Interpreting Machine Learning Models: Issues, Methods, and Limitations

M2 Statistics - Statistical Learning

Pierre Pudlo

Aix-Marseille Université / Faculté des Sciences

Introduction: Why Interpret?

Motivations for Model Interpretation

Why do we need to interpret machine learning models?

1. **Scientific understanding**
    - Understand relationships between features and outcomes
    - Generate new hypotheses
    - Validate domain knowledge
2. **Trust and adoption**
    - Users need to trust predictions
    - Stakeholders require justifications
    - Critical for high-stakes decisions (healthcare, finance, justice)
3. **Debugging and diagnostics**
    - Identify when models fail
    - Detect data quality issues
    - Improve model performance

More Motivations

4. **Bias and fairness**
    - Detect discriminatory patterns
    - Ensure equitable treatment across groups
    - Identify spurious correlations
5. **Legal constraints**
    - GDPR: “right to explanation”
    - Regulatory requirements in banking, insurance
    - Accountability and transparency

Key question: How can we understand what complex models are doing?

Running Example: California Housing Dataset

Throughout this course, we will use concrete examples based on the **California Housing dataset** with a Random Forest model.

Dataset: 20,640 census block groups in California (1990 U.S. Census)

Features (8): MedInc (median income), HouseAge, AveRooms, AveBedrms, Population, AveOccup, Latitude, Longitude

Target: MedHouseVal (median house value in $100k)

```python
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
import pandas as pd
import numpy as np

# Load data
california = fetch_california_housing()
X = pd.DataFrame(california.data, columns=california.feature_names)
y = pd.Series(california.target, name='MedHouseVal')

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
```

```
    X, y, test_size=0.25, random_state=42
)

# Fit Random Forest (will be used throughout examples)
rf = RandomForestRegressor(n_estimators=100, max_features=3,
                           random_state=42, n_jobs=-1)
rf.fit(X_train, y_train)
```

```
Random Forest R²: 0.814
Dataset: 15480 train, 5160 test samples
```

Two Main Families of Interpretation

Intrinsic interpretability:

- Models that are inherently understandable
- Interpretation is built into the model structure
- Examples: linear regression, decision trees, rule-based models

Post-hoc interpretability:

- Methods applied after training
- Work with any “black-box” model
- Examples: feature importance, SHAP, LIME

Trade-off: Intrinsically interpretable models may be less accurate, but post-hoc methods can be unstable or misleading.

1 Intrinsically Interpretable Models

Why models can be Interpretable intrinsically?

- **Linear models/Logistic regression**: coefficients directly indicate feature effects
- **Decision trees**: decisions are based on simple rules
- **GAM (Generalized Additive Models)**: smooth functions for each feature (not in this course)
- **Rule-based models**: explicit IF-THEN rules (not in this course)

When to Prefer Intrinsically Interpretable Models?

Choose intrinsic interpretability when:

- **Regulatory requirements** demand full transparency
- **Stakeholders** are non-technical
- **Data size** is small to moderate
- **Debugging** and understanding is more important than accuracy
- **Domain knowledge** can be validated through model structure

Accept post-hoc methods when:

- Predictive accuracy is paramount
- Complex patterns require flexible models
- Ensemble methods provide significant performance gains

Key principle: Always start simple! Try interpretable models first, then add complexity if needed.

1.1 Linear Models

Linear and Logistic Regression

Linear regression:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

Interpretation: β_j represents the change in $\hat{y}$ when x_j increases by 1 unit, holding all other features constant.

Logistic regression: if $p(x) = P(Y = 1|X = x)$, then

$$\log\left(\frac{p(x)}{1 - p(x)}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$$

Interpretation: $\exp(\beta_j)$ is the odds ratio for a one-unit increase in x_j.

Logistic Regression: Odds Ratio

$$\text{Odds ratio} = \frac{p(x)}{1-p(x)} = \exp\left(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p\right)$$

- Each term $\exp(\beta_j)$ represents the multiplicative change in odds for a one-unit increase in x_j, holding other features constant.

ⓘ Odds ratio?

- Odds (or “cote” in French) are precisely the quantities employed in **sports betting** to express the relative likelihood of an event (often winning).
- If team A has probability $p = 0.75$ of winning, then odds = $p/(1-p) = 0.75/0.25 = 3/1$
  $\longrightarrow$ odds of 3 to 1 in favour of a win of team A.

Interpreting Coefficients

Example: Predicting house prices

```python
from sklearn.linear_model import LinearRegression
import numpy as np
import pandas as pd

# Fit linear model
model = LinearRegression()
model.fit(X_train, y_train)

# Display coefficients
coef_df = pd.DataFrame({
    'Feature': feature_names,
    'Coefficient': model.coef_
})
print(coef_df.sort_values('Coefficient', ascending=False))
```

Interpretation: A coefficient of 0.5 for “number of rooms” means that each additional room increases the predicted price by 0.5 units (in the scale of the response).

Effect of Standardisation

Problem: Coefficients depend on the scale of features.

- Feature measured in meters vs. kilometers $\rightarrow$ coefficient changes by factor of 1000
- Hard to compare importance across features

Solution: Standardise features before fitting:

$$x_j^{\text{std}} = \frac{x_j - \bar{x}_j}{\text{sd}(x_j)}$$

Now: Coefficients represent the change in $\hat{y}$ for a one-standard-deviation change in x_j.

Advantage: Can now compare magnitudes of coefficients to assess relative importance.

Caution with Standardisation

⚠ Caution: test leakage!

- Standardisation must be done using **training data only**
- $\bar{x}_j$ and $\mathrm{sd}(x_j)$ computed on training set
- Apply same transformation to test data with the values from training set

The same is also true when imputing missing values.

Problems with Linear Model Interpretation

1. Collinearity

- When features are correlated, coefficients become unstable
- Small changes in data $\rightarrow$ large changes in coefficients
- Individual coefficients may not reflect true importance

2. Interactions

- Linear models assume additive effects
- Real-world: effect of x_1 often depends on x_2
- Must manually add interaction terms

3. Non-linear features

- Linear models assume linear relationships
- May need to add polynomial terms, splines, etc.
- Interpretation becomes more complex

1.2 Decision Trees

Reading a Decision Tree

Advantages:

- Visual representation of decision process
- Easy to explain to non-experts
- Captures non-linear relationships and interactions naturally

Feature Importance in Trees

Impurity reduction:

For each feature X_j, compute the total reduction in impurity (Gini, entropy) across all splits involving X_j:

$$\text{Importance}_j = \sum_{t:\text{split on } X_j} \text{Impurity decrease at node } t$$

Normalised: Sum to 1 across all features.

Advantages:

- Automatic
- Accounts for all splits in the tree

Limitations:

- Biased towards high-cardinality features
- Doesn’t account for feature interactions well

Strengths and Limitations

Strengths:

- Highly interpretable (for small trees)
- Handles categorical and numerical features
- Captures interactions and non-linearities
- No need for feature scaling

Limitations:

- Single trees are unstable (high variance)
- Deep trees become hard to interpret
- Greedy algorithm may miss optimal splits
- Performance often inferior to ensemble methods

When to use: When interpretability is paramount and data size is moderate.

2 Global Interpretation of Black-Box Models

2.1 Feature Importance

Permutation Importance (Breiman)

Idea: Measure how much model performance decreases when a feature's values are randomly shuffled.

Algorithm:

1. Compute baseline performance on validation set: Score_0
2. For each feature X_j:
   - Randomly permute values of X_j in validation set
   - Compute new performance: Score_j
   - Importance = $\text{Score}_0 - \text{Score}_j$
3. Repeat permutation multiple times and average

Interpretation: Higher importance = feature is more useful for prediction.

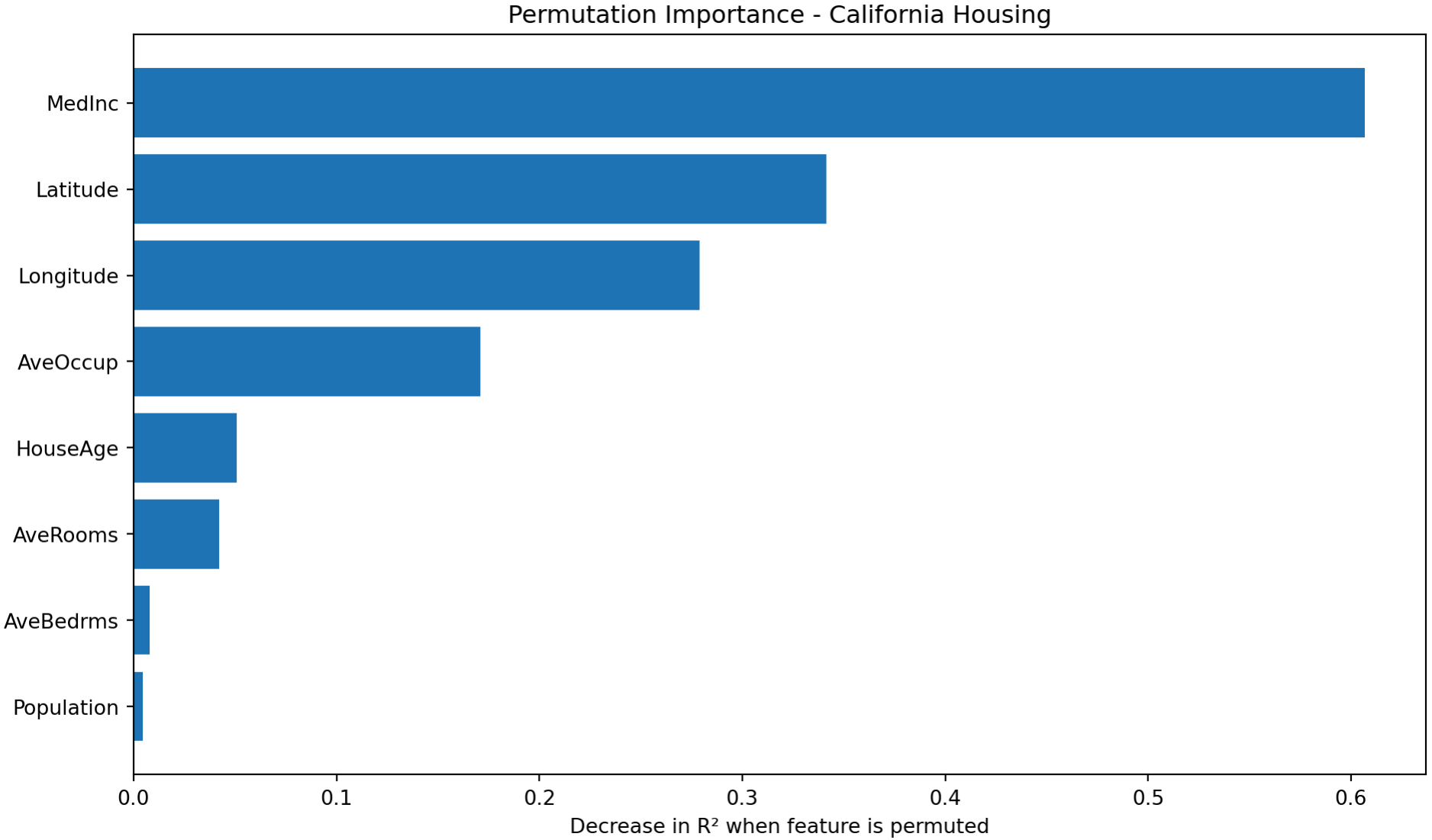
Permutation Importance: Implementation

```python
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt

# Compute permutation importance on our California Housing RF
result = permutation_importance(rf, X_test, y_test,
                                n_repeats=10, random_state=42,
                                scoring='r2')

# Display results
importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': result.importances_mean,
    'Std': result.importances_std
}).sort_values('Importance', ascending=False)

print(importance_df.to_string(index=False))

# Visualize
fig, ax = plt.subplots(figsize=(10, 6))
sorted_idx = result.importances_mean.argsort()
```

Observation: MedInc (median income) is by far the most important feature, followed by location (Latitude, Longitude).

Permutation Importance: Implementation

Feature	Importance	Std
MedInc	0.606608	0.011564
Latitude	0.341468	0.007400
Longitude	0.278832	0.005149
AveOccup	0.170829	0.004865
HouseAge	0.051001	0.002610
AveRooms	0.042067	0.001883
AveBedrms	0.007817	0.000525
Population	0.004737	0.000308


Permutation Importance - California Housing
MedInc
Latitude
Longitude
AveOccup
HouseAge
AveRooms
AveBedrms
Population
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Decrease in R² when feature is permuted

Permutation Importance: Strengths and Limitations

Strengths:

- Model-agnostic (works with any model)
- Based on actual performance metric
- Accounts for feature interactions
- No assumptions about model structure

Limitations:

1. **Correlated features:** If X_1 and X_2 are highly correlated, permuting X_1 may not decrease performance much (model uses X_2 instead)
2. **Computational cost:** Requires recomputing predictions multiple times
3. **Instability:** Results can vary with different random permutations

Best practice: Use multiple repetitions and report confidence intervals.

Gini/Gain Importance (Tree-Based Models)

For tree-based models (Random Forests, Gradient Boosting):

$$\text{Importance}_j = \frac{1}{B} \sum_{b=1}^{B} \sum_{t:\text{split on } X_j} \text{Impurity decrease at node } t$$

where B is the number of trees.

Advantages:

- Fast to compute (no additional predictions needed)
- Built into most tree implementations
- Available in `feature_importances_` attribute in scikit-learn

Gini/Gain Importance (Tree-Based Models) – continued

Disadvantages:

- Only for tree-based models
- Biased towards high-cardinality features
- Biased towards features with many possible split points
- Can be unreliable when features are correlated

Comparing Importance Measures

Key takeaway: Different methods can give different rankings! Always examine multiple perspectives.

2.2 Marginal Effects

Partial Dependence Plots (PDP)

Idea: Show the marginal effect of a feature, averaging over all other features.

Definition:

$$\mathrm{PDP}_j(x_j) = \mathbb{E}_{X_{\setminus j}}\left[\widehat{f}(x_j, X_{\setminus j}) \middle| \widehat{f}\right]$$

where $X_{\setminus j}$ denotes all features except X_j.

Estimation via averaging over the training dataset:

$$\widehat{\mathrm{PDP}}_j(x_j) = \frac{1}{n}\sum_{i=1}^{n} \widehat{f}(x_j, x_i^{(\setminus j)})$$

Interpretation: Shows how predictions change as X_j varies, on average.

Individual Conditional Expectation (ICE)

Idea: Show how predictions change for each individual observation as a feature varies.

Definition: $\mathrm{ICE}_i(x_j) = \widehat{f}(x_j, x_i^{(\setminus j)})$

Computation:

1. Choose grid of values for X_j
2. For each observation i and each value x_j in grid:
   - Set $X_j = x_j$ while keeping other features at $x_i^{(\setminus j)}$
   - Compute prediction

Result:

- One curve per observation
- $\widehat{\mathrm{PDP}}_j(x_j)$ is the average of all $\mathrm{ICE}_i(x_j)$ curves

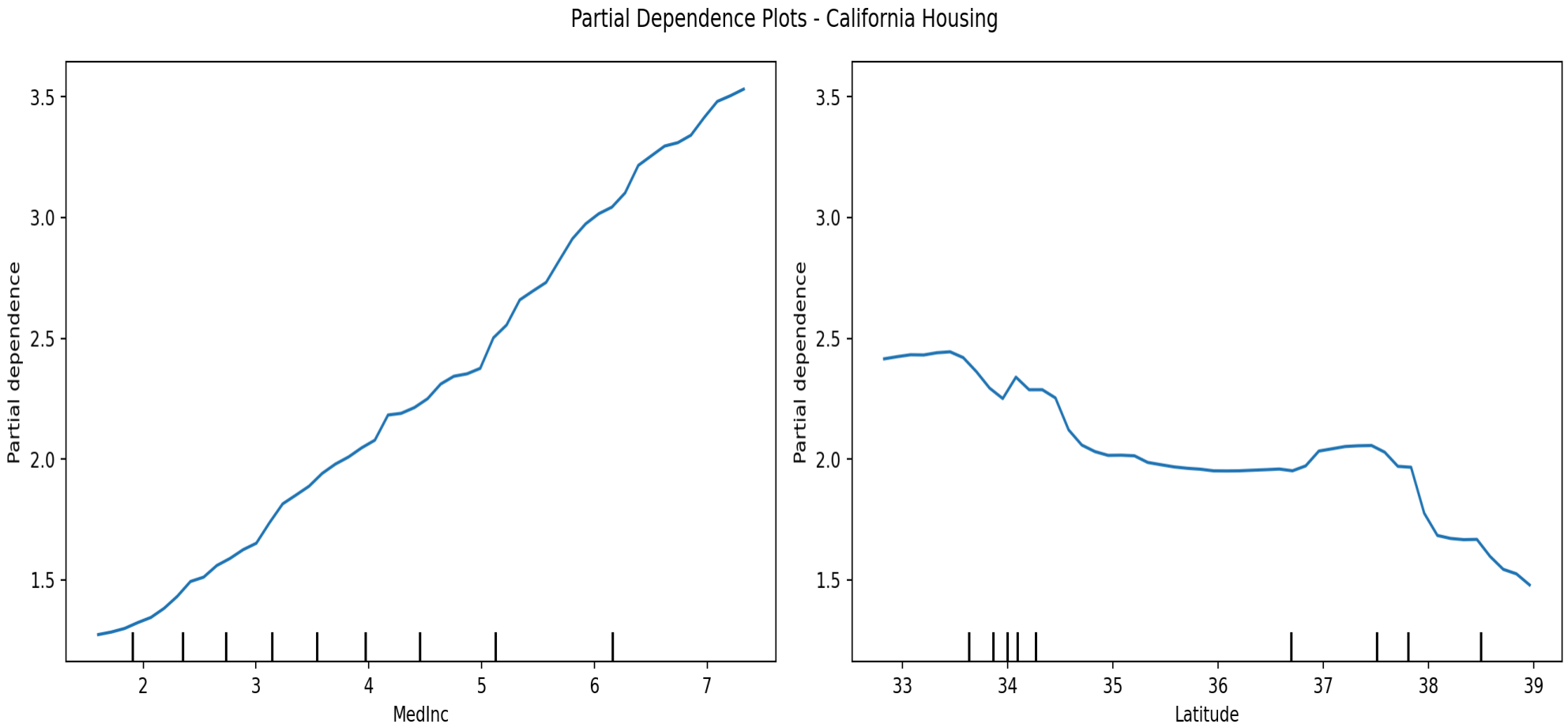
PDP: Example

```
from sklearn.inspection import PartialDependenceDisplay

# Create PDP for MedInc and Latitude
fig, ax = plt.subplots(1, 2, figsize=(14, 5))
PartialDependenceDisplay.from_estimator(
    rf, X_train, features=['MedInc', 'Latitude'],
    ax=ax, grid_resolution=50
)
plt.suptitle('Partial Dependence Plots - California Housing')
plt.tight_layout()
plt.show()
```

Reading the plot:

- **MedInc (left):** Strong positive relationship - higher median income → higher house prices
- **Latitude (right):** Non-linear effect - prices peak around lat 37-38° (San Francisco Bay Area)
- Y-axis shows average predicted house value

PDP: Example


Partial Dependence Plots - California Housing
Partial dependence
3.5
3.0
2.5
2.0
1.5
2
3
4
5
6
7
MedInc
Partial dependence
3.5
3.0
2.5
2.0
1.5
33
34
35
36
37
38
39
Latitude

PDP: Practical Parameters

Grid selection:

- **Automatic (default):** Uses deciles or percentiles of feature distribution
- **Manual:** Specify `grid_resolution=50` for finer grid
- **Custom:** Pass specific values via `grid` parameter

Example:

```python
# Fine grid for smooth curves
PartialDependenceDisplay.from_estimator(
    model, X_train, features=[0],
    grid_resolution=100  # More points = smoother curve
)
```

ICE plots:

- Use `kind='both'` to show both PDP (average) and ICE (individual curves)
- Use `kind='individual'` for ICE only
- **Warning:** With many samples, ICE can be cluttered (subsample if needed)

PDP: Limitations

1. Assumes feature independence

- PDP creates artificial data points by combining features
- If features are correlated, some combinations may be unrealistic
- Example: Setting “Age=20” and “Years of Experience=30”

2. Hides heterogeneous effects

- PDP shows only the average effect
- Individual effects may vary greatly
- Interactions are averaged out

PDP: Limitations – continued

3. Interpretation issues

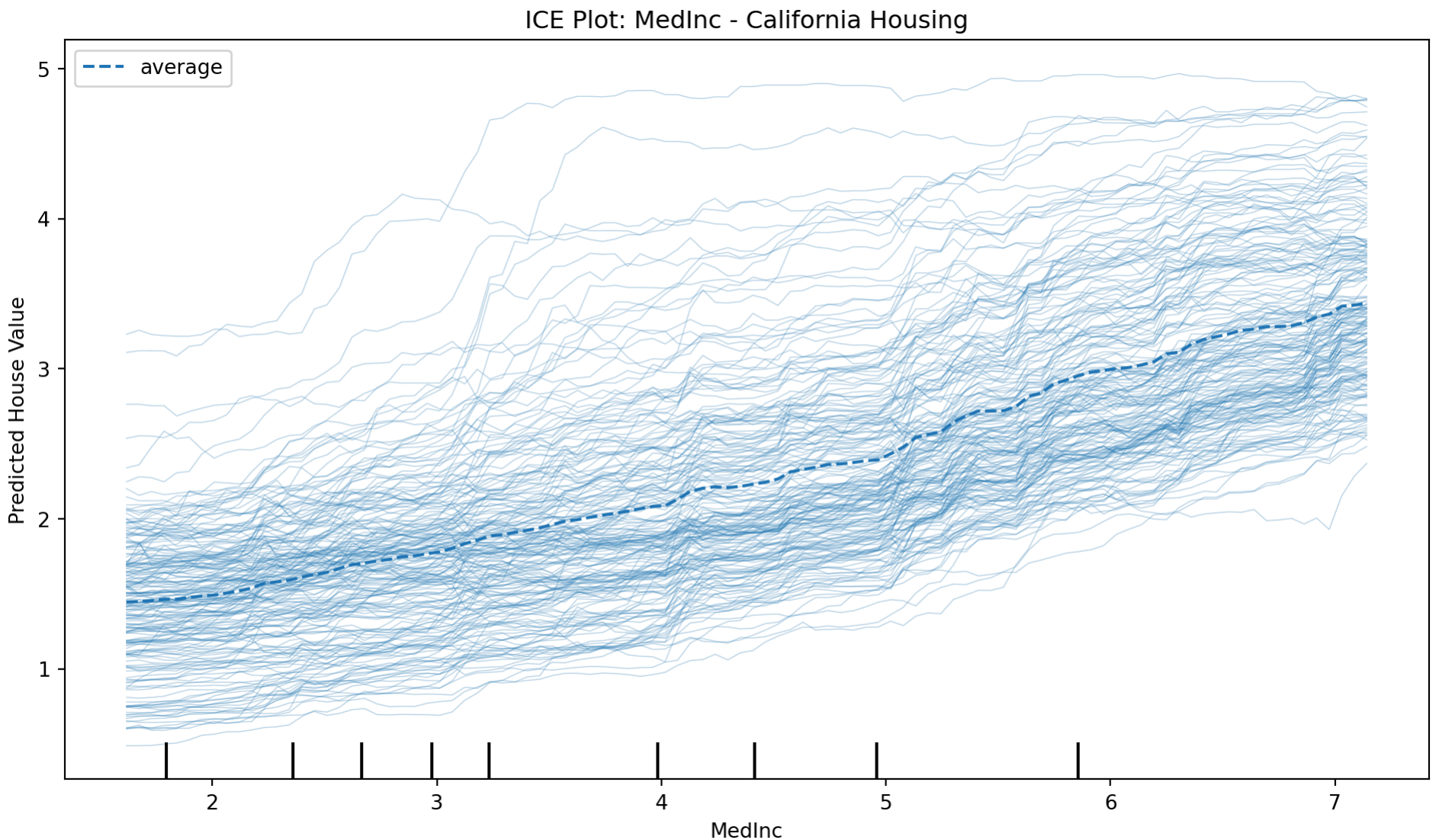
- For correlated features, PDP can be misleading
- Extrapolation to rare feature combinations

Solution: Use ICE plots and ALE plots to address these issues.

ICE: Example and Interpretation

```python
from sklearn.inspection import PartialDependenceDisplay

# Create ICE plot for MedInc
# Subsample to 200 curves for readability
np.random.seed(42)
sample_idx = np.random.choice(len(X_train), 200, replace=False)
X_sample = X_train.iloc[sample_idx]

fig, ax = plt.subplots(figsize=(10, 6))
display = PartialDependenceDisplay.from_estimator(
    rf, X_sample, features=['MedInc'],
    kind='both',  # Shows both PDP (thick) and ICE (thin lines)
    centered=False,
    ax=ax
)
display.axes_[0, 0].set_title('ICE Plot: MedInc - California Housing')
display.axes_[0, 0].set_ylabel('Predicted House Value')
plt.tight_layout()
plt.show()
```

ICE: Example and Interpretation



ICE: Example and Interpretation – continued

Reading ICE plots:

- Each thin blue line = one census block's conditional expectation
- Thick dashed blue line = PDP (average of all ICE curves)
- **Observation:** Curves are roughly parallel → effect of MedInc is similar across observations (no strong interactions)
- Small spread shows model is stable across different contexts

Accumulated Local Effects (ALE)

Problem with PDP: When features are correlated, PDP uses unrealistic feature combinations.

ALE solution: Use only local changes within the correlation structure of the data.

Idea:

- Divide feature X_j into intervals → Ensures each interval has real data
- For each interval, compute average change in predictions:
  for an interval $I_k = [a_k, a_{k+1}]$,

$$\widehat{\mathrm{ALE}_j}(k) = \mathbb{E}\left[\hat{f}(a_{k+1}, X_{\setminus j}) - \hat{f}(a_k, X_{\setminus j}) \middle| X_j \in I_k\right]$$

- Accumulate these local effects over intervals up to x_j to get ALE function at x_j.

Intuition behind ALE

$$\mathrm{ALE}_j(x_j) = \int_{z=\min}^{x_j} \mathbb{E}\Big[\frac{\partial \widehat{f}(z, X_{\setminus j})}{\partial z} \Big| X_j = z\Big] dz - \mathbb{E}\left[\int_{z=\min}^{X_j} \mathbb{E}\Big[\frac{\partial \widehat{f}(z, X_{\setminus j})}{\partial z} \Big| X_j = z\Big] dz\right]$$

- Imagine you walk along the feature axis
- During the walk, we measure how the model's prediction changes right around each value, using only realistic data points
- Then you sum up all local changes to get the global effect
- Finally, center the function to have mean zero: $\mathbb{E}[\mathrm{ALE}_j(X_j)] = 0$

ALE: Why It Works Better

Advantages over PDP:

1. **Respects feature correlations**
    - Uses only realistic feature combinations
    - Conditions on actual data distribution
2. **Unbiased effect estimation**
    - Not confounded by correlations
    - Shows true marginal effect

ALE: Why It Works Better – continued

3. **Works with correlated features**
    - PDP can be misleading with correlations
    - ALE provides correct interpretation

Disadvantage: Slightly more complex to implement and interpret.

Recommendation: Use ALE when features are correlated (cor > 0.3); otherwise PDP is fine.

ALE: Practical Parameters

Number of bins:

- Default: Usually ~50 bins (adjusted for sample size)
- **Too few bins:** Lose resolution, miss non-linearities
- **Too many bins:** Noisy estimates, unstable curves

Rule of thumb: Start with `n_bins=min(50, n_samples // 100)`

Example:

```python
from PyALE import ale

# Standard usage with default bins
ale_effect = ale(
    X=X_train,
    model=rf,
    feature=[0],  # Feature index
    feature_names=feature_names
)

# Custom number of bins for finer control
ale_effect = ale(
    X=X_train,
    model=rf,
```

```
    feature=[0],
    bins=30  # Fewer bins for small datasets
)
```

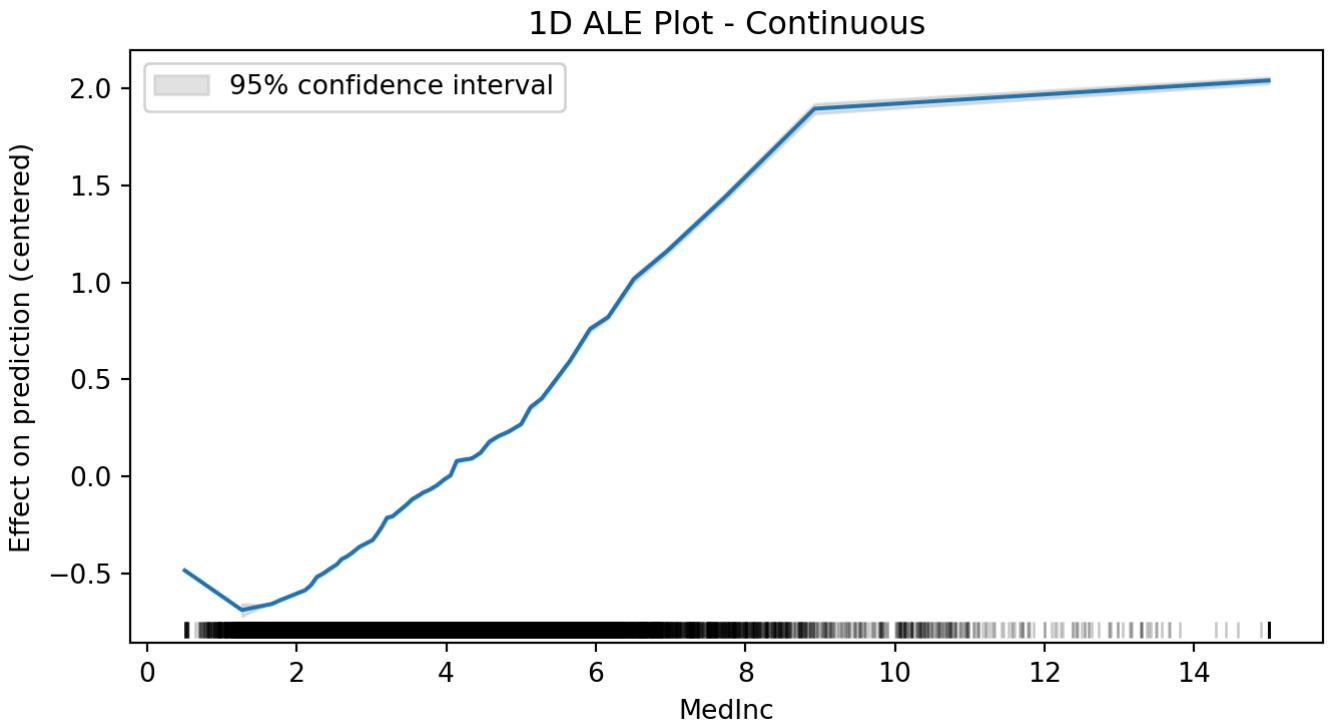
ALE: Implementation

```python
# Using PyALE package
from PyALE import ale

# Compute ALE for MedInc (feature 0)
ale_eff = ale(
    X=X_train,
    model=rf,
    feature=["MedInc"],
    grid_size=50,
    include_CI=True
)
# ale_eff is a dataframe with MedInc as index and columns: eff, size, lowerCI_95%, uppe
```

Reading ALE plot:

- **Positive slope:** Increasing MedInc increases house value predictions
- **Zero-centered:** ALE shows deviation from average effect
- **No unrealistic combinations:** ALE respects correlations in data (unlike PDP)
- **Comparison with PDP:** Shape is similar because features aren't highly correlated here

ALE: Implementation



ALE vs PDP: When Does It Matter?

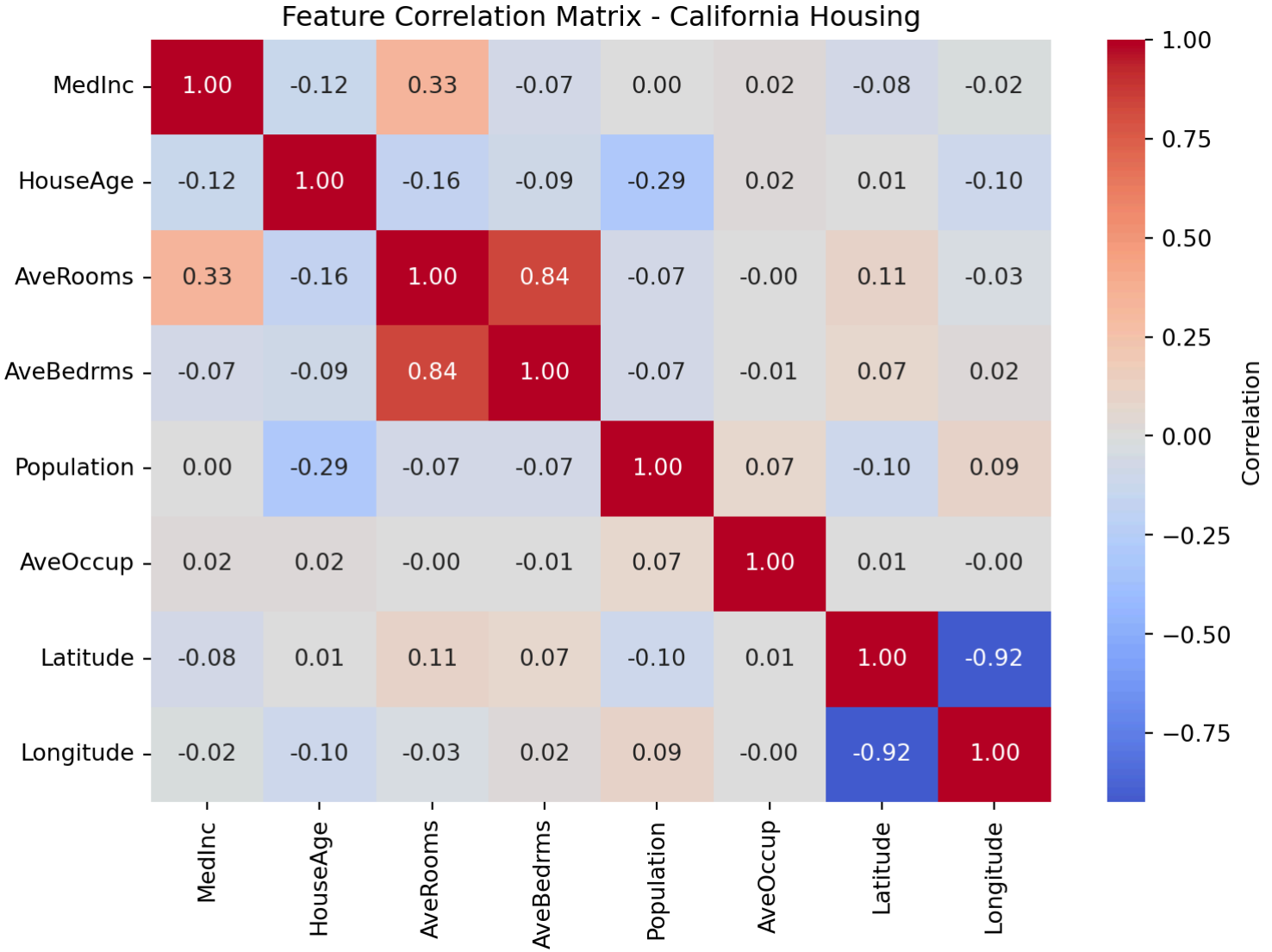
```python
# Check correlations in our data
import seaborn as sns

corr_matrix = X_train.corr()
print("Feature correlations with MedInc:")
print(corr_matrix['MedInc'].sort_values(ascending=False))

# Visualize correlation heatmap
fig, ax = plt.subplots(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm',
            center=0, ax=ax, cbar_kws={'label': 'Correlation'})
ax.set_title('Feature Correlation Matrix - California Housing')
plt.tight_layout()
plt.show()
```

Key observation:

- AveRooms and AveBedrms are highly correlated (0.85)
- For such features, ALE is preferable to PDP
- MedInc has low correlations → PDP and ALE give similar results

ALE vs PDP: When Does It Matter?

```
Feature correlations with MedInc:
MedInc        1.000000
AveRooms      0.331317
AveOccup      0.023268
Population    0.002841
Longitude    -0.016418
AveBedrms    -0.070085
Latitude     -0.077450
HouseAge     -0.120281
Name: MedInc, dtype: float64
```


Feature Correlation Matrix - California Housing
MedInc
HouseAge
AveRooms
AveBedrms
Population
AveOccup
Latitude
Longitude
1.00 -0.12 0.33 -0.07 0.00 0.02 -0.08 -0.02
-0.12 1.00 -0.16 -0.09 -0.29 0.02 0.01 -0.10
0.33 -0.16 1.00 0.84 -0.07 -0.00 0.11 -0.03
-0.07 -0.09 0.84 1.00 -0.07 -0.01 0.07 0.02
0.00 -0.29 -0.07 -0.07 1.00 0.07 -0.10 0.09
0.02 0.02 -0.00 -0.01 0.07 1.00 0.01 -0.00
-0.08 0.01 0.11 0.07 -0.10 0.01 1.00 -0.92
-0.02 -0.10 -0.03 0.02 0.09 -0.00 -0.92 1.00
Correlation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

2.3 Studying Interactions

2D Partial Dependence Plots

Idea: Extend PDP to two features to visualise interactions.

Estimated with:

$$\widehat{\mathrm{PDP}}_{jk}(x_j, x_k) = \frac{1}{n} \sum_{i=1}^{n} \hat{f}(x_j, x_k, x_i^{(\setminus jk)})$$

Visualisation: Heatmap or contour plot showing how predictions change with both X_j and X_k.

Interpretation:

- If plot shows parallel lines $\rightarrow$ no interaction
- If lines are not parallel or contours are curved $\rightarrow$ interaction present

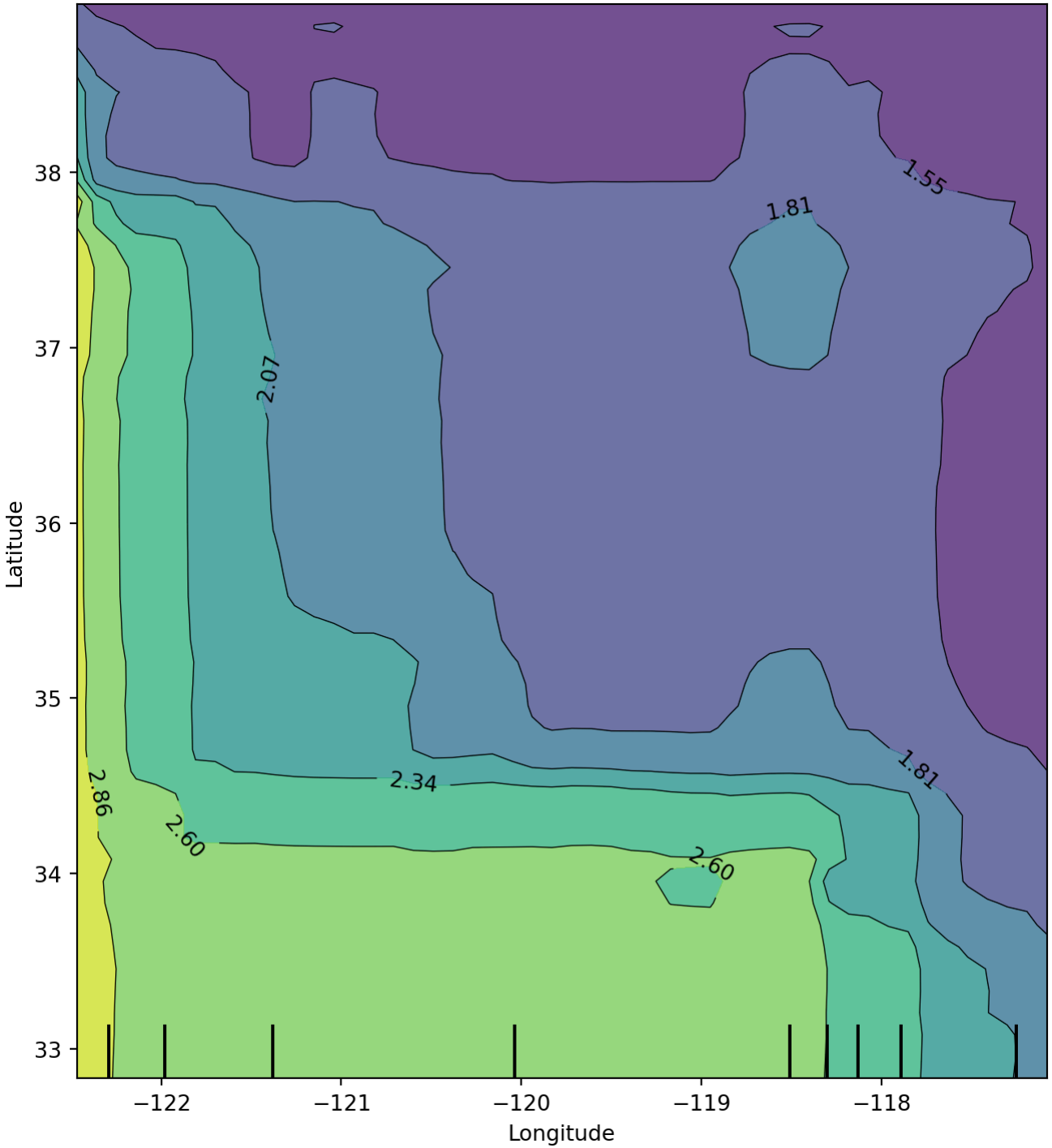
2D PDP: Example

```python
from sklearn.inspection import PartialDependenceDisplay

# Bornes réelles pour mettre les axes à l'échelle
lon_min, lon_max = X_train['Longitude'].min(), X_train['Longitude'].max()
lat_min, lat_max = X_train['Latitude'].min(), X_train['Latitude'].max()

# 2D PDP: Latitude vs Longitude (interaction géographique)
fig, ax = plt.subplots(figsize=(7, 7.63))
display = PartialDependenceDisplay.from_estimator(
    rf, X_train,
    features=[('Longitude', 'Latitude')],
    ax=ax,
    grid_resolution=50
)
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
ax.set_xlim(lon_min, lon_max)
ax.set_ylim(lat_min, lat_max)
ax.set_aspect('equal', adjustable='box')

ax.set_title('2D PDP: Geographic Interaction - California Housing')
```

Reading:

- Colour intensity = predicted house value
- **Observation:** Clear geographic patterns emerge
- High values (yellow) around San Francisco Bay Area (lat ~37.5, long ~-122)
- Low values (purple) in inland/northern regions
- **Interaction:** Location matters as a combination (not just latitude or longitude alone)

2D PDP: Example


Latitude
38
37
36
35
34
33
−122
−121
−120
−119
−118
Longitude
1.55
1.81
2.07
2.86
2.60
2.34
1.81
2.60

Friedman's H-statistic

Idea: Quantify the strength of interaction between features.

Definition: For features j and k:

$$H_{jk}^2 = \frac{\sum_i \left(\text{PDP}_{jk}(x_{ij}, x_{ik}) - \text{PDP}_j(x_{ij}) - \text{PDP}_k(x_{ik}) \right)^2}{\sum_i \text{PDP}_{jk}(x_{ij}, x_{ik})^2}$$

Interpretation:

- $H_{jk} = 0$: No interaction (effects are additive)
- $H_{jk} > 0$: Interaction present
- Larger values indicate stronger interactions

Use case: Screen for important interactions before creating 2D PDPs.

H-statistic: Example

	Feature 1	Feature 2	HStatistic
25	AveOccup	Latitude	1.023669
27	Latitude	Longitude	1.016592
23	Population	Latitude	1.007288
22	Population	AveOccup	1.005818
19	AveBedrms	AveOccup	1.003871
10	HouseAge	AveOccup	1.000766
20	AveBedrms	Latitude	1.000038
11	HouseAge	Latitude	0.997880
26	AveOccup	Longitude	0.990923
18	AveBedrms	Population	0.982424

Detecting Interactions: Summary

Methods:

1. **2D PDP:** Visual inspection of interaction patterns
2. **H-statistic:** Numerical measure of interaction strength
3. **ICE plots:** Show heterogeneity that may indicate interactions

Best practice:

- Start with H-statistic to identify candidate interactions
- Use 2D PDPs to visualise important interactions
- Validate with domain knowledge

Limitation: Computational cost increases with number of feature pairs ($p(p-1)/2$ pairs).

3 Fairness and Model Auditing

3.1 Measuring Fairness

Why Fairness Matters

Recall from introduction: Models can encode and amplify societal biases.

Key questions:

- Does the model treat different demographic groups fairly?
- Are predictions equally accurate across protected attributes (age, gender, race)?
- Does the model satisfy legal requirements (GDPR, anti-discrimination laws)?

Challenge: Multiple definitions of fairness, often incompatible.

This section: Practical metrics and tools to audit models for fairness.

Common Fairness Metrics

1. Demographic Parity (Statistical Parity)

$$P(\widehat{Y} = 1|A = 0) = P(\widehat{Y} = 1|A = 1)$$

where A is a protected attribute (e.g., gender, race).

Interpretation: Positive predictions should occur at the same rate across groups.

Example: Loan approval rate should be same for men and women.

Limitation: Ignores differences in actual outcomes (may be inappropriate if base rates differ).

More Fairness Metrics

2. Equalized Odds

$$P(\widehat{Y} = 1|Y = y, A = 0) = P(\widehat{Y} = 1|Y = y, A = 1) \quad \text{for } y \in \{0, 1\}$$

Interpretation: True positive rate and false positive rate should be equal across groups.

Stronger requirement: Ensures fairness conditional on actual outcome.

3. Equal Opportunity

$$P(\widehat{Y} = 1|Y = 1, A = 0) = P(\widehat{Y} = 1|Y = 1, A = 1)$$

Interpretation: True positive rate should be equal across groups (relaxed version of equalized odds).

Disparate Impact

Legal definition (US Equal Employment Opportunity Commission):

$$\text{Disparate Impact} = \frac{P(\hat{Y} = 1|A = 1)}{P(\hat{Y} = 1|A = 0)}$$

80% rule: If ratio < 0.8, there may be discrimination.

Example:

- Group A: 60% approval rate
- Group B: 40% approval rate
- Disparate impact = 40/60 = 0.67 < 0.8 → potential issue

Note: This is a screening tool, not definitive proof of discrimination.

Fairness-Accuracy Trade-off

Reality: Cannot satisfy all fairness criteria simultaneously while maintaining accuracy.

Impossibility theorem (Kleinberg et al., 2017):

Except in trivial cases, cannot simultaneously satisfy:

- Calibration (predicted probabilities match true probabilities)
- Equal false positive rates
- Equal false negative rates

Implication: Must choose which fairness criterion matters most for your application.

Best practice: Document choice and justify based on domain context.

Auditing with Fairlearn

```python
from fairlearn.metrics import MetricFrame, selection_rate
from sklearn.metrics import accuracy_score, balanced_accuracy_score

# Compute metrics stratified by protected attribute
metric_frame = MetricFrame(
    metrics={
        'accuracy': accuracy_score,
        'selection_rate': selection_rate,
        'balanced_accuracy': balanced_accuracy_score
    },
    y_true=y_test,
    y_pred=y_pred,
    sensitive_features=sensitive_attribute  # e.g., gender, race
)

print(metric_frame.by_group)
print("\nDisparate impact:")
sr_by_group = metric_frame.by_group['selection_rate']
print(f"Ratio: {sr_by_group.min() / sr_by_group.max():.3f}")
```

Reading output: Compare metrics across groups to identify disparities.

Using Interpretation Tools for Fairness

How existing methods help:

1. **Feature importance:** Identify if protected attributes (or proxies) drive predictions
2. **PDP/ALE:** Check if effects differ by group
3. **SHAP:** Analyze individual predictions for members of protected groups
4. **Counterfactuals:** Show what changes are needed for different outcomes

Example workflow:

1. Train model without protected attributes
2. Audit performance across groups (fairlearn)
3. Use SHAP to identify proxy features correlated with protected attributes
4. Use PDP to visualize differential effects
5. Generate counterfactuals to understand recourse options

Mitigation Strategies

If unfairness is detected:

1. Pre-processing:

- Reweight training samples
- Transform features to remove bias

2. In-processing:

- Add fairness constraints during training
- Use fairness-aware algorithms (e.g., fairlearn)

3. Post-processing:

- Adjust decision thresholds by group
- Calibrate predictions separately

Important: Removing protected attribute from features is NOT sufficient (proxies remain).

Tools: `fairlearn`, `aif360` (AI Fairness 360), `themis-ml`

4 Local Interpretation of Models

4.1 LIME

LIME: Local Interpretable Model-Agnostic Explanations

Key idea: Approximate a complex model locally with an interpretable model.

Algorithm:

1. **Select instance** to explain: x_0
2. **Generate perturbations** around x_0
3. **Get predictions** from black-box model for perturbed instances
4. **Weight samples** by proximity to x_0
5. **Fit simple model** (e.g., linear regression) on weighted samples
6. **Interpret** the simple model's coefficients

Result: Local linear approximation that explains prediction at x_0.

LIME: Mathematical Formulation

Objective:

$$\xi(x_0) = \arg\min_{g \in \mathcal{G}} \mathcal{L}(f, g, \pi_{x_0}) + \Omega(g)$$

where:

- f = black-box model
- g = interpretable model (e.g., linear)
- $\mathcal{G}$ = class of interpretable models
- $\mathcal{L}$ = loss function measuring how well g approximates f
- π_{x_0} = proximity measure (weights for perturbed samples)
- $\Omega(g)$ = complexity penalty (encourages simplicity)

Interpretation: Find the simplest interpretable model that locally approximates the black-box model.

LIME: Example Implementation

```python
import lime
import lime.lime_tabular

# Create LIME explainer
explainer = lime.lime_tabular.LimeTabularExplainer(
    X_train,
    feature_names=feature_names,
    class_names=['No', 'Yes'],
    mode='classification'
)

# Explain a prediction
i = 0  # Instance to explain
exp = explainer.explain_instance(
    X_test[i],
    rf.predict_proba,
    num_features=5
)

# Visualise
exp.show_in_notebook()
```

LIME: Strengths and Limitations

Strengths:

- Model-agnostic
- Locally faithful
- Human-interpretable output
- Works for tabular, text, and image data

Limitations:

1. **Instability:** Small changes in perturbations can lead to different explanations
2. **Choice of locality:** How to define “local”? Kernel width matters greatly
3. **Sampling:** Quality depends on how perturbations are generated
4. **Fidelity:** Local model may not accurately represent black-box model

Best practice: Generate multiple explanations with different random seeds and check consistency.

LIME: Practical Parameters

Key parameters to tune:

1. Number of samples (`num_samples`):

- Default: 5000
- **Too few:** Unstable explanations
- **Too many:** Slower, diminishing returns
- **Recommendation:** Start with 5000, increase to 10000 if unstable

2. Kernel width (`kernel_width`):

- Controls “locality” of explanation
- Default: `np.sqrt(n_features) * 0.75`
- **Smaller:** More local (only very similar samples)
- **Larger:** Less local (includes more distant samples)

3. Number of features to show (`num_features`):

- How many top features to include in explanation
- Default: 10
- **Recommendation:** 5-10 for interpretability

LIME: California Housing Example

Let’s apply LIME to explain predictions from our Random Forest model on the California Housing dataset.

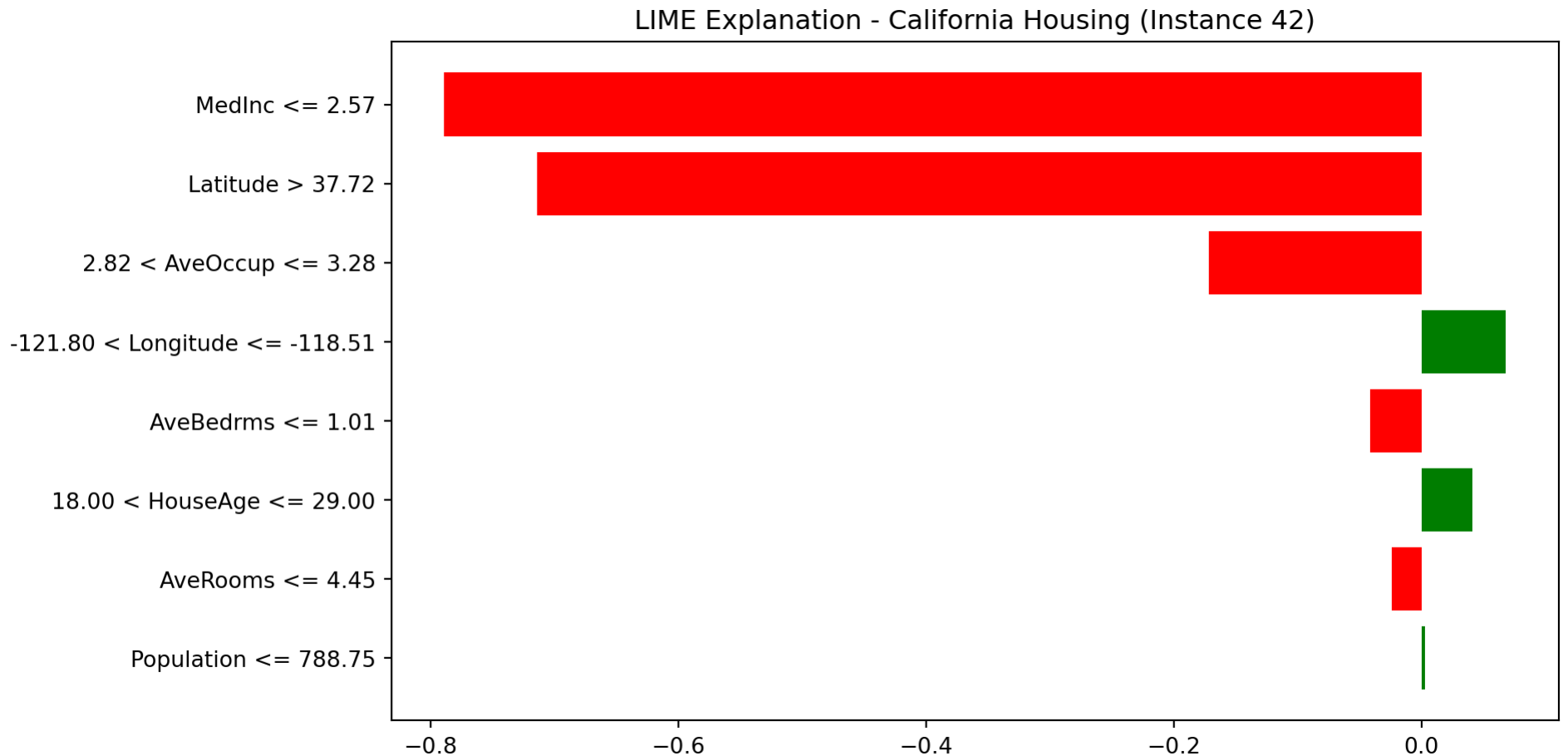
```
Instance 42:
  Actual house value: $0.71 (×100k)
  Predicted value: $1.07 (×100k)
  Prediction error: $0.36

Feature values:
  MedInc: 2.268
  HouseAge: 25.000
  AveRooms: 3.972
  AveBedrms: 0.962
  Population: 303.000
  AveOccup: 2.858
  Latitude: 37.780
  Longitude: -120.850
```

LIME: California Housing Example – continued

```python
import lime
import lime.lime_tabular
import matplotlib.pyplot as plt

# Create LIME explainer for regression
explainer = lime.lime_tabular.LimeTabularExplainer(
    X_train.values,
    feature_names=X_train.columns.tolist(),
    mode='regression',
    verbose=False
)

# Select an interesting instance to explain
instance_idx = 42
instance = X_test.iloc[instance_idx].values
actual_price = y_test.iloc[instance_idx]
predicted_price = rf.predict([instance])[0]

# Generate explanation
exp = explainer.explain_instance(
```

LIME: California Housing Example – continued

```
Local fidelity (R²): 0.291
Overall test R² of the trained Random Forest: 0.814
```



LIME: California Housing Example – continued

Reading this explanation:

- **Baseline (Intercept):** Average predicted house value from the local linear model
- **Feature contributions:** Red bars push prediction higher, blue bars push it lower
- **Feature ranges:** Show the actual value of each feature for this instance
- **Local fidelity:** R^2 score shows how well the linear model approximates the Random Forest locally

LIME: California Housing Example – continued

```
================================================
LIME Explanation:
================================================
MedInc <= 2.57                              decreases  prediction by 0.789
Latitude > 37.72                            decreases  prediction by 0.714
2.82 < AveOccup <= 3.28                     decreases  prediction by 0.172
-121.80 < Longitude <= -118.51              increases  prediction by 0.068
AveBedrms <= 1.01                           decreases  prediction by 0.041
18.00 < HouseAge <= 29.00                   increases  prediction by 0.041
AveRooms <= 4.45                            decreases  prediction by 0.024
Population <= 788.75                        increases  prediction by 0.003
```

4.2 SHAP

Shapley Values: Game Theory Background

Origin: Developed by Lloyd Shapley (1953) for fair distribution of payoffs in cooperative games.

Context:

- **Game Theory:** Many players cooperate to achieve a common goal. (Cooperative task)
- **Players:** Features in the model
- **Coalitions:** Subsets of features working together
- **Payoff:** Model prediction
- **Marginal contribution:** How much a feature adds to a coalition's payoff

Key concepts:

- Shapley values provides a principled way to decompose the overall outcome of a collaborative endeavour into individual shares
- by considering every conceivable way the coalition could be formed, and weighting each scenario appropriately

SHAP: SHapley Additive exPlanations

Idea: Use Shapley values from game theory to attribute prediction to features.

Shapley value for feature j:

$$\phi_j = \sum_{S \subseteq \{1,\ldots,p\} \setminus \{j\}} \frac{|S|!(p - |S| - 1)!}{p!} \left[\widehat{f}(S \cup \{j\}) - \widehat{f}(S) \right]$$

where:

- S = subset of features
- $\widehat{f}(S)$ = model prediction using only features in S
- Sum over all possible subsets not containing j

Interpretation: ϕ_j represents the average marginal contribution of feature j across all possible feature coalitions.

SHAP: Properties

Shapley values satisfy important properties:

1. **Efficiency:** $\sum_{j=1}^{p} \phi_j = f(x) - f(\emptyset)$
   - Contributions sum to the difference between prediction and baseline
2. **Symmetry:** If two features contribute equally, they get equal values
3. **Dummy:** If a feature doesn't contribute, its value is zero
4. **Additivity:** For ensemble models, Shapley values add up

Unique property: Shapley values are the only attribution method satisfying these properties!

SHAP: Approximation Methods

Problem: Computing exact Shapley values requires 2^p model evaluations (exponential!).

Solutions:

1. **KernelSHAP:** Uses weighted linear regression (LIME-like, but theoretically grounded)
2. **TreeSHAP:** Efficient exact computation for tree-based models
3. **DeepSHAP:** Approximation for neural networks
4. **LinearSHAP:** Exact for linear models

Most common: TreeSHAP for random forests and gradient boosting.

SHAP: Local Explanations (Force Plots)

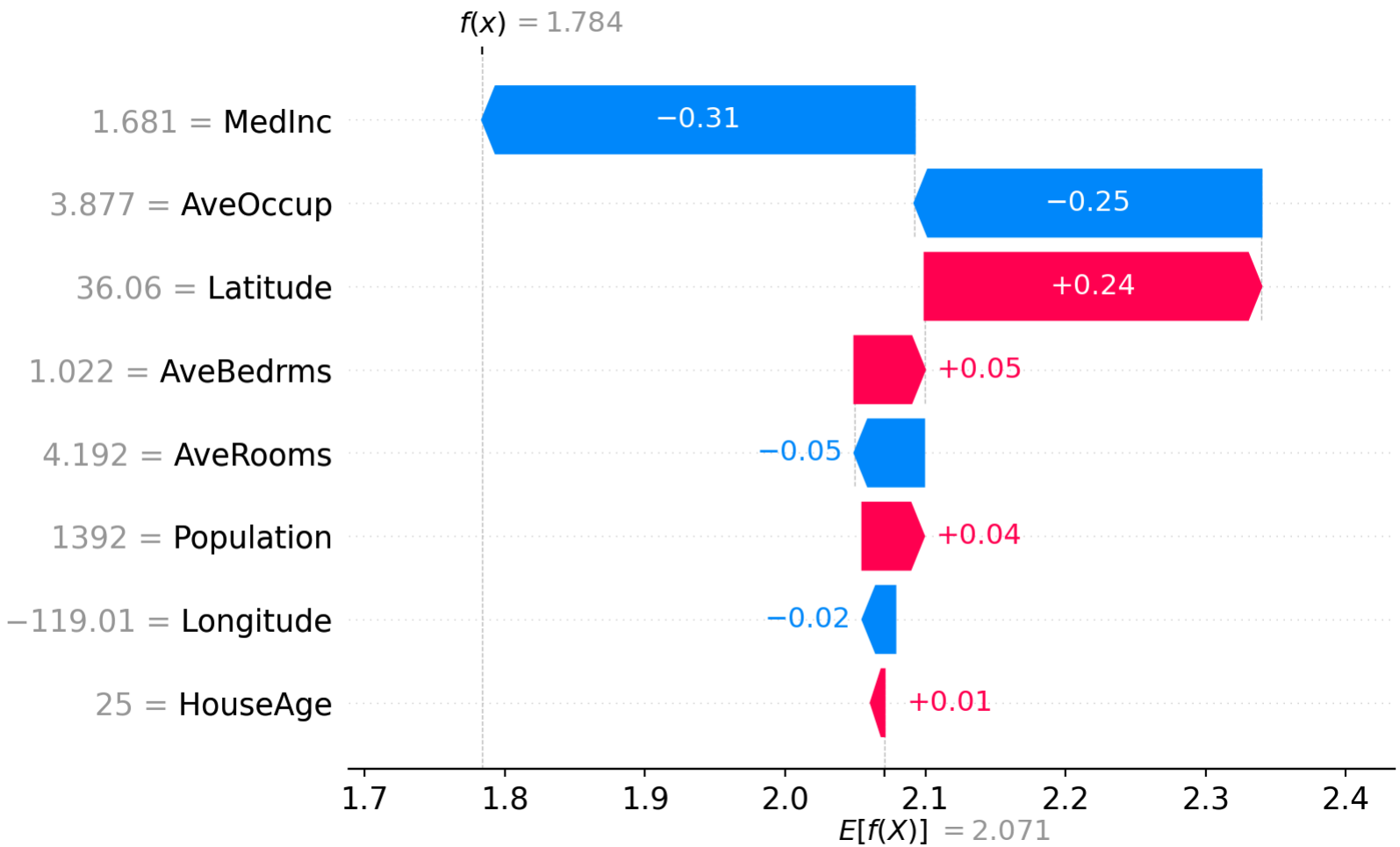
```python
import shap

print(f"Average of y in the training set: {np.mean(y_train):.3f}")
print(f"Prediction of the 12th test case: {np.mean(rf.predict(X_test[12:13])):.3f}")

# Create explainer for our Random Forest
explainer = shap.TreeExplainer(rf)

# Compute SHAP values for test set
shap_values = explainer.shap_values(X_test[12:13])

# Waterfall plot for first observation (more readable than force plot)
shap.plots.waterfall(shap.Explanation(
    values=shap_values[0],
    base_values=explainer.expected_value,
    data=X_test.iloc[0],
    feature_names=X_test.columns.tolist()
))
```

Reading waterfall plot:

- **E[f(X)]** = baseline (average prediction across all data)
- Bars show how each feature pushes prediction up (red) or down (blue)
- **f(x)** = final prediction for this specific observation
- Features ordered by absolute impact

SHAP: Local Explanations (Force Plots)

```
Average of y in the training set: 2.070
Prediction of the 12th test case: 1.784
```



SHAP: Global Explanations

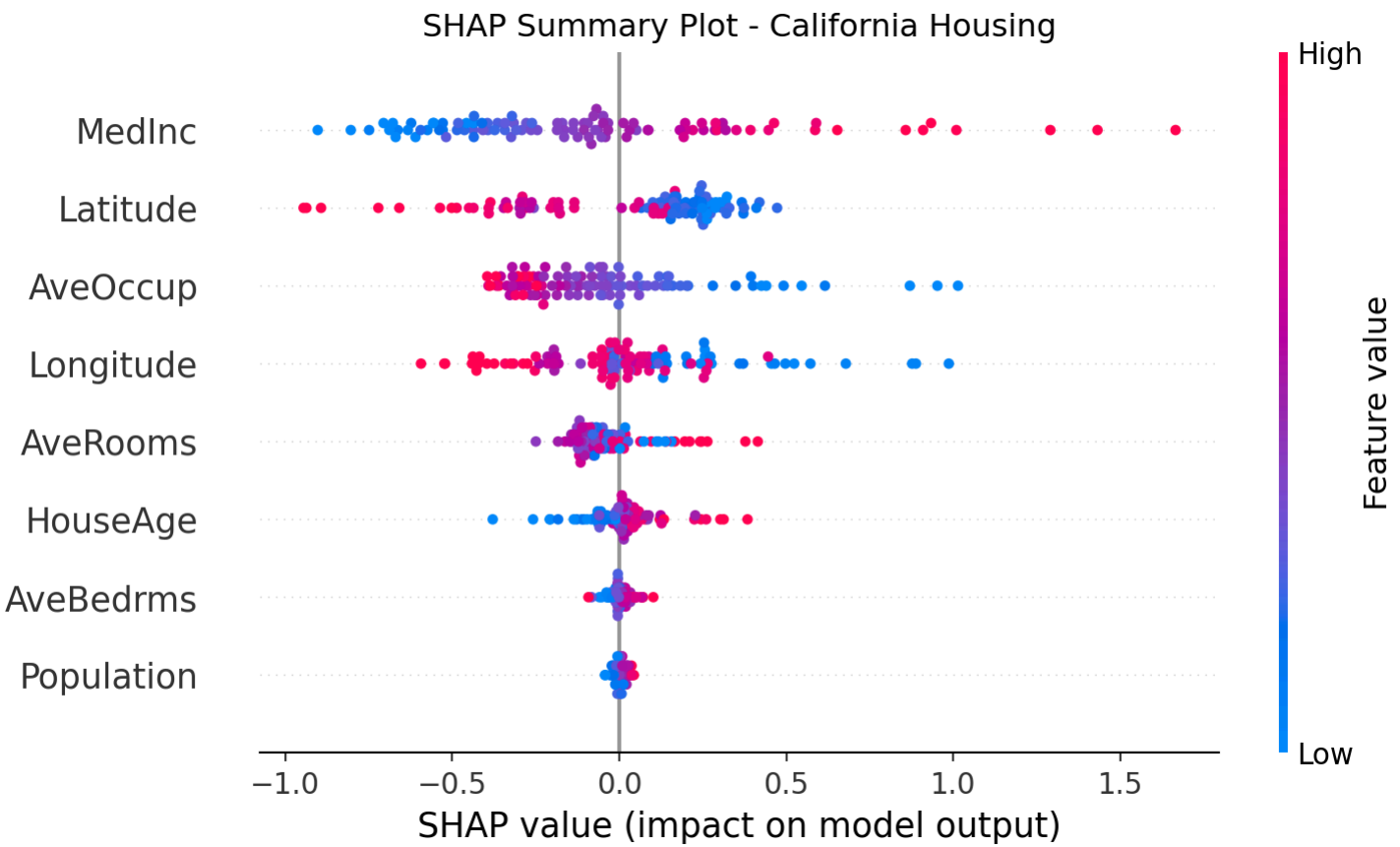
Feature importance:

Average absolute SHAP values:

$$\text{Importance}_j = \frac{1}{n} \sum_{i=1}^{n} \left| \phi_j^{(i)} \right|$$

```python
# Summary plot (combines importance and effects)
# Use subsample for faster rendering
shap_values_sample = explainer.shap_values(X_test[0:100])
X_test_sample = X_test.iloc[:100]

shap.summary_plot(shap_values_sample, X_test_sample,
                  feature_names=X_test.columns.tolist(),
                  show=False)
plt.title('SHAP Summary Plot - California Housing')
plt.tight_layout()
plt.show()
```

SHAP: Global Explanations



SHAP: Global Explanations – continued

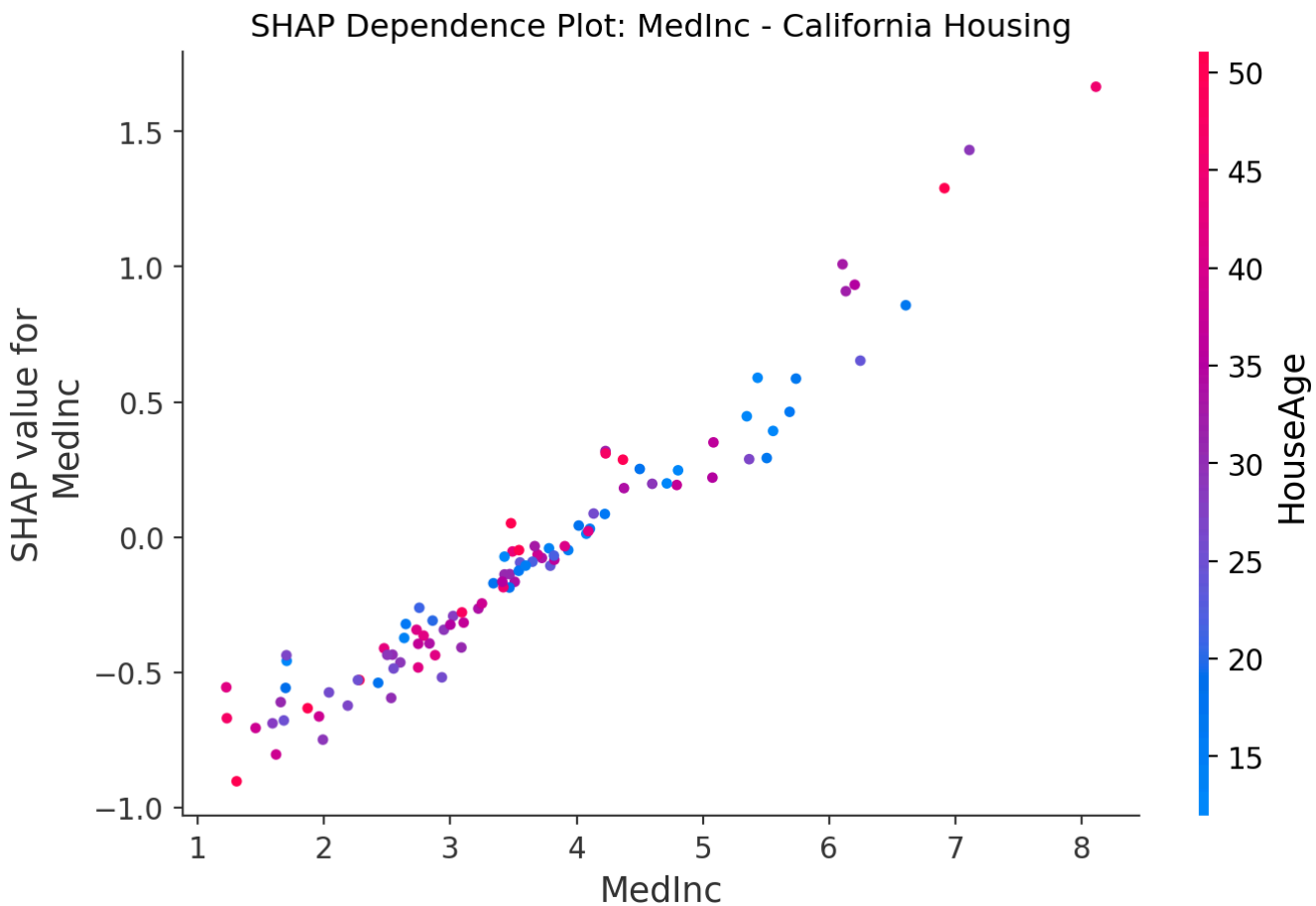
Reading summary plots:

- Features ordered by importance (top to bottom)
- **MedInc** dominates: red (high income) → high SHAP values (increases price)
- **Location** matters: extreme latitudes have strong effects
- Each dot = one census block's SHAP value for that feature

SHAP: Dependence Plots

```
# Dependence plot for MedInc
shap.dependence_plot(
    'MedInc',
    shap_values_sample,
    X_test_sample,
    feature_names=X_test.columns.tolist(),
    show=False
)
plt.title('SHAP Dependence Plot: MedInc - California Housing')
plt.tight_layout()
plt.show()
```

SHAP: Dependence Plots



SHAP: Dependence Plots – continued

Reading dependence plots:

- X-axis: MedInc values
- Y-axis: SHAP value (impact on predicted house value)
- **Observation:** Clear positive trend - higher income → higher SHAP value → higher predictions
- Color shows interaction with HouseAge (automatically selected)
- **Advantage over PDP:** Shows scatter/variance of individual effects, not just average

LIME vs SHAP: Comparison

Aspect	LIME	SHAP
Theory	Heuristic	Game-theoretic (axiomatic)
Consistency	Can be inconsistent	Consistent (properties guaranteed)
Speed	Fast	Slow (except TreeSHAP)
Stability	Less stable	More stable
Scope	Local only	Both local and global
Interpretability	Very intuitive	Requires understanding

Recommendation: Use SHAP when available (especially for tree-based models). Use LIME for quick exploration or when SHAP is too slow.

Conclusion and Key Messages

Model Interpretation Is Not a Solved Problem

Reality check:

- No single method provides complete understanding
- All methods have assumptions and limitations
- Interpretation can be misleading if not done carefully

Current state:

- Active research area
- New methods constantly emerging
- No consensus on “best” approach

Implication: Be cautious and critical when interpreting models.

No Tool Is Perfect

Common issues:

1. **Instability:** Different runs give different results (LIME)
2. **Bias:** Methods may favour certain features (Gini importance)
3. **Correlations:** Misleading results when features are correlated (PDP)
4. **Computational cost:** Some methods are very slow (exact SHAP)

Recommendation: Always understand limitations of the method you're using.

Importance of Using Multiple Methods

Best practice:

- Combine global and local methods
- Compare feature importance across different metrics
- Use both model-agnostic and model-specific tools
- Validate with domain knowledge

Example workflow:

1. Start with permutation importance (global overview)
2. Use SHAP for detailed feature analysis
3. Examine PDPs or ALEs for marginal effects
4. Cross-validate all findings

Do Not Confuse Interpretation with Causality

Critical distinction:

- **Interpretation:** What patterns did the model learn?
- **Causality:** What is the true causal effect?

Example:

- Model shows “ice cream sales” increases “drowning risk”
- Interpretation: ✓ Model uses ice cream sales to predict drowning
- Causality: ✗ Ice cream doesn’t cause drowning (confounded by temperature)

Causal inference requires:

- Randomised experiments or strong assumptions
- Causal models (DAGs, potential outcomes)
- Different tools (do-calculus, instrumental variables)

Warning: Feature importance ≠ causal importance!

Importance of Ethical Framing and Business Context

Interpretation must consider:

1. **Stakeholders:** Who needs to understand the model?
2. **Decisions:** How will interpretations be used?
3. **Fairness:** Are there disparate impacts across groups?
4. **Regulation:** What are legal requirements?
5. **Domain knowledge:** Do results make sense?

Example: Credit risk model

- Can’t just optimise for accuracy
- Must ensure fair treatment across demographics
- Need to provide explanations to applicants
- Must comply with financial regulations

Key principle: Technical interpretability is necessary but not sufficient.

Summary: Main Takeaways

1. **Start with interpretable models** when possible
    - Linear models, decision trees, rule-based models
    - Only add complexity if necessary
2. **Use multiple interpretation methods**
    - Global: permutation importance, PDP/ALE
    - Local: SHAP, LIME,…
    - Cross-validate findings
3. **Be aware of limitations**
    - Every method has blind spots
    - Correlations can mislead
    - Computational constraints matter

Summary: Main Takeaways – continued

4. **Interpretation ≠ Causation**
    - Models learn associations, not causal effects
    - Need causal inference for causal claims
5. **Context matters**
    - Consider stakeholders, fairness, regulations
    - Domain knowledge is essential
    - Ethics should guide technical choices

Final thought: Model interpretation is as much an art as a science. Use tools wisely, question assumptions, and always validate with domain expertise.

References and Further Reading

Books:

- Molnar, C. (2022). *Interpretable Machine Learning* (2nd ed.)
  - Free online: https://christophm.github.io/interpretable-ml-book/
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*

Key Papers:

- Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why Should I Trust You?: Explaining the Predictions of Any Classifier." *KDD*
- Lundberg, S. M., & Lee, S. I. (2017). "A Unified Approach to Interpreting Model Predictions." *NIPS*
- Apley, D. W., & Zhu, J. (2020). "Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models." *JRSS-B*

References and Further Reading – continued

Python Libraries:

- `scikit-learn`: Built-in interpretation tools (`permutation_importance`, `PartialDependenceDisplay`)
- `shap`: https://github.com/slundberg/shap
- `lime`: https://github.com/marcotcr/lime
- `PyALE`: https://github.com/DanaJomar/PyALE

Online Resources:

- Interpretable ML book: https://christophm.github.io/interpretable-ml-book/
- SHAP documentation: https://shap.readthedocs.io/