RAPID: Regression Analysis Pipeline with Intelligent Data preprocessing

A comprehensive machine learning pipeline for regression analysis featuring automated data preprocessing, intelligent feature selection, and ensemble model comparison with stacking capabilities.

Overview

This project provides an end-to-end automated workflow for regression modeling that handles:

- **Intelligent Data Preprocessing**: Automated missing data imputation, outlier detection, and quality checks
- **Feature Selection**: Multiple reduction strategies (correlation-based, variance-based, tree-based importance)
- **Model Comparison**: Evaluates 10+ regression algorithms with hyperparameter optimization
- **Ensemble Methods**: Stacking regressors with multiple meta-learners for optimal performance
- **Export Capabilities**: Generates comprehensive Excel reports and publication-ready PNG plots

Key Features

- ☑ **"Hit and Walk Away" Automation** - Set target variable once and run entire pipeline
- ☑ **Configurable Constants** - 30+ parameters for complete workflow customization
- ☑ **Advanced Imputation** - Threshold-based strategies (simple → KNN → iterative)
- ☑ **Robust Validation** - K-Fold cross-validation with stratified sampling
- ☑ **Professional Reporting** - Multi-tab Excel exports with narrative insights
- ☑ **Presentation-Ready Outputs** - High-DPI PNG plots for immediate use

---

Table of Contents

Requirements

System Requirements

- **Python Version**: 3.10+ (Tested on 3.12.0) - Download Python 3.12.x

- **Operating System**: Windows, macOS, or Linux
- **RAM**: 16GB minimum (more recommended for large datasets)
- **Storage**: 500MB for dependencies + space for your data

Python Version Note

While developed with Python 3.12.0, this project should work with Python 3.10 or higher. If you encounter compatibility issues, we recommend using Python 3.12.x for best results.

🚀 Installation

1. Clone the Repository

```
git clone https://github.com/chemnteach/regression_modeling.git
cd regression_modeling
```

2. Create Virtual Environment

Windows (Command Prompt - Recommended):

If your terminal shows `PS` at the prompt (PowerShell), switch to Command Prompt first:

```
cmd
```

Then create and activate the virtual environment:

```
python -m venv venv
venv\Scripts\activate
```

You'll see `(venv)` appear in your prompt when activated.

To deactivate when done:

```
deactivate
```

macOS/Linux:

```
python3 -m venv venv
source venv/bin/activate
```

To deactivate when done:

```
deactivate
```

3. Install Dependencies

```
pip install --upgrade pip
pip install -r requirements.txt
```

Note: If you're behind a corporate proxy (like Intel's), use:

```
pip install --proxy http://proxy-dmz.intel.com:912 -r requirements.txt
```

4. Verify Installation

Open Jupyter Notebook and ensure all imports work:

```
jupyter notebook "Feature Reduction.ipynb"
```

⚡ Quick Start

Streamlined Workflow (Recommended)

1. Open `Feature Reduction.ipynb` in Jupyter/VS Code
2. Run the packages installation cells
3. **Execute the first data loading cell** - it will:
    - Prompt you to select your CSV file
    - Display all numeric columns with statistics
    - Ask you to select your target variable by number (no typos!)
    - Ask if you want to run stacking ensemble analysis
4. Run all remaining cells (or Kernel → Run All)
5. Find outputs in the project directory:
    - `feature_importance_scores.csv` - Feature rankings
    - `Feature_Analysis_Report_YYYYMMDD_HHMMSS.xlsx` - Comprehensive 3-tab report
    - `best_model_*.png` - High-resolution model plots (if plot-saving cell is run)
    - `*.pkl` - Trained model pipelines for deployment

Option 3: Standalone Script (Future Enhancement)

```
python main.py --data your_data.csv --target column_name
```

(Note: main.py currently serves as project entry point; full CLI support planned)

Project Structure

```
regression_modeling/
│
├── Feature Reduction.ipynb     # Main analysis notebook (3,100+ lines)
├── main.py                      # Project entry point / utilities
├── feature_reduction.py         # Core preprocessing functions
│
├── requirements.txt             # Python package dependencies
├── README.md                    # This file
│
└── venv/                        # Virtual environment (created during setup)
```

Key Files Explained

File	Purpose	Lines	Status
`Feature Reduction.ipynb`	Complete ML pipeline with documentation	3,300+	Primary
`feature_reduction.py`	Reusable preprocessing functions	-	Support
`main.py`	Project scaffolding / future CLI	-	Support
`requirements.txt`	All package dependencies	-	Active

Usage Guide

Workflow Overview

The notebook follows a 7-stage pipeline:

```
┌─────────────────────────────────────────────────────────┐
│ 1. SETUP ENVIRONMENT                                    │
│    → Install packages, import libraries, set constants  │
└─────────────────────────────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│ 2. LOAD DATA                                            │
│    → Interactive file selection or automatic CSV loading│
└─────────────────────────────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│ 3. INITIAL QUALITY CHECK                                │
│    → Remove columns with >40% missing data              │
│    → Remove date columns, duplicates, high cardinality  │
└─────────────────────────────────────────────────────────┘
                          ↓
```

```
┌─────────────────────────────────────────────────────────┐
│ 4. STRATEGIC IMPUTATION                                 │
│    → <5% missing: Median imputation                     │
│    → 5-20% missing: KNN imputation                      │
│    → 20-40% missing: Iterative imputation               │
└─────────────────────────────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│ 5. FEATURE SELECTION                                    │
│    → Correlation analysis, variance thresholds          │
│    → Tree-based feature importance                      │
└─────────────────────────────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│ 6. MODEL TRAINING & COMPARISON                          │
│    → Train 8 base models with cross-validation          │
│    → Hyperparameter optimization via RandomizedSearchCV │
│    → Evaluate using R² scoring                          │
└─────────────────────────────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│ 7. ENSEMBLE STACKING (OPTIONAL)                         │
│    → User prompted if not auto-enabled                  │
│    → Combine top models with stacking ensemble          │
│    → Test multiple meta-learners                        │
│    → Select best performing combination                 │
│    → Skipped entirely if user declines                  │
└─────────────────────────────────────────────────────────┘
                          ↓
┌─────────────────────────────────────────────────────────┐
│ 8. EXPORT RESULTS                                       │
│    → CSV: Feature importance scores                     │
│    → Excel: 3-tab comprehensive report                  │
│    → PNG: High-resolution model plots                   │
└─────────────────────────────────────────────────────────┘
```

First Cell Workflow

The first executable cell handles all initial setup interactively:

Step 1: CSV Selection

- Opens file dialog to select your CSV file
- Loads data and displays basic statistics

Step 2: Target Variable Selection

- Lists all numeric columns with:
  - Column name
  - Data type
  - Missing data percentage
- **Select by number** (e.g., enter "5" for the 5th column)

- Eliminates typos and ensures valid selection
- Selected column is **protected** from all automated cleaning
- **Automatically removes rows** with missing target values (cannot train models on missing targets)

Step 3: Stacking Analysis Option

- Asks if you want to run ensemble stacking after individual models
- **Choose "yes"** if:
  - Individual models have moderate R^2 (0.7-0.9)
  - You want maximum performance
  - Computational time is acceptable
- **Choose "no"** if:
  - An individual model achieves excellent R^2 (>0.95)
  - You prefer simpler, more interpretable models
  - Time is constrained

Conditional Stacking Behavior: If you choose "no" in the first cell, you'll be prompted again before the stacking section:

- Review individual model performance first
- Decide whether to continue based on actual results
- Enter "yes" or "no" to proceed or skip
- Stacking is completely skipped if declined, saving computation time

---

⚙ Configuration

Key Constants (Located in Constants Cell)

Missing Data Handling

```
MAX_MISSING_DATA = 0.4            # Remove columns with >40% missing (AUTOMATED)
LOW_MISSING_THRESHOLD = 0.05      # <5% missing → Simple imputation (median for
numeric, mode for categorical)
MEDIUM_MISSING_THRESHOLD = 0.20   # 5-20% missing → KNN imputation
HIGH_MISSING_THRESHOLD = 0.40     # 20-40% missing → Iterative imputation (MICE)
```

Automated Imputation Strategy:

- **< 5%**: Median for numeric, mode for categorical - Fast and effective
- **5-20%**: KNN imputation - Preserves local relationships
- **20-40%**: Iterative/MICE imputation - Advanced statistical modeling
- **> 40%**: Column automatically dropped (unless it's the protected target variable)

Data Quality Filters

```
REMOVE_DATE_COLUMNS = True         # Drop datetime columns
HIGH_CARDINALITY_THRESHOLD = 0.8   # Remove likely ID columns
REMOVE_DUPLICATE_ROWS = True       # Drop exact duplicate rows
LOW_VARIANCE_THRESHOLD = 0.99      # Remove near-constant columns
```

Model Training

```
TRAINING_DATA_SPLIT = 0.8          # 80% train, 20% validation
CROSS_VALIDATION_FOLDS = 5         # K-Fold CV splits
RANDOM_STATE = 42                  # Reproducibility seed
HYPERPARAMETER_ITERATIONS = 20     # RandomizedSearchCV iterations
```

Feature Selection

```
FEATURE_IMPORTANCE_THRESHOLD = 0.001  # Minimum importance to keep
CORRELATION_THRESHOLD = 0.95          # Remove highly correlated features
VARIANCE_THRESHOLD = 0.01             # Minimum variance required
```

Visualization

```
HISTOGRAM_BINS = 30                # Residual histogram bins
figSize = [10, 10]                 # Default plot dimensions
```

All 30+ Configurable Parameters

See the **Constants** cell in `Feature Reduction.ipynb` for the complete list with detailed comments.

🎯 Feature Selection Methodology (Detailed)

This section provides a comprehensive, step-by-step explanation of the feature selection process used in this pipeline. The methodology combines correlation analysis with ensemble-based importance ranking to identify the most predictive features while minimizing multicollinearity.

Overview: Two-Stage Feature Reduction

The pipeline uses a **two-stage approach** to feature selection:

1. **Stage 1 - Correlation-Based Reduction**: Removes highly correlated (redundant) features
2. **Stage 2 - Importance-Based Selection**: Identifies features with highest predictive power

This sequential approach ensures we keep only features that are both **unique** (non-redundant) and **predictive** (useful for modeling).

Stage 1: Correlation-Based Feature Reduction

Objective: Eliminate multicollinearity by removing redundant features that provide duplicate information.

Step 1.1: Data Preparation

```
working_data = model_ready_data.copy()
working_data[dependent_var] = working_data[dependent_var].astype('float64')
```

Actions:

- Create working copy of fully preprocessed data
- Ensure target variable is numeric (required for correlation analysis)
- Remove any remaining string columns (these should have been encoded earlier)
- Save snapshot of data before feature reduction for audit trail

Output: Clean numeric dataset ready for correlation analysis

Step 1.2: Calculate Correlation Matrix

```
corr_matrix = working_data.corr()
target_corr = corr_matrix[dependent_var].abs().drop(dependent_var)
```

Actions:

- Compute pairwise Pearson correlation coefficients for all features
- Calculate absolute correlation between each feature and target variable
- Use absolute values because both +1 and -1 indicate strong relationships

Technical Details:

- Pearson correlation coefficient range: -1.0 to +1.0
- Values near ±1.0 indicate strong linear relationships
- Values near 0.0 indicate weak/no linear relationship

Step 1.3: Identify Highly Correlated Feature Pairs

```
CORRELATION_THRESHOLD = 0.95  # Configurable constant
```

Algorithm:

```
FOR each feature pair (i, j):
    IF abs(correlation) >= CORRELATION_THRESHOLD:
        IF target_corr[feature_i] > target_corr[feature_j]:
            Mark feature_j for removal
        ELSE:
            Mark feature_i for removal
```

Logic:

- When two features have correlation ≥ 0.95, they provide redundant information
- Keep the feature with **stronger correlation to the target**
- Drop the feature with weaker target correlation
- This preserves maximum predictive information while reducing redundancy

Example:

- Feature A and Feature B have 0.97 correlation (highly redundant)
- Feature A has 0.65 correlation with target
- Feature B has 0.48 correlation with target
- **Action**: Keep Feature A, drop Feature B

Step 1.4: Remove Redundant Features

```
working_data_reduced = working_data.drop(columns=to_drop)
```

Actions:

- Drop all features marked for removal
- Preserve target variable and all non-redundant features
- Log which features were dropped and why

Typical Results:

- 10-30% of features removed (dataset dependent)
- Remaining features are functionally independent
- No loss of unique predictive information

Stage 2: Importance-Based Feature Selection

Objective: Rank remaining features by predictive power and select the most important ones.

Step 2.1: Create Training/Validation Split

```
X_train, X_validation, Y_train, Y_validation = train_test_split(
    independent, dependent,
```

```
    test_size=VALIDATION_SIZE,  # Typically 0.2
    random_state=SEED
)
```

Actions:

- Separate features (X) from target (Y)
- Split into 80% training, 20% validation
- Use fixed random seed for reproducibility
- Validation set held out for final model evaluation

Why This Matters:

- Prevents overfitting during feature selection
- Ensures feature importance scores are unbiased
- Training set used for all selection decisions
- Validation set only used for final performance measurement

Step 2.2: Train Multiple Feature Importance Models

Model Suite: 4 tree-based algorithms with built-in feature importance

```
feature_importance_models = [
    ('RandomForest', RandomForestRegressor(n_estimators=100, random_state=SEED)),
    ('XGBoost', XGBRegressor(n_estimators=100, random_state=SEED)),
    ('GradientBoosting', GradientBoostingRegressor(n_estimators=100,
random_state=SEED)),
    ('CatBoost', CatBoostRegressor(iterations=100, random_seed=SEED,
verbose=False))
]
```

Why Multiple Models?

- **Random Forest**: Importance based on mean decrease in impurity across trees
- **XGBoost**: Gain-based importance with L1/L2 regularization effects
- **Gradient Boosting**: Sequential importance based on loss reduction
- **CatBoost**: Handles categorical features differently, may identify different patterns

Different algorithms use different internal mechanics, so they may disagree on which features are most important. Using multiple models captures a **consensus view**.

Step 2.3: Extract and Normalize Feature Importances

```
for name, model in feature_importance_models:
    pipeline = Pipeline([
        ('scaler', StandardScaler()),
```

```
        ('model', model)
    ])
    pipeline.fit(X_train, Y_train)

    # Extract raw importances
    importance = pipeline.named_steps['model'].feature_importances_

    # Normalize to sum to 1.0 (makes models comparable)
    normalized_importances = importance / np.sum(importance)
```

Process:

1. Standardize features (zero mean, unit variance)
2. Train model on training data
3. Extract feature importances from trained model
4. Normalize importances to sum to 1.0

Why Normalize?

- Makes importance scores comparable across different models
- Raw importance scales vary between algorithms
- Normalization creates a common 0-1 scale

Step 2.4: Calculate Feature Ranks

```
ranks = np.argsort(np.argsort(-normalized_importances)) + 1
```

Rank Calculation:

- Rank 1 = Most important feature
- Rank N = Least important feature
- Ties broken by array order

Example:

```
Feature A: Importance 0.35 → Rank 1
Feature B: Importance 0.28 → Rank 2
Feature C: Importance 0.20 → Rank 3
Feature D: Importance 0.17 → Rank 4
```

Why Use Ranks Instead of Raw Scores?

- Robust to outliers (one model giving extreme importance)
- Less sensitive to scale differences between algorithms
- Easier to interpret (percentile-based)
- Statistical theory supports rank aggregation

Step 2.5: Aggregate Rankings Across Models

```
for feature in features:
    mean_rank = np.mean(feature_ranks[feature])
    mean_importance = np.mean(all_importances[feature])
    std_rank = np.std(feature_ranks[feature])
```

Calculations:

- **Mean Rank**: Average rank across all 4 models (lower = better)
- **Mean Importance**: Average normalized importance score
- **Rank Std Dev**: Measures consistency (low std = consistent across models)

Example:

```
Feature "Temperature":
  - RandomForest rank: 2
  - XGBoost rank: 1
  - GradientBoosting rank: 3
  - CatBoost rank: 2
  → Mean Rank: 2.0 (very important, consistent)

Feature "NoiseColumn":
  - RandomForest rank: 45
  - XGBoost rank: 38
  - GradientBoosting rank: 50
  - CatBoost rank: 42
  → Mean Rank: 43.75 (low importance, consistent)
```

Step 2.6: Select Features Using Cumulative Importance

```
cumulative_threshold = 0.95  # Keep features accounting for 95% of importance

cumulative_importance = 0
selected_features = []

for feat_stat in sorted_by_importance:
    cumulative_importance += feat_stat['mean_importance']
    selected_features.append(feat_stat)

    if cumulative_importance / total_importance >= cumulative_threshold:
        break

# Safety net: Keep at least 20% of features
min_features = max(int(np.ceil(num_features * 0.20)), 10)
```

```
if len(selected_features) < min_features:
    selected_features = feature_statistics[:min_features]
```

Selection Strategy:

1. Sort features by mean importance (highest first)
2. Add features to selection until cumulative importance reaches 95%
3. Enforce minimum of 20% of original features (or 10, whichever is larger)

Why Cumulative Importance?

- **Adaptive**: Selection adapts to actual importance distribution
- **No Arbitrary Cutoffs**: Doesn't assume "top 50%" is always right
- **Comprehensive**: Captures all meaningfully predictive features
- **Safe**: Minimum threshold prevents over-aggressive reduction

Example Scenario:

```
Starting with 100 features after correlation reduction:

Feature 1-10:   Cumulative importance = 75%
Feature 11-15:  Cumulative importance = 90%
Feature 16-18:  Cumulative importance = 95% ← STOP HERE

Result: 18 features selected (18% of original)
Remaining 82 features contribute only 5% to predictions
```

Step 2.7: Create Final Feature Set

```
feature_names = [f['feature'] for f in selected_features]
independent = working_data_reduced[feature_names]
dependent = working_data_reduced[dependent_var]
```

Actions:

- Extract feature names from selected statistics
- Create final modeling dataset with selected features only
- Preserve target variable
- Save feature importance scores to CSV for documentation

Output: Feature Importance Report

CSV File: `feature_importance_scores.csv`

Columns Explained:

Column	Description	Value Range	Interpretation
feature	Feature name exactly as it appears in the dataset	String	Identifies which variable this row describes
mean_rank	Average rank across 4 tree-based models (RF, XGB, GBT, CB)	1.0 to N	**Lower is better**. Rank 1 = most important feature. If a feature ranks 2, 1, 3, 2 across the 4 models, mean_rank = 2.0
std_rank	Standard deviation of ranks across the 4 models	≥ 0.0	**Lower = more consensus**. Low std_rank (< 2.0) means all models agree this feature is important/unimportant. High std_rank (> 5.0) means models disagree
mean_importance	Average normalized importance score across 4 models	0.0 to 1.0	**Higher is better**. Sum of all features ≈ 1.0. A score of 0.15 means this feature accounts for 15% of total predictive power
rank_percentile	Percentile ranking within the selected feature set	0% to 100%	**Higher is better**. 100% = best feature, 50% = median feature, 0% = worst feature in selected set. Calculated as: (1 - mean_rank/N) × 100

Use Cases:

- Identify top predictive features for business insights
- Validate feature engineering decisions
- Explain model predictions to stakeholders
- Guide future data collection priorities

Summary: Complete Feature Selection Workflow

```
STAGE 1: Correlation Reduction
├─ 1. Calculate pairwise correlations
├─ 2. Identify pairs with |correlation| ≥ 0.95
├─ 3. For each pair, keep feature with stronger target correlation
└─ 4. Drop redundant features
   → Result: Multicollinearity eliminated

STAGE 2: Importance Selection
├─ 1. Split data (80% train, 20% validation)
├─ 2. Train 4 tree-based models (RF, XGB, GBT, CB)
├─ 3. Extract and normalize feature importances
├─ 4. Calculate ranks for each feature in each model
├─ 5. Average ranks across all models
├─ 6. Sort by mean importance, select top features
├─ 7. Keep features accounting for 95% cumulative importance
```

```
  └─ 8. Enforce minimum 20% retention safety net
     → Result: Optimal predictive feature set

FINAL OUTPUT
└─ Reduced feature set: Unique + Predictive + Documented
```

Advantages of This Methodology

- ☑ **Reduces Bias**: Multiple models prevent any single algorithm from dominating selection
- ☑ **Eliminates Redundancy**: Correlation analysis removes duplicate information
- ☑ **Preserves Information**: Keeps features with unique predictive value
- ☑ **Robust to Outliers**: Rank-based aggregation handles extreme importance values
- ☑ **Adaptive**: Cumulative importance adjusts to actual data patterns
- ☑ **Interpretable**: Mean ranks are easy to understand and explain
- ☑ **Reproducible**: Fixed random seeds ensure consistent results
- ☑ **Safe**: Minimum thresholds prevent over-reduction

📊 Output Files

1. Feature Importance CSV

Filename: `feature_importance_scores.csv`

Contains ranked features with comprehensive statistics from ensemble-based feature selection.

Columns:

Column	Typical Values	What It Tells You
feature	Column names from your dataset	Which variable this row describes
mean_rank	1.0 to N (lower is better)	How consistently important this feature is. A mean_rank of 2.0 means it averaged 2nd place across 4 models
std_rank	0.0 to ~10.0 (lower is better)	How much models agree. std_rank < 2 = strong consensus, > 5 = disagreement
mean_importance	0.0 to 1.0 (higher is better)	Percentage of predictive power. 0.15 = this feature drives 15% of predictions
rank_percentile	0% to 100% (higher is better)	Where this feature ranks overall. 95th percentile = top 5% of features

How to Read the File:

- **Sort by mean_rank** (ascending): Find the most important features
- **Filter std_rank < 2**: Find features all models agree on
- **Sum mean_importance** for top N features: See how much predictive power they capture

- **Filter rank_percentile > 90**: Focus on top-tier features only

Use Cases:

- Identify top predictive features for business insights
- Validate feature engineering decisions
- Focus data quality efforts on high-importance features
- Explain model behavior to stakeholders

---

2. Excel Comprehensive Report

Filename: `Feature_Analysis_Report_YYYYMMDD_HHMMSS.xlsx`

Four-tab workbook with comprehensive analysis:

Tab 1: Narrative

- Executive summary
- Methodology overview
- Model performance highlights
- Key insights and patterns
- Data quality assessment
- Actionable recommendations

Tab 2: Feature Metrics

- All features ranked by importance
- Complete statistical profiles
- Missing data analysis
- Data type classification

Tab 3: Model Comparison

- All models ranked by R^2 score
- Base learners + stacking ensembles
- Performance deltas
- Model selection rationale

Tab 4: Imputation Log

- Columns that underwent missing data imputation
- Method used (Simple/KNN/Iterative)
- Original missing data percentage
- Data type and threshold category
- Empty if no imputation tracking available

Use Case: Client presentations, stakeholder reports, documentation, data quality audits

---

3. Model Performance Plots (PNG)

Location: `data/` folder
Filenames: `<ModelName>_<plot_type>_YYYYMMDD_HHMMSS.png`

Three high-resolution (300 DPI) plots generated for each model and stacking ensemble:

1. **Predicted vs Actual**: Scatter plot with perfect prediction line (`*_predicted_vs_actual_*.png`)
2. **Residuals CDF/PDF**: Distribution analysis with 95% threshold (`*_cdf_pdf_residuals_*.png`)
3. **Residuals Distribution**: Histogram with KDE overlay (`*_residual_distribution_*.png`)

Timestamp Format: YYYYMMDD_HHMMSS ensures unique filenames for each run

Use Case: Presentations, publications, model validation reports, performance tracking over time

---

4. Dataset Snapshots (CSV)

Location: `data/` folder
Filenames: Timestamped CSV files at key pipeline stages

- `final_modeling_data_YYYYMMDD_HHMMSS.csv` - Before feature reduction
- `final_modeling_data_post_feature_selection_YYYYMMDD_HHMMSS.csv` - After feature selection

Use Case: Audit trail, reproducibility, data quality verification, intermediate analysis

---

Supported Models

Base Learners (8 Algorithms)

Category	Models
Tree-Based	Random Forest, Extra Trees, Gradient Boosting
Boosting	XGBoost, LightGBM, CatBoost
Instance-Based	KNN Regressor
Ensemble	Bagging Regressor

Meta-Learners (Stacking)

- All 8 base learners
- Linear Regression

Model Selection Criteria

- **Primary Metric**: R^2 score (coefficient of determination)
- **Validation**: 5-fold cross-validation
- **Optimization**: RandomizedSearchCV with 20 iterations
- **Selection**: Top 5 base models → Stacking → Best meta-learner

---

Troubleshooting

Common Issues

Issue: Excel export fails with "name 'feature_statistics' is not defined"

Cause: Kernel was restarted and variables lost

Solution:

1. Re-run entire notebook (Kernel → Restart & Run All)
2. The Excel report is automatically generated in the final cell

---

Issue: "Behind corporate proxy" - pip install fails

Solution: Use proxy flag:

```
pip install --proxy http://your-proxy:port -r requirements.txt
```

For Intel networks: `http://proxy-dmz.intel.com:912`

Performance Optimization

For Large Datasets (>1M rows):

- Reduce `CROSS_VALIDATION_FOLDS` from 5 to 3
- Decrease `HYPERPARAMETER_ITERATIONS` from 20 to 10
- Consider using only fastest models (Random Forest, Extra Trees) for initial exploration

For High-Dimensional Data (>100 features):

- Tighten `CORRELATION_THRESHOLD` to 0.90
- Increase `FEATURE_IMPORTANCE_THRESHOLD` to 0.005
- Enable aggressive variance filtering

Memory Issues:

- Use `TRAINING_DATA_SPLIT = 0.7` for smaller training set
- Remove ensemble models and use best base learner only
- Process data in chunks if possible

Contributing

Reporting Issues

Submit bug reports or feature requests via GitHub Issues.

Include:

- Python version
- Full error traceback
- Minimal reproducible example
- Dataset characteristics (rows, columns, missing data %)

Pull Requests

1. Fork the repository
2. Create feature branch: `git checkout -b feature/your-feature-name`
3. Follow PEP 8 style guidelines
4. Add docstrings (Google/NumPy style)
5. Test with sample data
6. Submit PR with clear description

License

This project is provided as-is for educational and commercial use. See repository for specific license terms.

Contact

Author: Craig Brown
Organization: Intel Corporation
Email: craig.d.brown@intel.com
Repository: github.com/chemnteach/regression_modeling

Acknowledgments

Built with:

- scikit-learn ecosystem
- XGBoost, LightGBM, CatBoost
- pandas, NumPy, seaborn
- Jupyter/VS Code notebook environment

Additional Resources

Learning Materials

- scikit-learn Documentation
- Ensemble Methods Guide
- Feature Selection Strategies

Related Projects

- `Vmin Kitchen Sink.ipynb` - Extended analysis examples

- `feature_reduction.py` - Reusable function library

Last Updated: November 24, 2025
Version: 1.0
Status: Production Ready ☑