SmartExplain AI – Project Details

Interpretable & Adaptive House Price Prediction Engine

Table of Contents

---

1. Project Overview

SmartExplain AI is a production-level machine learning system that predicts house prices using **linear regression trained via gradient descent** (not sklearn's closed-form solution). It emphasizes:

- **Interpretability** – per-feature contributions to each prediction
- **Adaptive optimization** – momentum, learning rate decay, early stopping
- **Modularity** – separate core, visualization, simulator, and app modules
- **Reproducibility** – fixed random seeds and clear pipeline

Main model: Custom `LinearRegressionGD` class. Sklearn's `LinearRegression` is used only for comparison.

---

2. How It Works (End-to-End Flow)

```
┌─────────────────────────────────────────────────────────────────┐
│                    SmartExplain AI Pipeline                      │
├─────────────────────────────────────────────────────────────────┤
│                                                                  │
│  1. DATA LOADING                                                 │
│     └── data/train.csv (Kaggle House Prices)  OR  data/housing.csv (California)│
│                                                                  │
│  2. FEATURE ENGINEERING                                          │
│     └── core/feature_engineering.py (California)                 │
```

```
│     └── core/feature_engineering_house_prices.py (Kaggle)            │
│         • Handle missing values                                      │
│         • Outlier capping (1st-99th percentile)                      │
│         • Log transforms (skewed features)                           │
│         • Engineered features (interactions, location, etc.)         │
│         • Polynomial expansion (squares + pairwise products)         │
│         • Standardization: (x - μ) / σ                               │
│                                                                      │
│  3. TRAIN/TEST SPLIT (80/20)                                         │
│     └── Random permutation, reproducible with seed                   │
│                                                                      │
│  4. MODEL TRAINING                                                   │
│     └── core/model.py – LinearRegressionGD                          │
│         • Batch gradient descent                                     │
│         • Momentum (β=0.9)                                          
│
│         • Learning rate decay (time-based)                           │
│         • Early stopping (patience=150)                              │
│         • L2 regularization (λ=0.02)                                 │
│                                                                      │
│  5. RETRAIN ON FULL DATA                                             │
│     └── Final model trained on all samples for deployment            │
│                                                                      │
│  6. SAVE MODEL                                                       │
│     └── model.pkl (weights, bias, feature engineer, feature names)   │
│                                                                      │
│  7. PREDICTION & EXPLAINABILITY                                      │
│     └── core/explainability.py – Contribution_i = w_i × x_i          │
│     └── simulator/what_if.py – simulate_price_change()               │
│                                                                      │
│  8. INTERACTIVE APP                                                  │
│     └── app/streamlit_app.py – Sliders → prediction + contribution chart   │
│                                                                      │
└─────────────────────────────────────────────────────────────────────┘
```

---

3. Datasets

3.1 Kaggle House Prices (Primary)

- **File:** `data/train.csv`
- **Source:** House Prices - Advanced Regression Techniques
- **Samples:** 1,460
- **Target:** `SalePrice` (house sale price in USD)
- **Features:** 80 columns (numeric + categorical)
- **Selected numeric:** LotArea, OverallQual, GrLivArea, GarageCars, TotalBsmtSF, YearBuilt, FullBath, Fireplaces, etc.

If `train.csv` exists in `data/`, this dataset is used.

3.2 California Housing (Fallback)

- **File:** `data/housing.csv`
- **Samples:** ~20,000
- **Target:** `median_house_value`
- **Features:** longitude, latitude, housing_median_age, total_rooms, total_bedrooms, population, households, median_income, ocean_proximity

Used when `train.csv` is not present.

---

4. Feature Engineering

4.1 Kaggle House Prices (`HousePricesFeatureEngineer`)

1. **Numeric columns:** 34 columns (LotFrontage, LotArea, OverallQual, YearBuilt, GrLivArea, GarageCars, etc.)
2. **Missing values:** filled with median per column
3. **Standardization:** $(x - \mu) / \sigma$
4. **Polynomial expansion (degree 2):**
   - Squares: x_i^2 for first 8 features
   - Interactions: $x_i \times x_j$ for pairs of first 8 features
5. **Output:** ~69 features

4.2 California Housing (`FeatureEngineer`)

1. **Base features:** longitude, latitude, median_income, total_rooms, total_bedrooms, housing_median_age, population, households
2. **Engineered features:**
   - `area_location_rating` = total_rooms × location_rating (from ocean_proximity)
   - `age_depreciation` = $\exp(-0.02 \times \text{age})$
   - `distance_from_center` = $\sqrt{(\text{lon} - \text{lon}_c)^2 + (\text{lat} - \text{lat}_c)^2}$
   - `rooms_per_household`, `bedrooms_per_room`
   - `income_area`, `income_age`, `dist_location`
3. **Log transforms:** log1p(total_rooms), log1p(bedrooms), log1p(population), etc.
4. **One-hot encoding:** ocean_proximity
5. **Outlier capping:** features clipped at 1st and 99th percentiles
6. **Polynomial expansion:** degree 2 on first 10 features
7. **Standardization:** $(x - \mu) / \sigma$

---

5. Model: LinearRegressionGD

Location: `core/model.py`

5.1 Formula

- **Prediction:** $y_{pred} = Xw + b$
- **Cost:** $J(w,b) = (1/2m)\ \Sigma(y_{pred} - y)^2 + \lambda\ \Sigma w^2$ (MSE + L2)
- **Gradients:**
  - $dw = (1/m)\ X^T(y_{pred} - y) + 2\lambda w$

   - db = (1/m) Σ(y_pred - y)

5.2 Modes

Mode	Batch Size	Description
`batch`	m (full data)	Uses all samples per iteration
`minibatch`	32 (default)	Random subset per iteration
`sgd`	1	Single sample per iteration

5.3 Features

- **Momentum:** $v = \beta \cdot v + \nabla J$, update w using v
- **Learning rate decay:** $\alpha_t = \alpha_0 / (1 + decay\_rate \cdot t)$ (time decay)
- **Early stopping:** stop if cost does not improve for `patience` iterations
- **Gradient clipping:** prevents overflow during training

5.4 Stored Artifacts

- `weights` (w)
- `bias` (b)
- `cost_history` (cost per iteration)

---

6. Optimization (Gradient Descent)

6.1 Momentum (`core/optimizers.py`)

```
v = β × v + ∇J
w = w - α × v
```

- $\beta = 0.9$
- Smoother updates and faster convergence in stable directions

6.2 Learning Rate Scheduler

Type	Formula
Time	$\alpha_t = \alpha_0 / (1 + \gamma \cdot t)$
Step	$\alpha_t = \alpha_0 \times \gamma$^(t // step)
Exponential	$\alpha_t = \alpha_0 \times \exp(-\gamma \cdot t)$

Used: **time decay** with $\gamma = 0.01$.

---

7. Mathematical Formulation

7.1 Linear Model

$$y = Xw + b$$

7.2 Cost Function (L2 Regularized)

$$J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} (y_{pred}^{(i)} - y^{(i)})^2 + \lambda \sum_{j} w_j^2$$

7.3 Gradient Updates

$$\frac{\partial J}{\partial w} = \frac{1}{m} X^T (y_{pred} - y) + 2\lambda w$$

$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i} (y_{pred}^{(i)} - y^{(i)})$$

7.4 Training Step

1. Compute $y_{pred} = Xw + b$
2. Compute dw, db
3. Apply momentum: $v = \beta \cdot v + dw$
4. Update: $w \leftarrow w - \alpha \cdot v$, $b \leftarrow b - \alpha \cdot db$

---

8. Explainability

Location: `core/explainability.py`

8.1 Feature Contribution

For linear model $y = w \cdot x + b$:

$$\text{Contribution}_i = w_i \times x_i$$

8.2 Output

- **total_prediction:** predicted price
- **contributions:** per-feature contribution ($w_i \times x_i$)
- **percentages:** relative influence = $100 \times |contribution_i| / \Sigma|contribution_j|$

8.3 Use Case

Shows which features (e.g., GrLivArea, OverallQual) most influence each prediction.

---

9. What-If Simulator

Location: `simulator/what_if.py`

9.1 Function

```
simulate_price_change(model, X_current, feature_name, new_value, feature_names,
explainer)
```

9.2 Returns

- `original_prediction` – price before change
- `updated_prediction` – price after change
- `price_difference` – Δ = updated - original
- `contribution_breakdown_original` – contributions before
- `contribution_breakdown_updated` – contributions after

9.3 Example

Change `GrLivArea` from 1710 to 2500 → get new price and updated contribution breakdown.

---

10. Visualization

10.1 `visualization/plots.py`

- **Cost vs iterations** – training cost curve
- **Optimizer comparison** – batch vs minibatch vs SGD
- **Actual vs predicted** – scatter plot
- **Learning rate comparison** – convergence for different α

10.2 `visualization/cost_surface.py`

- **3D cost surface** – J(w1, w2) vs w1, w2
- **Gradient descent path** – trajectory over the surface

---

11. Streamlit App

Location: `app/streamlit_app.py`

11.1 House Prices Mode (Kaggle)

Sliders:

- Lot Area, Overall Quality, Gr Liv Area, Garage Cars
- Total Basement Sq Ft, Year Built, Full Bath, Fireplaces

11.2 California Mode

Sliders:

- Area, Bedrooms, Location Rating, Age, Distance
- Median Income, Population, Households
- Longitude, Latitude, Ocean Proximity

11.3 Display

- Predicted price

- Feature contribution bar chart
- Training cost curve

---

12. Training Pipeline

12.1 `train.py` Flow

1. **Detect dataset:** use `train.csv` if present, else `housing.csv`
2. **Feature engineering:** call `fe.fit_transform(df)`
3. **Split:** 80% train, 20% test (random, seeded)
4. **Train:** `LinearRegressionGD.fit(X_train, y_train)`
5. **Retrain:** fit again on full data
6. **Evaluate:** MAE, RMSE, R^2 on train, test, and full data
7. **Save:** pickle `{model, fe, feature_names, dataset}` to `model.pkl`

12.2 Hyperparameters (Current)

Parameter	Value
learning_rate	0.03
n_iterations	5000
regularization	0.02
momentum	0.9
decay_type	time
patience	150
random_state	42

---

13. Project Structure

```
SmartExplain-AI/
├── data/
│   ├── housing.csv          # California housing
│   ├── train.csv            # Kaggle House Prices
│   ├── test.csv
│   ├── data_description.txt
│   └── house_prices.zip
├── notebooks/
│   ├── SmartExplain_AI.ipynb
│   └── SmartExplain_AI_executed.ipynb
├── core/
│   ├── __init__.py
│   ├── model.py             # LinearRegressionGD
│   ├── optimizers.py        # Momentum, LR decay
│   ├── feature_engineering.py        # California
│   ├── feature_engineering_house_prices.py  # Kaggle
```

```
│   ├── metrics.py           # MAE, MSE, RMSE, R² (manual)
│   └── explainability.py    # Feature contributions
├── visualization/
│   ├── plots.py
│   └── cost_surface.py
├── simulator/
│   └── what_if.py
├── app/
│   └── streamlit_app.py
├── train.py
├── model.pkl
├── requirements.txt
├── README.md
└── PROJECT_DETAILS.md
```

14. Results

14.1 Kaggle House Prices (train.csv)

Metric	Train 80%	Test	Full Data
R^2	0.87	**0.90**	0.88
MAE	$18,821	$16,410	—
RMSE	$28,914	$22,751	—

14.2 California Housing (housing.csv)

Metric	Typical Value
R^2	~0.67–0.70
MAE	~48,000–53,000
RMSE	~66,000–73,000

Quick Commands

```
# Install
pip install -r requirements.txt

# Train
python train.py

# Run app
streamlit run app/streamlit_app.py

# Run notebook
jupyter notebook notebooks/SmartExplain_AI.ipynb
```

```
# or
python -m jupyter nbconvert --to notebook --execute
notebooks/SmartExplain_AI.ipynb --output SmartExplain_AI_executed.ipynb
```