
2) Compute the residual errors.
3) Train the next tree on the errors of the previous tree.
4) Repeat & combine

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$$

$$obj(\theta) = \sum_i^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

Phase 2: Stacking Ensemble Example

```python
# Level 0: Base Models (4 models only)
level0_models = [
    ('et', ExtraTreesRegressor(n_estimators=500, max_depth=20, min_samples_split=3,
                               max_features=0.7, random_state=42, n_jobs=-1)),
    ('xgb', XGBRegressor(n_estimators=1000, learning_rate=0.02, max_depth=5,
                         subsample=0.8, colsample_bytree=0.7, random_state=42)),
    ('gb', GradientBoostingRegressor(n_estimators=500, learning_rate=0.03,
                                     max_depth=5, random_state=42)),
    ('ridge', Ridge(alpha=10.0))
]

# Level 1: Meta Model
meta_model = Ridge(alpha=5.0)

# Create the Stacking Regressor
stacking = StackingRegressor(
    estimators=level0_models,
    final_estimator=meta_model,
    cv=5,
    n_jobs=-1
)

# Train the whole team
stacking.fit(X_train, y_train)
```

Phase 2: Feature Engineering & Preprocessing

Problem: 112 original features may miss important patterns.

Goal: Create new features that reveal hidden relationships.

Feature Engineering (Brute Force)

- **Polynomial:** Capture nonlinear curves.
- **Statistical:** Summaries statistic capture each row's overall shape
- **Cluster-distance:** Measure how close each point is to learned neighborhoods.

Outcome: 112 -> 208 features

Example: A sample with unusually high variance across features may signal an outlier class—new features help the model detect this.

Preprocessing

- **Problem:** Features on different scales; outliers distort patterns (especially with **N = 300**).
- **RobustScaler:** Uses median & IQR → focuses on the central data, ignores outliers.

Why Scaling?

- Ridge Meta model -> linear model needs scaling
- Feature Engineering creates wildly different scales**:**
  - Polynomial features -> huge values
  - Statistical features -> different ranges
  - Cluster distances -> arbitrary Euclidean scales
- Without scaling, features like [0–1] vs [0–10,000] would dominate the Ridge model.

Phase 2: Feature Selection Example

```python
# Step 4: Pick the Best Features
print("[Step 4] Feature Selection")
selector = SelectFromModel(
    ExtraTreesRegressor(n_estimators=100, random_state=42, n_jobs=-1),
    threshold='median'
)
X_selected = selector.fit_transform(X_processed, np.array(y))
X_test_selected = selector.transform(X_test_processed)
print(f"  We kept {X_selected.shape[1]} features.")
```

SelectFromModel: filter the features in a model that are important.

Threshold: "median" (keep top 50% features)

Used ExtraTrees to handle non-linear feature importance & reduce the variance.

=> 104 final features

Phase 2: Result

Models	CV RMSE	Actual Score est
Standard (No Target Transform)	0.513112	0.54086
Yeo-Johnson	0.455419	0.51613
Quantile	0.537756	N/A
Blend	0.499646	0.50852

Hypothesis: MSE assume the error terms follow normality assumption. If target y is skewed, the models may not capture the core signal.

Yeo-Johnson: Applies power-based transformation using a parameter λ, chosen to optimize normality. Works with both positive and negative values (unlike Box-Cox).

Quantile Transformation: Sorts and ranks data, then replaces each value with its quantile-based equivalent.

Blend: weighted average (model with lower error will weight higher)

λ	Transformed Data
-2	y^{-2}
-1	y^{-1}
-0.5	$1/\sqrt{y}$
0	ln(y)
0.5	$\sqrt{y}$
1	y
2	y^2

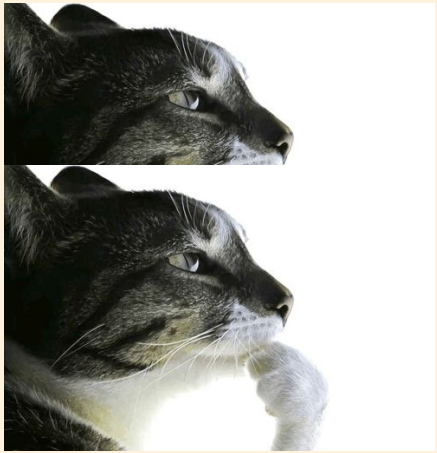
The Transition from Phase 2 to 3

Recap from phase 2:

- Model tuning couldn't solve the small dataset problem
- Tree based model needs lots of data

Intuition: What if there is a way to create artificial data to train the model

Phase 3 Components:
Model: Stacking Ensemble 3 base models (Level 0) + meta-learner (Level 1)
Feature Selection: 112 -> 34 (Dual Selection: Lasso + CatBoost)
Feature Engineering: Top 8 features for Interaction 34+28 = 62
Target Transformation: Quantile
Data Augmentation: Train via VRM



x2 Hmm

Phase 3: Model Stacking

Goal: Stack the best model to predict.

L0 Base Models:

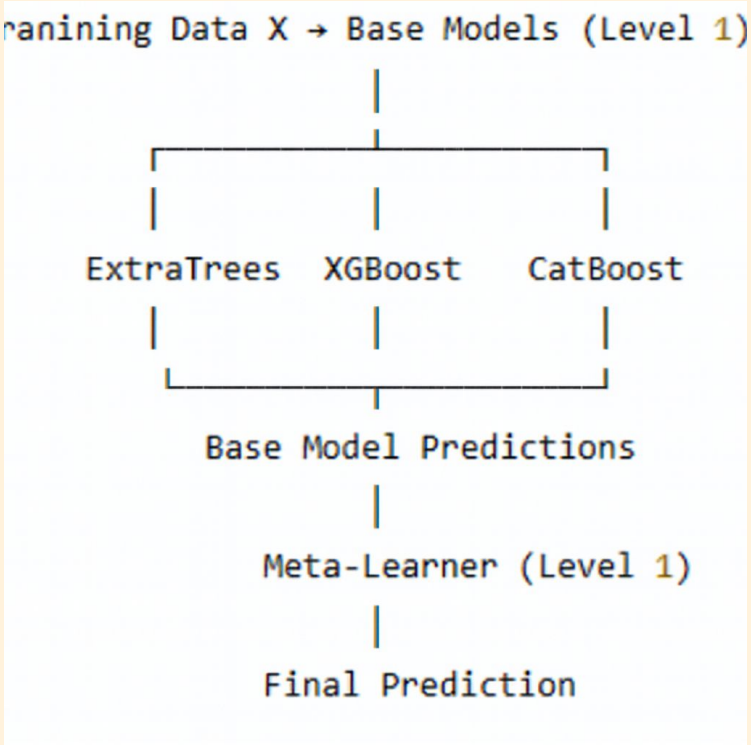
- ExtraTreesRegressor: random splits.
- XGBoost: reduce bias with regularization
- CatBoost: handle categorical + L2 regularization

L1 Meta Learner:

- Ridge Regression

CatBoost:

- Similar to Gradient Boost but designed to handle both categorical and numerical features.
- Great with large-scale datasets with many independent features



Phase 3: Feature Selection then engineering

```
class DualFeatureSelector:
    def __init__(self, n_lasso_features=25, n_tree_features=25):
        self.n_lasso = n_lasso_features
        self.n_tree = n_tree_features
```

Lasso (Linear Judge):

- Uses L1 penalty to shrink weak coefficients to zero.
- Captures linearity trends

CatBoost (Non-Linear Judge):

- Captures complex interactions

The Union:

- Take the union of both lists of Top 25 features.
- Result: 112 → 34 features

- **Why interactions:**
  - Polynomials, stats, clusters added noise.
  - Only create feature that the model is confident in predicting (xgboost, catboost)
- **Approach (similar in phase 2):**
  - Use ExtraTrees on 34 selected & rank by minimizing impurity features (MSE for ski-kit learn)
  - Select the top 8 most important features (if use all 34 => 561 new features => overfit)
- **Result:**
  - Created 28 interactions (8 choose 2)
  - Feature space: 34+28 = 62

```
self.selected_indices = np.unique(np.concatenate([lasso_indices, tree_indices]))
```

Phase 3: Target Transformation and VRM

In Phase 2: we used Yeo-Johnson

Why Quantile?

- The transformed data are exactly normal, adding Gaussian noise generates synthetic points that spread evenly across the latent space, filling gaps and covering the full range.

- Once the target is normalized, VRM simply creates synthetic targets by injecting random Gaussian noise into this transformed space

Assuming we applied Quantile transformation

Generate "synthetic neighbors" vis VRM:

y normal′ = y normal + ε, ε is random error (Gaussian).

Predictors (Xn): Feature-selected, engineered, scaled for outliers, then augmented by small random noise.
X’ = X + ε, ε is random error (Gaussian).

We get:
(y normal′ = y normal + ε, X’ = X + ε)

This makes the math behind the model correct!

Phase 3: VRM Example Code

```python
for i, (name, model) in enumerate(wrapped_models):
    print(f"  Training {name}...")
    fold_scores = []
    for fold, (train_idx, val_idx) in enumerate(kf.split(X_eng, y)):
        # Augment ONLY the training fold
        X_fold_train = X_eng[train_idx]
        y_fold_train = y[train_idx]
        X_fold_aug, y_fold_aug = augment_data(X_fold_train, y_fold_train, noise_level=NOISE_LEVEL,

        model.fit(X_fold_aug, y_fold_aug)
```

Critical Detail:
We augment only the training fold, not the validation fold.

We want to test on real, unseen data -> keep validation set untouched & unaugmented

Per-Fold Breakdown (10-Fold CV):
Original set: n = 300
Training split: n = 270, validation: n=30
VRM augmentation: n = 270*2 = 540

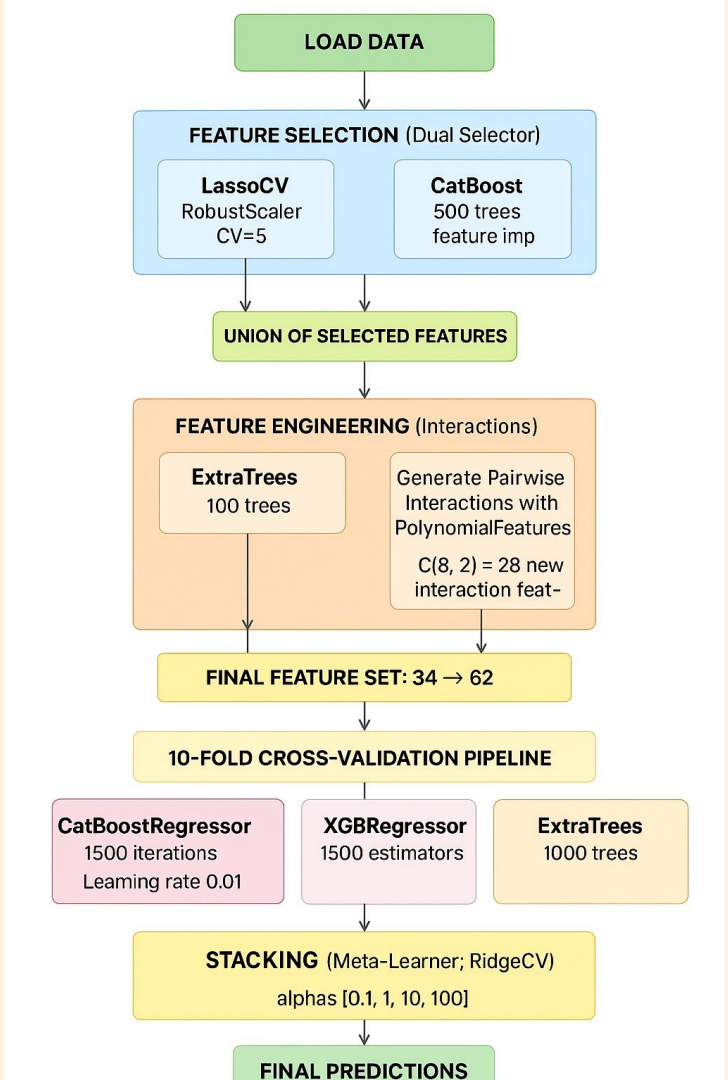
3 models × 10 folds = 30 model instances, and each sees 540 samples:

Total training: 30*540 = 16,200 sample combinations

Recap Final Model

Summary:

- Combined VRM with feature selection and ensemble models
- Capture **sparse linear signals** and **non-linear interactions**

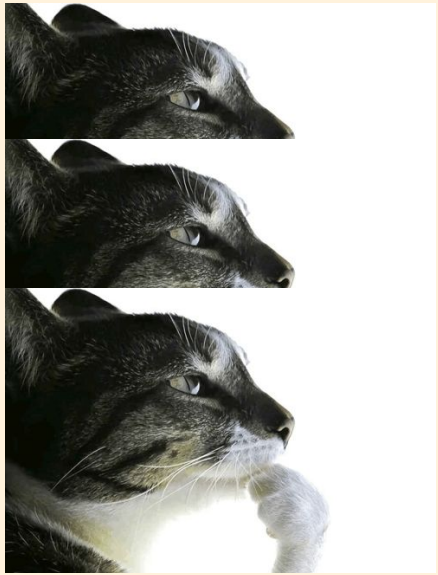


The key insight

Phase 2: Build a **brute force** pipeline + pick the best (intuitively)
Result: Hit a wall at RMSE ~0.51 because the data was too small.

Phase 3: Build refine model + train on **synthetic data**
Result: VRM effectively multiplying the dataset size
RMSE ~0.46

The key insight: Problem solved using data augmentation



x3 Hmm

What I learned?

1

Brute Force might not be bad!

By trying different models, CV, Folds, transformation, and different ensemble methodologies…

2

Problem Solving

We often don't have the luxury to submit and see scores for direction.

On the other hand, real world dataset would have domain-specific knowledge to understand root cause & capability of coming up with hypothesis

3

Software Engineer

Uses of packages, tools (scikit-learn, etc.)

CatBoost may not support the newest Python versions; without wheels, must use an older Python version or setting up build tools and dependencies.

Towards the future:

- further refine the statistical model
- apply deep learning technique

Thank You

Feedbacks & Discussion