IMEN574/IDSC574 Programming for Data Science

13. **Interpretable ML**

Spring 2026

Instructor: **Sangwoo Mo**

Today's Agenda

01	**What is Interpretation?**	Definition, good explanations, why it matters
02	**Interpretable Model vs Model Interpretation**	Intrinsic vs post-hoc, accuracy tradeoff
03	**Taxonomy of Methods**	Global/Local, Agnostic/Specific, three categories
04	**Visualization-based Interpretation**	t-SNE, UMAP as interpretation tools
05	**Feature Attribution**	Permutation Importance, PDP/ICE, SHAP, LIME
06	**Example-based Methods**	Counterfactual, Prototypes, Influence Functions
07	**Practice**	Full pipeline, anti-patterns, checklist

What is Interpretation?

Before methods — let’s define what we mean

Defining Interpretation

What does it mean to “understand” a model?

> **Interpretability** is the degree to which a human can understand the cause of a decision.
> *— Miller (2019), adapted from Kim et al. (2016)*



Transparency

Can we inspect the model’s internal mechanics? (e.g., view weights, rules)



Simulatability

Can a human mentally simulate the model’s computation on a given input?



Decomposability

Can each part (feature, parameter) be given an intuitive explanation?

What Makes a Good Explanation?

Three properties every explanation should satisfy



Faithfulness

The explanation must reflect the model’s actual reasoning, not a convenient story. An unfaithful explanation is worse than none.



Stability

Similar inputs should produce similar explanations. If results change every run, you can’t trust them.



Comprehensibility

The explanation must be understandable to its audience — not just technically correct, but communicable.

Why Interpretability Matters

Accuracy alone is not enough — we need to understand why

Trust & Adoption

Stakeholders won't deploy models they can't understand. Interpretability builds confidence in production.

Debugging & Fairness

Detecting spurious correlations, data leakage, bias. Is the model right for the wrong reasons?

Regulation (GDPR)

"Right to explanation": EU law requires explanations for automated decisions affecting people.

Scientific Discovery

In medicine, science: which features actually drive the outcome? Interpretability = insight.

Tip: A 95% accurate model that learns the wrong pattern is dangerous. Interpretability catches this.

Interpretable Model vs Model Interpretation

Two fundamentally different approaches to understanding ML

Interpretable Model vs Model Interpretation

The most important distinction in IML

Interpretable Models

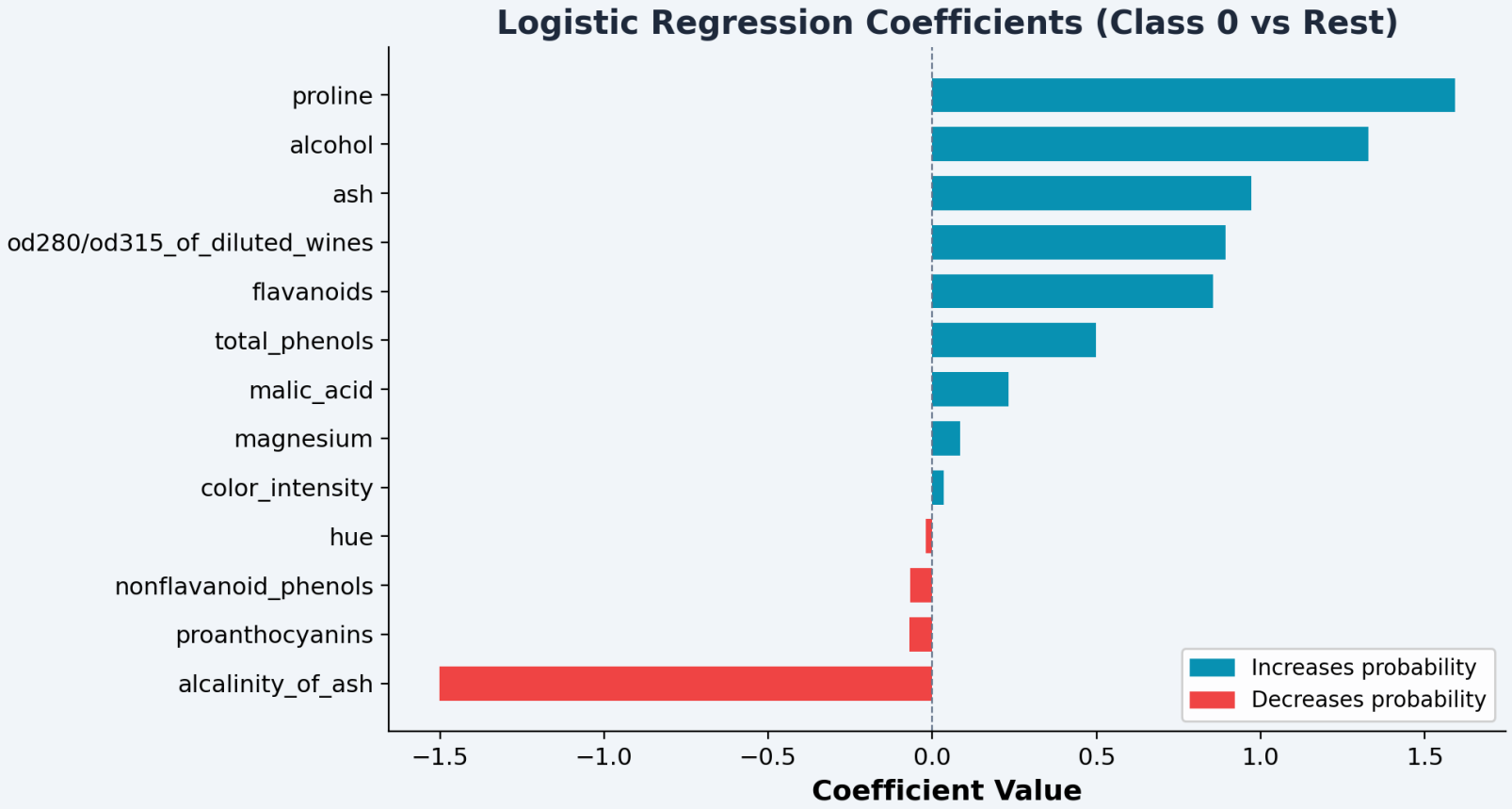
- Model itself is understandable (intrinsic)
- Linear Regression: each coefficient directly shows a feature's effect on the outcome
- Decision Trees: if-then rules, visualizable
- Logistic Regression: coefficients as odds ratios
- GAMs: smooth feature functions
- Pros: No extra tools needed, faithful
- Cons: Often lower accuracy on complex data

Model Interpretation

- Give complex models coefficient-like explanations (post-hoc)
- SHAP: per-feature contribution scores (like coefficients)
- LIME: fit a local linear model to mimic the black box
- PDP/ICE: marginal feature effects
- Counterfactual: what-if scenarios
- Pros: Works on any model (XGBoost, NN, etc.)
- Cons: Approximation, may not reflect true logic

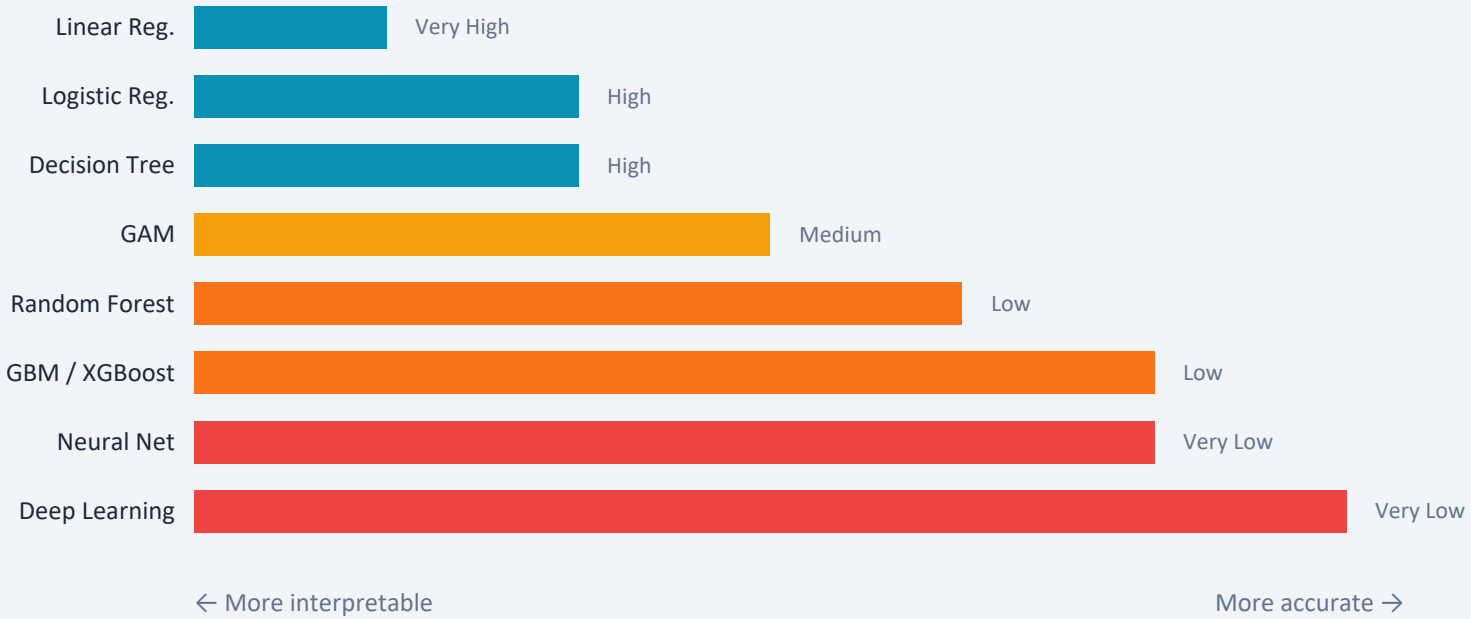
Linear Regression — Coefficients as Explanation

Each coefficient: "If this feature increases by 1 (standardized), prediction changes by β." This is the gold standard of interpretability.



Accuracy vs Interpretability Tradeoff

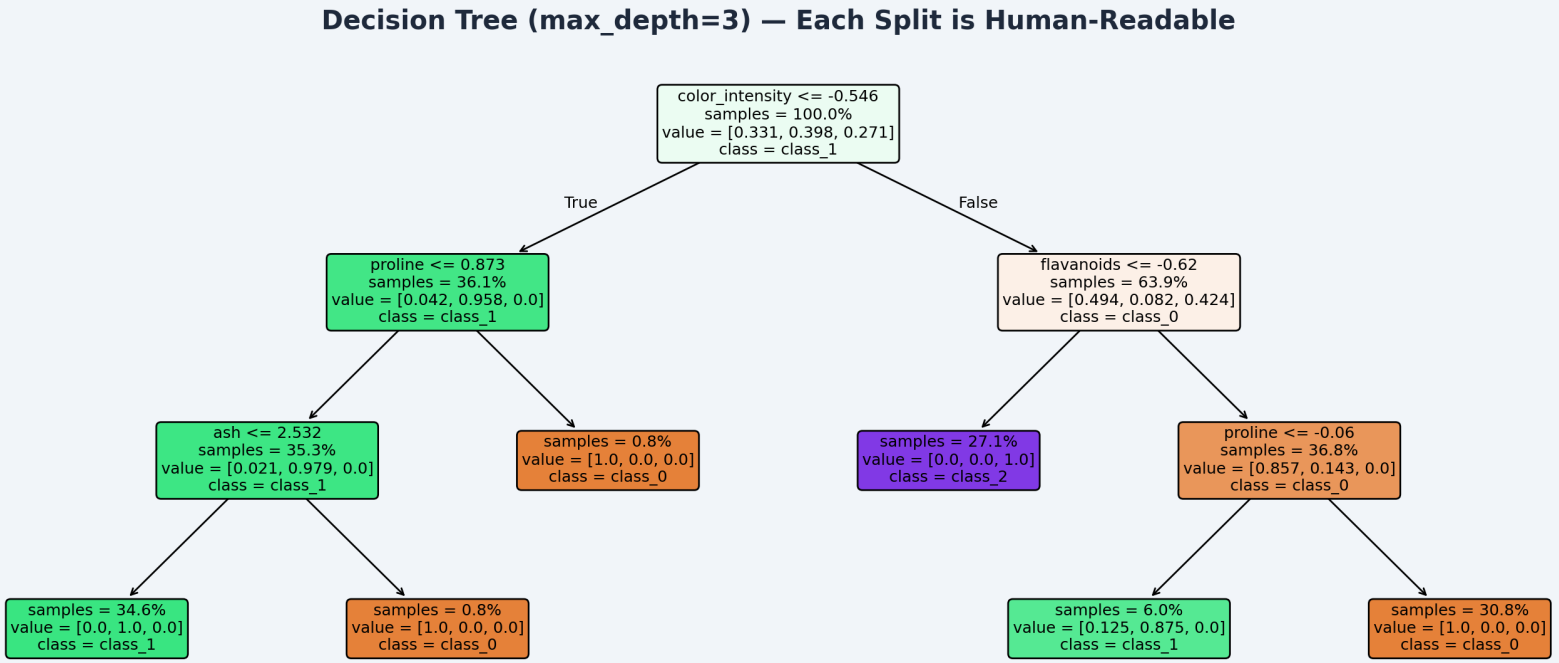
Complex models achieve higher accuracy but are harder to interpret. Post-hoc methods bridge the gap.



The Solution: Use complex models for accuracy, then apply post-hoc methods (SHAP, LIME, PDP) to recover coefficient-like interpretability. Best of both worlds!

Decision Tree — Human-Readable If-Then Rules

Every prediction can be traced root to leaf. Each split is interpretable. No post-hoc method needed.



Taxonomy of Interpretation Methods

Classifying methods by scope, dependency, and approach

IML Taxonomy: Two Key Dimensions

Scope and model dependency define the landscape

Global vs Local

Global: How does the model behave overall? Which features matter on average?

Local: Why did the model make this specific prediction for this instance?

Many tools do both: SHAP provides global summaries and local waterfall plots.

Model-Agnostic vs Specific

Agnostic: Works on any model (SHAP, LIME, PDP, Permutation Importance)

Specific: Built for one family (TreeSHAP, attention, Grad-CAM for CNNs)

Prefer model-agnostic when comparing across different model types.

	Model-Agnostic	Model-Specific
Global	PDP, Perm. Imp., SHAP summary	MDI (trees), Attention (NNs)
Local	SHAP waterfall, LIME, ICE	TreeSHAP, Grad-CAM (CNNs)

Three Categories of Interpretation Methods

Beyond the Global/Local dimension — how methods explain

Visualization

Project high-dimensional data or model internals to a visual space humans can inspect.

Examples:

t-SNE, UMAP, activation visualization

Feature Attribution

Assign a per-feature importance score — like regression coefficients, but for any model.

Examples:

SHAP, LIME, PDP/ICE, Permutation Importance

Example-based

Explain by showing relevant examples: what would need to change, or which training instances are most similar.

Examples:

Counterfactual, Prototypes, Influential instances

Visualization-based Interpretation

Understanding data and models through visual projection

t-SNE & UMAP as Interpretation Tools

Not just dimensionality reduction — these are tools for understanding your model and data

t-SNE

- Preserves local structure (nearby points stay nearby)
- Stochastic — run multiple times for stability
- Key param: perplexity (5–50)
- Visualization ONLY — never use as ML features
- Cannot transform new data (must refit)

UMAP

- Preserves local AND global structure
- Much faster than t-SNE: O(n log n)
- Key params: n_neighbors, min_dist
- Can transform new data (pipeline-ready)
- Usable as preprocessing step

When Visualization Serves Interpretation

Visualization answers questions that feature attribution cannot

Q: Are my classes separable?

Project embeddings with UMAP — if clusters form, the model has learned meaningful representations.

Q: Is there hidden structure?

t-SNE can reveal sub-clusters within a class that you didn't know existed (e.g., subtypes of disease).

Q: Is my model's internal representation sensible?

Visualize intermediate layer outputs — do similar items cluster together?

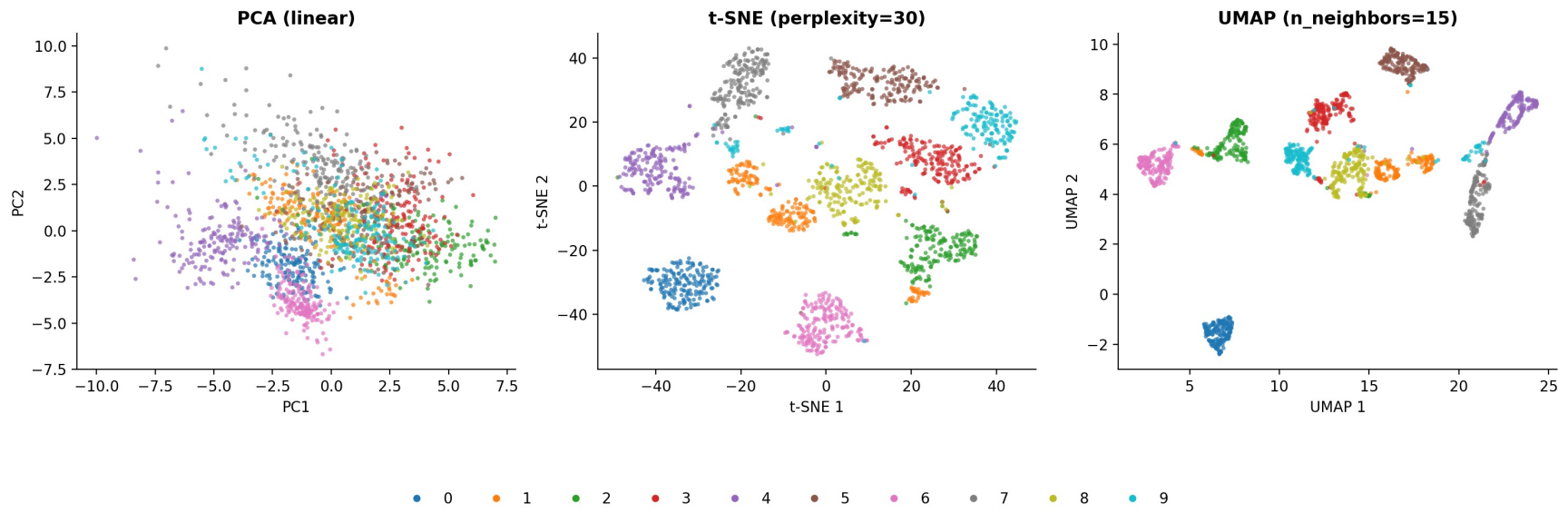
Q: Where does the model fail?

Color points by prediction correctness — misclassified instances in unexpected regions reveal failure modes.

	t-SNE	UMAP
Speed	Slow: $O(n^2)$	Fast: $O(n \log n)$
Global structure	Not preserved	Preserved
New data	Must refit	transform()

t-SNE & UMAP — Results (Lecture 12)

PCA preserves global variance; t-SNE reveals local clusters; UMAP balances local & global structure.



Feature Attribution Methods

Bringing coefficient-like interpretability to any model

Feature Importance — MDI vs Permutation

Two approaches that answer slightly different questions

MDI (Mean Decrease Impurity)

- Built into tree models (feature_importances_)
- How much does splitting on this feature reduce impurity?
- Fast but biased: favors high-cardinality features
- Only works for tree-based models
- Computed on training data (can reflect memorization)

Permutation Importance

- Shuffle one feature at a time; measure accuracy drop
- Model-agnostic: works for any estimator
- Unbiased but slower (multiple evaluations)
- Measures impact on generalization (use on test set!)
- Can be misleading with correlated features

Always use Permutation Importance on the TEST set. MDI is fast for screening, but validate with permutation.

Permutation Importance — Code & Output

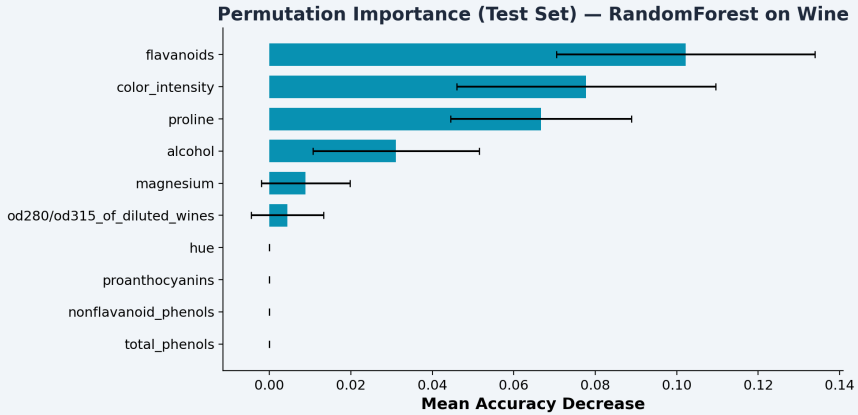
Left: code. Right: result on Wine dataset (test set). Higher bar = more important feature.

```python
from sklearn.inspection import permutation_importance
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, random_state=42).fit(X_tr, y_tr)

# MDI (built-in, training data)
mdi = rf.feature_importances_

# Permutation Importance (on TEST set!)
result = permutation_importance(
    rf, X_te, y_te, n_repeats=10, random_state=42, n_jobs=-1
)
perm = result.importances_mean
```



Correlated Features & Importance Pitfalls

Feature importance can be misleading when features are correlated

The Problem

- If features A and B are correlated:
- MDI splits importance between them randomly
- Permutation: shuffling A doesn’t hurt because B compensates
- Both features appear less important than they are!

Solutions

- Check correlation matrix first
- Group correlated features before importance
- Use SHAP (handles interactions better)
- Consider dropping redundant features (VIF)

High importance ≠ causation. High SHAP value ≠ causal effect. Always validate with domain knowledge.

Partial Dependence Plots (PDP) & ICE

How does the model output change as we vary one feature?

How PDP Works

- For feature j: sweep x_j over a grid
- For each grid value, set ALL samples to that x_j, predict, average
- Shows the marginal effect of feature j
- Assumption: features are independent (limitation)

ICE (Individual Conditional Expectation)

- One line per instance — shows heterogeneity
- PDP = average of all ICE lines
- If ICE lines cross, subgroups have opposite effects
- 2D PDP: vary two features simultaneously for interactions

Common PDP Patterns

Linear

Straight line → linear effect

Threshold

Flat then jumps → cutoff effect

U-shaped

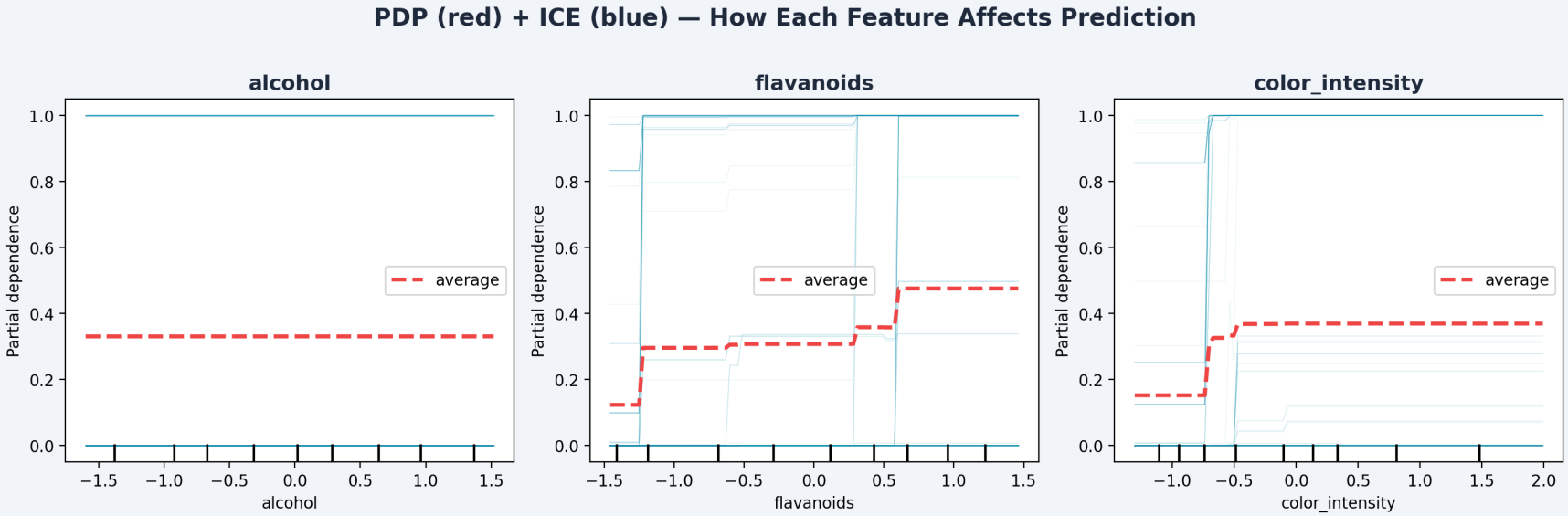
Both ends increase prediction

Flat

No marginal effect → consider removing

PDP + ICE — Visualizing Feature Effects

Red = average effect (PDP). Blue = individual instances (ICE). Crossing ICE lines = interaction effects.



SHAP

SHapley Additive exPlanations — regression coefficients for any model

SHAP — Shapley Values

From cooperative game theory: how much does each feature contribute to THIS prediction?

The Shapley Value Idea: Coefficients via Game Theory

- Consider ALL possible coalitions of other features
- Measure marginal contribution of this feature to each coalition
- Average over all coalitions = Shapley value

$$f(x) = E[f(x)] + SHAP_1 + SHAP_2 + \ldots + SHAP_p$$

Unique Properties

Efficiency: values sum to prediction

Symmetry: equal features get equal SHAP

Dummy: irrelevant features get 0

Additivity: consistent across models

TreeSHAP (Fast for Trees): Exact Shapley values in O(TLD) for tree models (RF, XGBoost, LightGBM). KernelSHAP: model-agnostic but slower (samples coalitions). The only IML method satisfying all four Shapley axioms.

SHAP — Code & Beeswarm Plot

Each dot = one sample. X = SHAP value (impact). Color = feature value (red=high, blue=low).

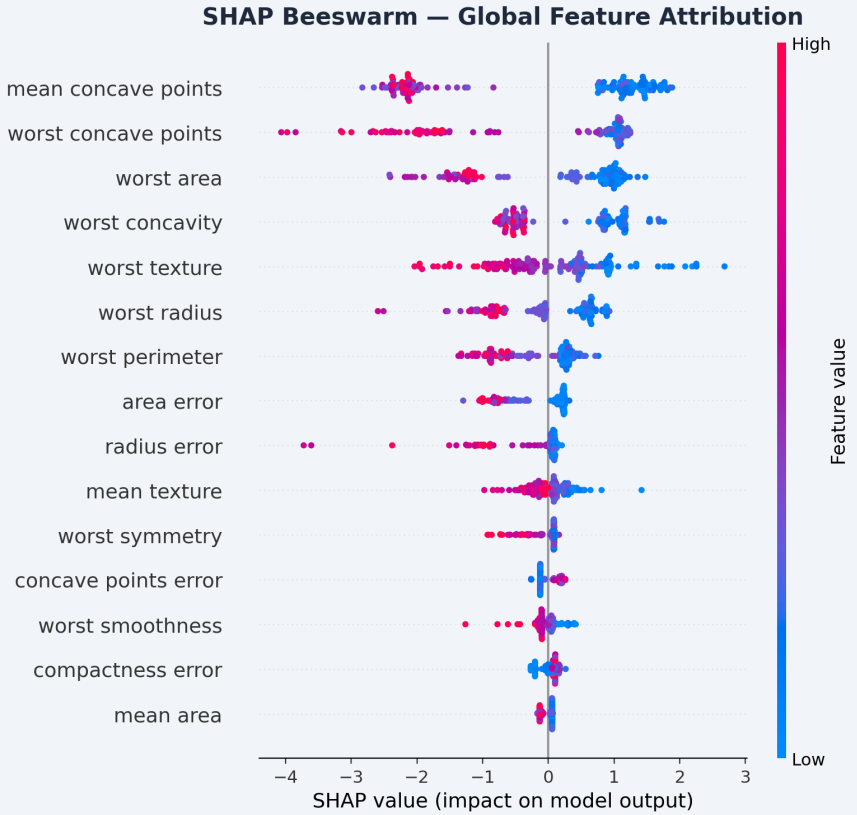
```python
import shap

model = GradientBoostingClassifier(n_estimators=100).fit(X_tr, y_tr)

# TreeExplainer (fast, exact for tree models)
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_te)

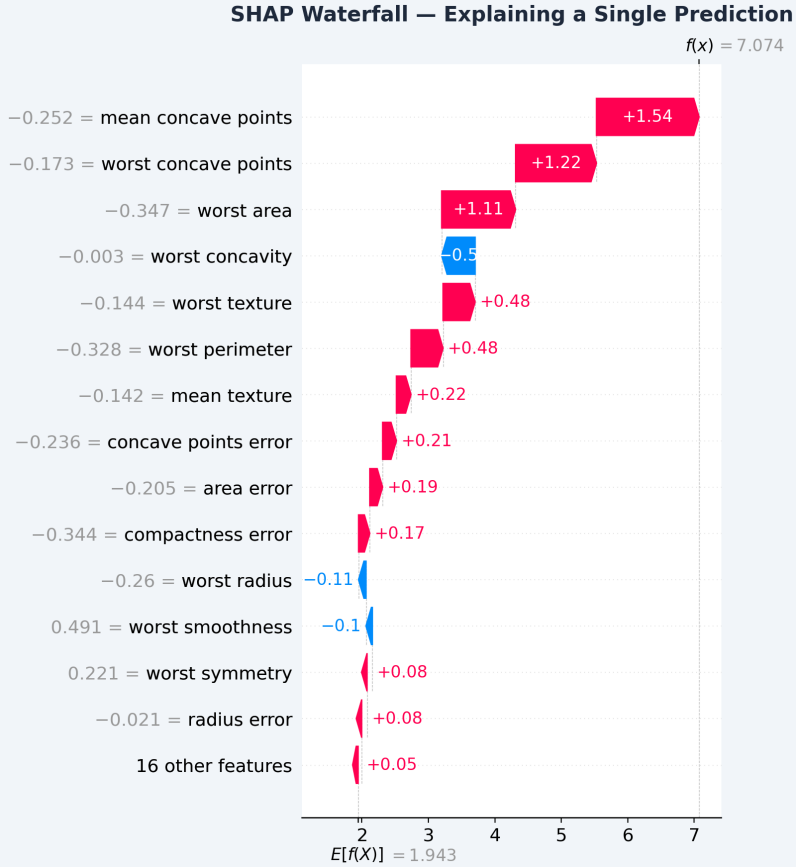
# Global summary — beeswarm plot
shap.summary_plot(shap_values, X_te, feature_names=feat_names)

# Local explanation — waterfall for one instance
shap.plots.waterfall(shap.Explanation(
    values=shap_values[0], base_values=explainer.expected_value,
    data=X_te[0], feature_names=feat_names
))
```



SHAP Waterfall — Explaining One Prediction

From base value E[f(x)], each feature pushes prediction up (red) or down (blue). Like reading coefficients for one instance.



SHAP — Real-World Interpretation

Example: Predicting loan default. How SHAP explains model decisions.

```
model = XGBClassifier(n_estimators=200)

model.fit(X_train, y_train)

explainer = shap.TreeExplainer(model)

shap_values = explainer(X_test)


# Why was this loan denied? (instance 42)

shap.plots.waterfall(shap_values[42])
```

Stakeholder Interpretation

"This loan was denied primarily because of a high debt-to-income ratio (+0.32) and large loan amount (+0.28), despite a good credit score (-0.45 pushes toward approval)."

```
Waterfall result: debt_to_income (+0.32), loan_amount (+0.28), credit_score (-0.45), employment_years (-0.12)

Global bar: credit_score > debt_ratio > income > employment_years > loan_amount
```

LIME

Local Interpretable Model-Agnostic Explanations

LIME — Core Idea

Explain any model's prediction by fitting a simple model locally

How LIME Works

1. Pick instance to explain
2. Generate perturbations (randomly modify nearby features)
3. Get black-box predictions for all perturbations
4. Weight perturbations by proximity to original
5. Fit simple model (linear regression) on weighted samples
6. Simple model's coefficients = local feature importance

Strengths

- Model-agnostic (any model)
- Intuitive output (feature weights)
- Works on tabular, text, images

Limitations

- Unstable (different runs → different results)
- Local only (no global view)
- Perturbation quality matters a lot

Key insight: any complex model looks approximately linear if you zoom in close enough. LIME exploits this — its coefficients are local feature attributions.

LIME — Code & Local Linear Coefficients

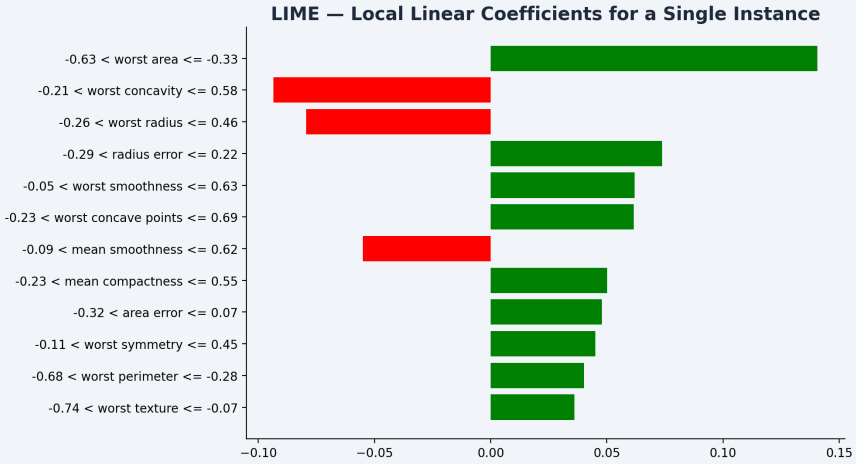
LIME fits a local linear model around this instance. Its coefficients = local feature attributions.

```python
from lime.lime_tabular import LimeTabularExplainer

explainer = LimeTabularExplainer(
    X_train, feature_names=feat_names,
    class_names=['Not Default', 'Default'], mode='classification'
)

# Explain single instance — fits local linear model
exp = explainer.explain_instance(
    X_test[0], model.predict_proba,
    num_features=10, num_samples=1000
)

# View local coefficients (= feature attributions)
exp.show_in_notebook()
print(exp.as_list())  # [(feature, coefficient), ...]
```



SHAP vs LIME — When to Use Which?

Both have strengths; often use both in practice

	SHAP	LIME
Theory	Game theory (Shapley axioms)	Local linear surrogate
Scope	Global + Local	Local only
Consistency	Deterministic (for trees)	Stochastic (varies per run)
Speed	TreeSHAP fast / KernelSHAP slow	Moderate
Output style	Additive attributions (sum to pred)	Feature conditions (if-then)
Best for	**Rigorous analysis, model debugging**	**Quick explanation, non-tech audience**

Best practice: SHAP for thorough analysis (development), LIME for quick explanations (stakeholder communication).

Example-based Methods

Counterfactual • Prototypes • Influence Functions

Counterfactual Explanations

"What would need to change for the model to predict differently?"

The Idea

Instead of "why was this predicted?", ask "what's the smallest change that flips the prediction?"

Example: *"Your loan was denied. If your income were $5k higher OR your debt-to-income ratio were below 0.4, it would have been approved."*

The most natural form of explanation for end users — contrastive and actionable.

Key Properties

- Model-agnostic
- Actionable: tells user what to change
- Human-friendly (contrastive)
- Multiple valid counterfactuals exist
- Libraries: DiCE, Alibi

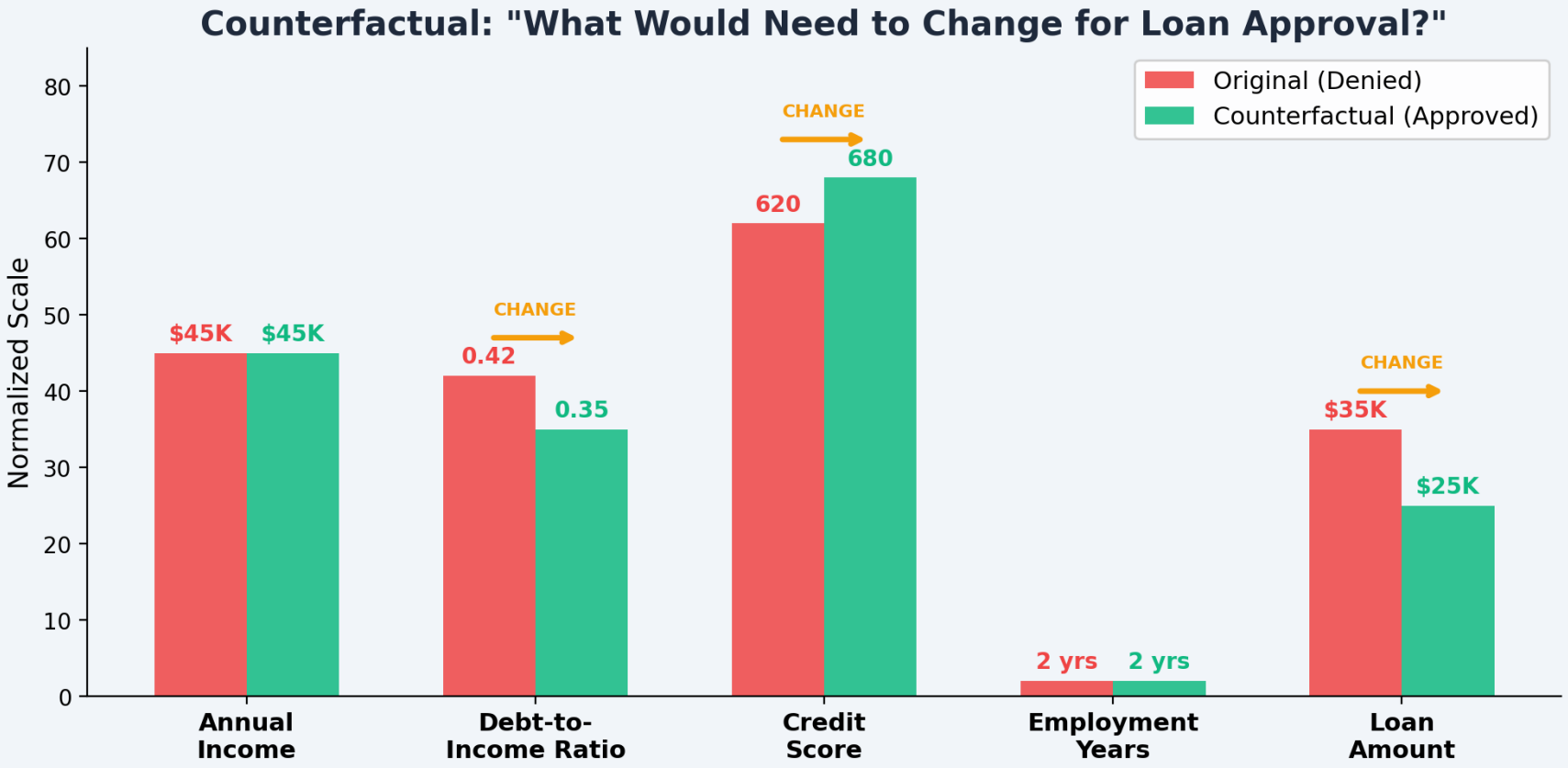
```
# pip install dice-ml
import dice_ml

d = dice_ml.Data(dataframe=df,
    continuous_features=cont_feats, outcome_name='default')
m = dice_ml.Model(model=model, backend='sklearn')
exp = dice_ml.Dice(d, m, method='random')

# Generate 3 counterfactuals for instance 42
cf = exp.generate_counterfactuals(
    X_test.iloc[[42]], total_CFs=3, desired_class="opposite")
cf.visualize_as_dataframe()
```

Counterfactual — "What Would Need to Change?"

Minimal changes to flip the prediction. Arrows show required adjustments and their magnitude.



Prototypes & Exemplars

"This prediction is similar to these training examples"

The Idea

Prototype: A representative instance that summarizes a cluster or class. "Typical approved loan looks like this."

Criticism: An instance that is NOT well-represented by existing prototypes. Highlights edge cases and diversity.

Nearest exemplar: "Your case is most similar to these past cases, which were approved/denied." Directly interpretable via comparison.

Why Example-based?

- Humans naturally reason by analogy
- No feature attribution needed
- Works with any data type (tabular, image, text)
- Especially powerful for doctors, judges, loan officers
- MMD-critic (Kim et al., 2016) for prototype selection

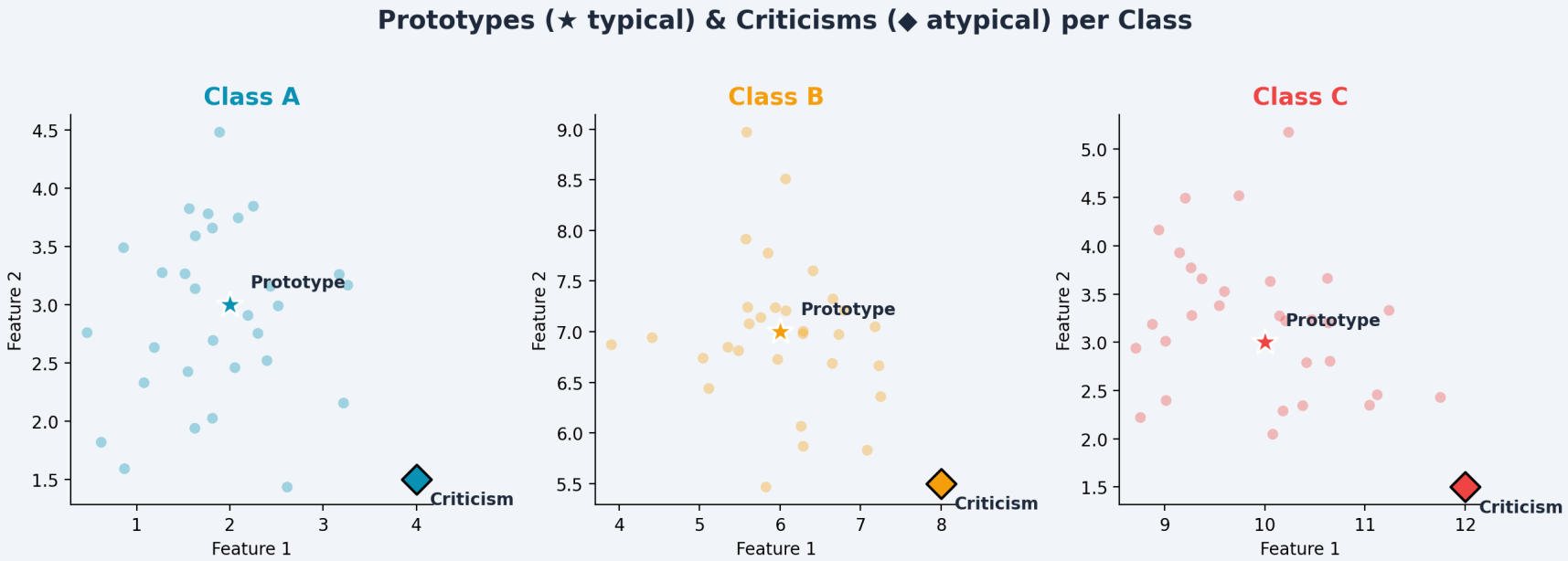
```python
from sklearn.neighbors import NearestNeighbors

# Find k nearest training examples to a test instance
nn = NearestNeighbors(n_neighbors=5, metric='euclidean')
nn.fit(X_train_scaled)
dists, idxs = nn.kneighbors(X_test_scaled[[42]])

# Show similar past cases and their outcomes
similar_cases = X_train.iloc[idxs[0]]
similar_outcomes = y_train.iloc[idxs[0]]
print("Similar cases:", similar_cases)
print("Their outcomes:", similar_outcomes.values)
```

Prototypes & Criticisms — Explaining by Example

Prototypes (★) = most representative instances per class. Criticisms (◆) = atypical edge cases.



Influence Functions

"Which training points most influenced this prediction?"

The Idea (Koh & Liang, 2017)

"If we removed training point z from the dataset, how much would the prediction on test point x change?"

Uses second-order approximation (Hessian) to estimate the effect without retraining.

Practical uses:

- Debug mislabeled training data
- Understand which examples shaped a specific prediction
- Data valuation: which training points matter most?

vs Nearest Exemplars

Exemplar: similar in input space (feature distance)

Influence: similar in effect on the model (gradient space)

A training point can be highly influential without being "similar" in feature space — it shaped the decision boundary.

Libraries: captum (PyTorch), influence-functions, TracIn

Caveat: Influence functions require computing/approximating the Hessian — expensive for large models. TracIn (Pruthi et al., 2020) offers a scalable gradient-based alternative.

Influence of training point z on test prediction x:

$$I(z, x) = -\nabla_\theta L(x, \theta)^T \cdot H^{-1}_\theta \cdot \nabla_\theta L(z, \theta)$$

Practice

Full pipeline, end-to-end example, and best practices

Full Interpretability Pipeline

A systematic workflow for explaining any ML model

1 Train Model

Fit your model normally. Save train/test split.

2 Permutation Importance

Quick global ranking: which features does the model rely on? (test set)

3 PDP / ICE

Understand shape of each important feature's effect. Check nonlinearity.

4 SHAP Analysis

Global summary (beeswarm) + local explanations (waterfall) for key instances.

5 LIME (optional)

Intuitive explanations for stakeholder communication.

6 Validate & Document

Cross-check with domain knowledge. Document findings. Flag bias risks.

End-to-End Example — Setup & Feature Importance

Step 1: Load data, train model, run permutation importance on the test set.

```
import numpy as np, pandas as pd, shap
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.inspection import permutation_importance

df = pd.read_csv("credit_data.csv")
X, y = df.drop('default', axis=1), df['default']
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.25, random_state=42)

scaler = StandardScaler().fit(X_tr)
X_tr_s, X_te_s = scaler.transform(X_tr), scaler.transform(X_te)

model = GradientBoostingClassifier(n_estimators=200, max_depth=4)
model.fit(X_tr_s, y_tr)

# Step 1: Permutation Importance (test set!)
perm = permutation_importance(model, X_te_s, y_te, n_repeats=10, n_jobs=-1)
```

End-to-End Example — SHAP + LIME

Step 2: SHAP for global+local analysis. Step 3: LIME for stakeholder communication.

```
# Step 2: SHAP (global + local)
explainer = shap.TreeExplainer(model)
shap_values = explainer(X_te_s)

shap.plots.beeswarm(shap_values)          # Global overview
shap.plots.waterfall(shap_values[42])     # Why this instance?

# Step 3: LIME for stakeholder explanation
from lime.lime_tabular import LimeTabularExplainer
lime_exp = LimeTabularExplainer(
    X_tr_s, feature_names=feat_names,
    class_names=['Approved', 'Denied'], mode='classification'
)
exp = lime_exp.explain_instance(
    X_te_s[42], model.predict_proba, num_features=8
)
exp.show_in_notebook()
```

Common Anti-Patterns

Explaining Training Data

Running importance/SHAP on training set measures memorization. Always use test set.

Cherry-Picking Explanations

Showing only instances where the model looks reasonable. Report failures too.

Ignoring Feature Correlations

Importance splits among correlated features. Check VIF and group related features.

Over-Interpreting LIME

LIME is stochastic and local. Don’t generalize one instance’s explanation to the whole model.

Skipping Validation

Explanations without domain expert review are dangerous. Always sanity-check with humans.

Assuming Causation

Feature importance shows what the model uses, not what causes the outcome. Be careful with language.

Interpretability Checklist

Before deploying, ensure you can answer these questions

☐ Which features does the model rely on most?	Permutation Importance, SHAP bar
☐ How does each important feature affect predictions?	PDP, ICE, SHAP dependence
☐ Why did the model make this specific prediction?	SHAP waterfall, LIME
☐ Are there spurious correlations or data leakage?	Check if top features make domain sense
☐ Is the model fair across demographic groups?	SHAP cohort analysis, group metrics
☐ Do different explanation methods agree?	Compare SHAP, LIME, PDP
☐ Can you explain the model to a non-technical person?	LIME, SHAP force plot, plain language
☐ Have domain experts validated the explanations?	Human review, sanity checks

Fairness & Bias Detection with IML

Use SHAP to audit models for bias before deployment

Bias Red Flags

Protected attributes (gender, race) in top SHAP features → Direct bias

Proxy features (zip code for race) appear important → Indirect bias

SHAP patterns differ across groups → Differential treatment

Model performance unequal across groups → Check fairness metrics

Limitations to Remember

Correlation ≠ Causation: SHAP shows model usage, not causal effect

Explanations are model-specific: a wrong model gives wrong explanations

Adversarial attacks can fool SHAP/LIME (Slack et al., 2020)

Always combine IML with standard fairness metrics (demographic parity, equalized odds)

Key Takeaways

- Interpretation = explaining model predictions in human-understandable terms. Faithfulness, stability, comprehensibility.
- Interpretable models (linear, tree) are inherently understandable; post-hoc methods explain black boxes.
- Three pillars: Visualization (t-SNE/UMAP) / Feature Attribution (SHAP, LIME) / Example-based (counterfactual, prototypes).
- SHAP: theoretically grounded (Shapley axioms), global + local, gold standard. Use TreeSHAP for tree models.
- LIME: quick local explanations for stakeholders. Counterfactuals: actionable, human-friendly alternatives.
- Prototypes explain by analogy; Influence Functions trace predictions back to training data.
- Different methods answer different questions — combine them for robust understanding.
- High importance ≠ causation. Always validate explanations with domain expertise.