RAPID: Regression Analysis Pipeline with Intelligent Data Preprocessing

Version 2.0 | PEP 8 Compliant | Production Ready

A professional-grade machine learning pipeline for regression analysis featuring modular configuration, comprehensive logging, type-safe code, and automated preprocessing with intelligent feature selection.

🎯 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**: Optimized stacking regressors (automatically runs) with multiple meta-learners
- **Export Capabilities**: Generates comprehensive Excel reports, JSON feature lists, and publication-ready PNG plots

Key Features

✅ **"Hit and Walk Away" Automation** - Set target variable once and run entire pipeline
✅ **Centralized Configuration** - All constants managed in `config.py` module
✅ **Professional Logging** - Dual-handler system (file + console) with timestamp tracking
✅ **Type Safety** - Complete type hints on all refactored functions
✅ **PEP 8 Compliant** - Industry-standard code quality and formatting
✅ **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!)
4. Run all remaining cells (or Kernel → Run All)
   - Stacking ensemble analysis runs automatically (optimized for ~1 hour runtime)
5. Find outputs in the project directory:
   - `feature_importance_scores.csv` - Feature rankings
   - `Feature_Analysis_Report_YYYYMMDD_HHMMSS.xlsx` - Comprehensive 5-tab report
   - `model_features_YYYYMMDD_HHMMSS.json` - Feature list for deployment
   - `logs/feature_reduction_YYYYMMDD_HHMMSS.log` - Pipeline execution log

   - `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 (4,000+ lines, refactored)
├── config.py                   # ⭐ NEW: Centralized configuration constants
├── excel_reporter.py           # ⭐ NEW: Consolidated Excel report generation
├── main.py                     # Project entry point / utilities
├── feature_reduction.py        # Core preprocessing functions (legacy)
├── display_features.py         # Feature display utility
│
├── requirements.txt            # Python package dependencies
├── old_requirements.txt        # Previous dependency snapshot
├── README.md                   # This documentation file
├── REFACTORING_SUMMARY.md      # ⭐ NEW: Complete refactoring changelog
├── VALIDATION_CHECKLIST.md     # ⭐ NEW: Final validation report
├── PARALLELIZATION_GUIDE.md    # ⭐ NEW: Parallel processing guide
├── GPU_ACCELERATION_GUIDE.md   # ⭐ NEW: GPU acceleration setup & usage
│
├── logs/                       # Pipeline execution logs (timestamped)
├── data/                       # Output datasets and snapshots
├── catboost_info/             # CatBoost training metadata
│
└── venv/                       # Virtual environment (created during setup)
```

Key Files Explained

File	Purpose	Lines	Status
`Feature Reduction.ipynb`	Complete ML pipeline with type hints & logging	4,000+	**Primary**
`config.py`	All configuration constants (thresholds, paths, settings)	261	**Required**
`excel_reporter.py`	Unified Excel report generation with type hints	~200	**Active**
`feature_reduction.py`	Reusable preprocessing functions (legacy support)	-	Support

File	Purpose	Lines	Status
`display_features.py`	Feature visualization utility	-	Support
`main.py`	Project scaffolding / future CLI	-	Support
`requirements.txt`	All package dependencies	-	**Active**
`REFACTORING_SUMMARY.md`	Documentation of all code improvements	-	Docs
`VALIDATION_CHECKLIST.md`	Final validation and quality metrics	-	Docs

📖 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 (AUTOMATIC)                             │
│    → Combine top models with optimized stacking ensemble     │
│    → Test multiple meta-learners                             │
│    → Select best performing combination                      │
│    → Runs automatically (~1 hour with optimizations)         │
└──────────────────────────────────────────────────────────────┘
                          ↓
┌──────────────────────────────────────────────────────────────┐
│ 8. EXPORT RESULTS                                            │
│    → CSV: Feature importance scores                          │
│    → Excel: 5-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: Automatic Stacking Ensemble

- Stacking ensemble analysis **runs automatically** after individual models
- Combines top-performing models to maximize predictive performance
- Uses optimized approach that completes in ~1 hour (8x faster than standard)
- Benefits:
  - Generates out-of-fold predictions from base learners once
  - Tests 9 different meta-learners on pre-generated predictions
  - No redundant model retraining (89 operations vs 720)
  - Always produces best possible ensemble performance

⚙️ Configuration

Centralized Configuration System

All configuration is now managed in `config.py` - a dedicated module for easy customization without editing notebook code.

Key Configuration Groups

Hardware & Performance (Auto-Detected)

```
# In config.py - Automatically configured based on your system
N_JOBS = 28                         # Parallel CPU cores (auto-detected: 2-28)
HYPERPARAM_SEARCH_ITER = 100        # Hyperparameter iterations (10-100 based on hardware)
USE_GPU = True                      # GPU availability (auto-detected)
XGBOOST_TREE_METHOD = 'gpu_hist'    # XGBoost acceleration method
LIGHTGBM_DEVICE = 'gpu'             # LightGBM device
CATBOOST_TASK_TYPE = 'GPU'          # CatBoost task type
```

Hardware Profiles (Auto-Detected):

- **LAPTOP** (<20 GB RAM): 2-4 jobs, 10 iterations, conservative memory usage
- **WORKSTATION** (20-64 GB RAM): 6-12 jobs, 20 iterations, balanced approach
- **SERVER** (64+ GB RAM): 16-28 jobs, 100 iterations, aggressive parallelization

GPU Acceleration:

- Automatically detects CUDA-capable GPUs via PyTorch or XGBoost
- Provides **2-6x speedup** for XGBoost, LightGBM, and CatBoost
- Gracefully falls back to CPU if GPU not available
- See `GPU_ACCELERATION_GUIDE.md` for details

Missing Data Handling

```
# In config.py
MAX_MISSING_DATA = 0.40          # Remove columns with >40% missing (AUTOMATED)
LOW_MISSING_THRESHOLD = 0.05     # <5% missing → Simple imputation
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)

Categorical Encoding Thresholds

```
ONE_HOT_ENCODING_MAX_CATEGORIES = 10   # Max categories for one-hot encoding
LABEL_ENCODING_MAX_CATEGORIES = 50     # Max categories for label encoding
HIGH_CARDINALITY_THRESHOLD = 0.95      # Threshold for ID detection
```

Data Quality Filters

```
REMOVE_DATE_COLUMNS = True          # Drop datetime columns
REMOVE_DUPLICATE_ROWS = True        # Drop exact duplicate rows
REMOVE_LOW_VARIANCE_COLS = True     # Remove near-constant columns
LOW_VARIANCE_THRESHOLD = 0.01       # Minimum variance required (1%)
```

Machine Learning Parameters

```
RANDOM_STATE = 42                   # Reproducibility seed
TEST_SIZE = 0.20                    # 20% validation split
CV_FOLDS = 5                        # K-Fold cross-validation splits
MAX_FEATURES = None                 # Feature limit (None = no limit)
```

Feature Selection Thresholds

```
CORRELATION_THRESHOLD = 0.10        # Minimum correlation for importance (10%)
LOW_VARIANCE_THRESHOLD = 0.01       # Minimum variance to retain feature
```

Logging Configuration

```
FILE_LOG_LEVEL = 'DEBUG'            # File logging detail level
CONSOLE_LOG_LEVEL = 'INFO'          # Console output level
LOG_DIR = 'logs'                    # Log file directory
LOG_FORMAT_FILE = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
```

Directory Paths

```
LOG_DIR = 'logs'                    # Execution logs
DATA_DIR = 'data'                   # Output datasets
FIGURES_DIR = 'figures'             # Generated plots
```

Configuration Benefits

☑ **Single Source of Truth** - All settings in one file
☑ **No Code Edits** - Modify behavior without touching notebook
☑ **Validation** - Built-in consistency checks on import
☑ **Documentation** - Comprehensive inline comments
☑ **Type Safety** - Constants properly typed for IDE support

Modifying Configuration

1. Open `config.py` in your editor
2. Locate the setting group you want to modify
3. Change the value (keeping the same data type)
4. Save the file
5. Restart your notebook kernel to load new values

Example: To be more aggressive with feature reduction:

```
# In config.py, change:
CORRELATION_THRESHOLD = 0.10  # Original

# To:
CORRELATION_THRESHOLD = 0.05  # More selective (5% minimum)
```

All 30+ Configurable Parameters

See `config.py` for the complete list with detailed comments and validation logic.

🎯 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

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`

Five-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

5. Feature List JSON

Filename: `model_features_YYYYMMDD_HHMMSS.json`

Machine-readable feature list for model deployment and validation.

Contents:

```json
{
  "generated": "2025-01-15 14:30:45",
  "target_variable": "Resistance",
```

```
    "feature_count": 18,
    "features": ["Feature1", "Feature2", ...],
    "best_model": "ExtraTreesRegressor",
    "best_r2": 0.9234
  }
```

Use Case: Model deployment validation, feature consistency checks, CI/CD pipelines, API integration

---

6. Pipeline Logs ★ NEW

Location: `logs/` folder
Filename: `feature_reduction_YYYYMMDD_HHMMSS.log`

Professional dual-handler logging system providing comprehensive audit trails for debugging, compliance, and reproducibility.

Logging Architecture

- **File Handler**: DEBUG level - Complete diagnostic information
- **Console Handler**: INFO level - User-friendly progress updates
- **Timestamp Format**: YYYY-MM-DD HH:MM:SS
- **UTF-8 Encoding**: International character support
- **Automatic Directory Creation**: `logs/` folder created if missing

Log Levels

Level	Description	Written To
DEBUG	Detailed diagnostic info (parameters, feature lists, intermediate values)	File only
INFO	Key pipeline events (data loaded, models trained, checkpoints)	File + Console
WARNING	Non-critical issues (missing optional data, fallback behavior)	File + Console
ERROR	Failures requiring attention (file errors, invalid data)	File + Console
CRITICAL	System-level failures (out of memory, corrupt files)	File + Console

Log Format

```
2025-12-01 14:30:45 - FeatureReduction - INFO - RAPID Pipeline Initialized
2025-12-01 14:30:45 - FeatureReduction - INFO - Log file:
logs/feature_reduction_20251201_143045.log
2025-12-01 14:30:45 - FeatureReduction - DEBUG - Configuration loaded:
MAX_MISSING_DATA=0.40
2025-12-01 14:30:47 - FeatureReduction - INFO - Data loaded: 1,234 rows × 56
columns
2025-12-01 14:30:47 - FeatureReduction - INFO - Target variable: Resistance
2025-12-01 14:31:02 - FeatureReduction - INFO - Feature selection complete: 18
features selected
```

```
2025-12-01 14:45:23 - FeatureReduction - INFO - RandomForest: R² = 0.9242
(optimized)
2025-12-01 15:30:45 - FeatureReduction - INFO - Best stacking meta-learner: ETR
with R² = 0.9587
2025-12-01 15:30:46 - FeatureReduction - INFO - PIPELINE COMPLETED SUCCESSFULLY
```

Key Events Logged

- **Session Initialization**: Start time, configuration summary, environment
- **Data Operations**: File paths, row/column counts, target variable selection
- **Configuration Values**: All thresholds and parameters at DEBUG level
- **Data Quality**: Missing data analysis, duplicate detection, variance checks
- **Feature Selection**: Number of features before/after each reduction stage
- **Model Training**: Each model's performance with cross-validation scores
- **Hyperparameter Tuning**: Best parameters found for each model
- **Stacking Results**: Meta-learner selection and final ensemble performance
- **Report Generation**: Excel/CSV/JSON export confirmations
- **Errors and Warnings**: Full stack traces with context

Configuration in config.py

```
FILE_LOG_LEVEL = 'DEBUG'          # Control file detail level
CONSOLE_LOG_LEVEL = 'INFO'        # Control console verbosity
LOG_DIR = 'logs'                  # Output directory
LOG_FORMAT_FILE = '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
LOG_FORMAT_CONSOLE = '%(levelname)s: %(message)s'  # Simpler console format
LOG_DATE_FORMAT = '%Y-%m-%d %H:%M:%S'
```

Use Case: Debugging failed runs, compliance audits, performance tracking, reproducing results, troubleshooting configuration issues, monitoring production pipelines "best_model": "ExtraTreesRegressor", "best_r2": 0.9234 }

```


**Use Case**: Model deployment validation, feature consistency checks, CI/CD
pipelines, API integration

---

### 6. Pipeline Logs
**Location**: `logs/` folder
**Filename**: `feature_reduction_YYYYMMDD_HHMMSS.log`

Comprehensive logging of all pipeline stages for debugging, auditing, and
reproducibility.

**Log Levels**:
| Level | Description | Written To |
```

```
|-------|-------------|------------|
| **DEBUG** | Detailed diagnostic info (parameters, feature lists) | File only |
| **INFO** | Key pipeline events (data loaded, models trained) | File + Console |
| **WARNING** | Non-critical issues (missing optional data) | File + Console |
| **ERROR** | Failures requiring attention | File + Console |

**Log Format**:
```

2025-11-30 14:30:45 | INFO | Feature Reduction Pipeline Started 2025-11-30 14:30:45 | INFO | Data loaded: 1,234 rows × 56 columns 2025-11-30 14:30:45 | DEBUG | Configuration: CORRELATION_THRESHOLD=0.95, CUTOFF_R2=0.5 2025-11-30 14:31:02 | INFO | Feature selection complete: 18 features selected 2025-11-30 14:45:23 | INFO | Hyperparameter optimization - RF: R² = 0.9242 (optimized) 2025-11-30 15:30:45 | INFO | Best stacking meta-learner: ETR with R² = 0.9587 2025-11-30 15:30:46 | INFO | PIPELINE COMPLETED SUCCESSFULLY

```


**Key Events Logged**:
- Pipeline start/end timestamps
- Data loading (rows, columns, target variable)
- Configuration constants
- Feature selection results
- Model optimization scores
- Stacking ensemble results
- Report generation status
- Errors with full context

**Use Case**: Debugging failed runs, audit trails, performance tracking,
reproducing results

---

## 🤖 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**: R² score (coefficient of determination)
- **Validation**: 5-fold cross-validation
- **Optimization**: RandomizedSearchCV with 20 iterations
```

```
- **Selection**: Top 5 base models → Automatic 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:
```bash
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.

🔄 Recent Updates (Version 2.0)

Major Refactoring (December 2025)

☑ Configuration Management

- **NEW**: `config.py` module with all constants centralized
- Extracted 30+ magic numbers to named constants
- Built-in validation for logical consistency
- Configuration display helper function

☑ Logging Infrastructure

- **NEW**: Professional dual-handler logging system
- File logs at DEBUG level with complete diagnostic info
- Console logs at INFO level for user progress
- Timestamp-based log filenames for audit trails
- UTF-8 encoding support for international characters

☑ Code Quality (PEP 8 Compliance)

- All imports reorganized by category (stdlib → third-party → sklearn)
- Added comprehensive type hints to 27 refactored functions
- Standardized naming conventions (UPPER_CASE constants, snake_case functions)
- Updated docstrings to NumPy/Google style format
- Removed all magic numbers from code

☑ Module Consolidation

- **NEW**: `excel_reporter.py` - Unified Excel report generation
- Replaced 3 duplicate scripts with single module
- All functions include type hints and proper documentation
- Fixed hardcoded file paths in `display_features.py`

☑ Documentation

- **NEW**: `REFACTORING_SUMMARY.md` - Complete change log
- **NEW**: `VALIDATION_CHECKLIST.md` - Quality metrics and validation
- Updated README with current architecture
- Consistent markdown headers throughout notebook

☑ Type Safety

- 100% type hint coverage on refactored functions
- Proper imports: `from typing import Optional, List, Dict, Tuple, Any`
- Return types specified for all functions
- Parameter types documented

☑ Performance Optimization

- **NEW**: Auto-adaptive parallel processing based on hardware detection
- Laptop (16 GB): 2-4 jobs, conservative memory usage
- Workstation (32-64 GB): 6-12 jobs, balanced approach
- Server (128+ GB): 16-28 jobs, aggressive parallelization
- **NEW**: GPU acceleration support with auto-detection
- 2-6x speedup for XGBoost, LightGBM, CatBoost on CUDA GPUs
- Graceful CPU fallback when GPU not available
- See `PARALLELIZATION_GUIDE.md` and `GPU_ACCELERATION_GUIDE.md`

Backward Compatibility

All original functionality preserved. Existing workflows continue to work without modification.

📚 Additional Resources

Project Documentation

- `REFACTORING_SUMMARY.md` - Detailed changelog and improvements
- `VALIDATION_CHECKLIST.md` - Quality assurance report
- `PARALLELIZATION_GUIDE.md` - Hardware-adaptive parallel processing guide
- `GPU_ACCELERATION_GUIDE.md` - GPU setup, troubleshooting, and performance
- `config.py` - Configuration reference with inline comments

Learning Materials

- scikit-learn Documentation
- Ensemble Methods Guide

- Feature Selection Strategies
- PEP 8 Style Guide
- Python Type Hints

Related Notebooks

- `Vmin Kitchen Sink.ipynb` - Extended analysis examples
- `Feature Reduction.ipynb` - Main pipeline (refactored)

Utility Modules

- `config.py` - Centralized configuration
- `excel_reporter.py` - Report generation utilities
- `feature_reduction.py` - Legacy preprocessing functions (still supported)
- `display_features.py` - Feature visualization

Last Updated: December 1, 2025
Version: 2.0
Status: Production Ready ☑
Code Quality: PEP 8 Compliant | Type-Safe | Professionally Logged

- 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: January 15, 2025
Version: 1.1
Status: Production Ready ☑