Interpreting Machine Learning Models with SHAP: Explaining Predictions with Shapley Values

---

Introduction: Why Model Interpretability Matters

In modern machine learning, particularly in domains like healthcare, finance, and criminal justice, accuracy alone is no longer enough. Today's practitioners must be able to **explain how their models make decisions**, especially when those decisions affect people's lives.

Machine learning models — especially complex ones like Random Forests, Gradient Boosting, and Neural Networks — are often treated as **black boxes**. They can produce highly accurate predictions but offer little transparency about *why* a particular prediction was made. This raises ethical, legal, and practical concerns.

SHAP(SHapley Additive exPlanations) is a viable approach that sheds light on these opaque models by presenting a unified mathematically grounded framework for feature attribution. SHAP builds on Shapley values, a concept from cooperative game theory that determines the contribution of each feature to a final prediction. (Lundberg & Lee, 2017).

Unlike traditional feature importance methods that simply show correlation with the target, SHAP **assigns responsibility** for individual predictions — enabling model debugging, trust-building, and compliance with regulations like GDPR.

Analogy: The Doctor and the Diagnosis

Imagine visiting a doctor and receiving a diagnosis: *"You're at high risk for heart disease."* Naturally, you ask: *"Why?"*

A responsible doctor might say:

- Your **cholesterol is high**
- You **smoke regularly**
- Your **family history** shows genetic risk

Now imagine if the doctor just said "the model said so" — would you trust the treatment?

That's how machine learning models often behave — making decisions without explanation. SHAP is the **doctor's explanation layer**: it breaks down a prediction and assigns credit (or blame) to each contributing factor.

Just as we wouldn't accept a diagnosis without rationale, we shouldn't accept ML predictions without interpretability.

Where and Why to Use SHAP

SHAP is most useful when you are working with:

- **Complex non-linear models** (e.g., XGBoost, LightGBM, CatBoost, Random Forests)
- **Regulatory or compliance-heavy industries** where you must explain individual predictions (e.g., banking, healthcare)
- **Bias detection tasks** where understanding model fairness is critical
- **Model debugging** — e.g., discovering that a model relies too much on irrelevant features

Use Case	How SHAP Helps
Loan Approval	Explains why a customer is denied (e.g., low income)
Clinical Diagnosis	Identifies key factors in disease prediction
Customer Churn	Highlights which behaviors lead to attrition
Fraud Detection	Details which transactions are flagged and why

---

Limitations and Considerations of SHAP

While SHAP is powerful, it's important to acknowledge its **limitations** — a critical lesson for master's students building production-level systems.

🔹 1. Computational Cost

Calculating exact Shapley values is **exponentially expensive** with respect to the number of features. Although SHAP uses approximations (like TreeSHAP), it can still be slow for very large datasets or models with thousands of features.

SHAP has polynomial time complexity even with optimizations, which can be prohibitive in real-time applications (Lundberg et al., 2020).

🔹 2. Interpretability ≠ Causality

SHAP explains what the model *saw* and used — not necessarily what is *causally true*. A high SHAP value doesn't mean that the feature caused the outcome in the real world. Correlation is not causation.

For example, if ZIP code strongly correlates with loan default, SHAP might highlight it — even though using ZIP code may introduce bias or violate fairness guidelines.

🔹 3. Assumes Model Additivity

SHAP's math assumes that feature contributions can be linearly added. While this works well for many models, **non-additive interactions** (e.g., decision boundaries in deep networks) might not always be faithfully represented.

🔹 4. Visual Complexity

SHAP summary plots and force plots can become **hard to interpret** when there are too many features or overlapping effects. In such cases, simplifying the feature space or using domain-specific groupings is essential.

What Makes SHAP Different?

Let's briefly contrast SHAP with traditional feature importance:

Method	Description	Global or Local?	Accuracy
Coefficients (Linear)	Weights in regression	Global	Simple, but limited to linear models
Feature Importance (Tree-based)	Mean decrease in impurity	Global	Fast, but biased by cardinality
Permutation Importance	Measures drop in score when feature is shuffled	Global	More accurate but slow
SHAP	Based on cooperative game theory, explains each prediction	Both	Most faithful + interpretable

---

Exploratory Data Analysis (EDA): Understanding Before Modeling

Before we train any model — especially one we plan to interpret deeply — we must understand the structure, distribution, and potential issues in the data. This is where **(EDA)** plays a vital role. As John Tukey (1977) famously stated, *"The greatest value of a picture is when it forces us to notice what we never expected to see."*

We conduct exploratory data analysis for this section on the Breast Cancer Wisconsin dataset, which is one of the most widely used medical datasets in the UCI repository and is accessible through scikit-learn. The dataset consists of 569 samples and 30 numerical features reflecting the characteristics of cells' nuclei in digitized images from tissue of breast masses. The binary target (0=malignant, 1=benign) reflects the classification objective.

Data Preview: Head and Summary Statistics

We begin with a `head()` command to preview the first five rows, giving us an immediate sense of scale and feature names. Following this, `describe()` generates summary statistics such as **mean, standard deviation, min, and quartiles** for each numeric feature.

This overview tells us, for instance:

- Some features (e.g., `mean radius`, `mean area`) have larger value ranges, potentially dominating models if not standardized
- Feature scales vary widely, confirming the need for preprocessing

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	17.33	184.60	2019.0	0.1622	0.6656
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	158.80	1956.0	0.1238	0.1866
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	152.50	1709.0	0.1444	0.4245
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	98.87	567.7	0.2098	0.8663
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	152.20	1575.0	0.1374	0.2050

5 rows × 31 columns

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	te
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.0
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	...	25.6
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...	6.1
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...	12.0
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...	21.0
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...	25.4
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...	29.7
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...	49.5

8 rows × 31 columns

Figure 1 Initial rows and descriptive statistics give insight into value distributions, outliers, and scaling issues.

❓ Missing Values Check

Before modeling, it is critical to ensure that no missing data will disrupt calculations or skew model learning. Using `isnull().sum()`, we check for **null or NaN values** across all features. In this dataset, we confirm there are **no missing values**, making it ideal for direct modeling.

```
X.isnull().sum()
```

mean radius	0
mean texture	0
mean perimeter	0
mean area	0
mean smoothness	0
mean compactness	0
mean concavity	0
mean concave points	0
mean symmetry	0
mean fractal dimension	0
radius error	0
texture error	0
perimeter error	0
area error	0
smoothness error	0
compactness error	0
concavity error	0
concave points error	0
symmetry error	0
fractal dimension error	0
worst radius	0

Class Distribution

Class imbalance can severely affect the performance of models and mislead accuracy metrics. A `countplot()` reveals that the dataset is slightly imbalanced: **62.7% benign** and **37.3% malignant**. This mild skew does not require aggressive resampling but reminds us to focus on **precision and recall**, especially for the minority malignant class.

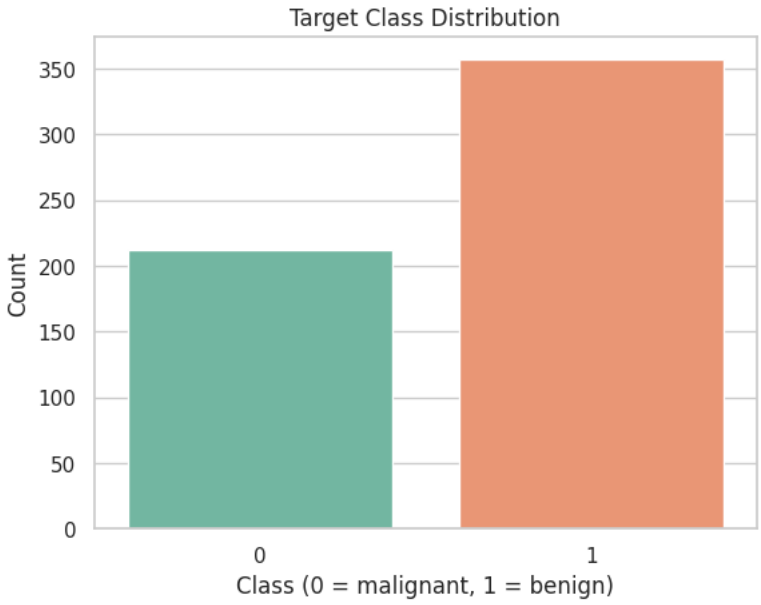


Figure 2 Class distribution bar plot: Slight imbalance favoring benign cases (class 1).

Feature Correlation Heatmap

We use a correlation matrix and heatmap to identify **linear relationships between features**. For instance, `mean radius` is highly correlated with `mean perimeter`, `area`, and `concavity`. This insight is essential for:

- Understanding multicollinearity in linear models
- Anticipating feature redundancy
- Interpreting SHAP results where correlated features may share credit

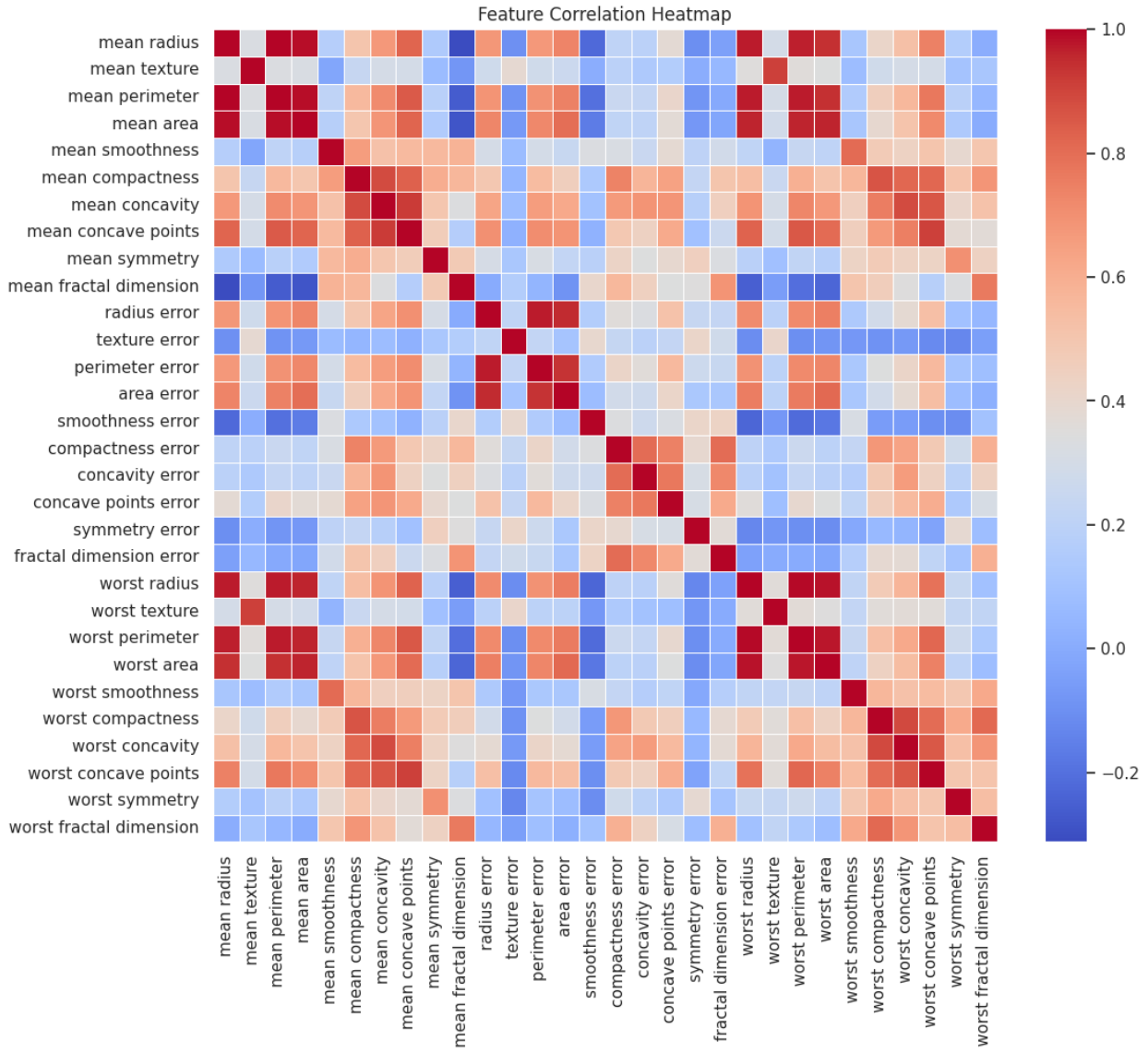


Figure 3 Feature correlation heatmap identifies clusters of interrelated variables, useful for feature engineering and model interpretation.

Data Preparation with Standardization

Before training the model, we perform **feature scaling using StandardScaler**, which standardizes features by removing the mean and scaling to unit variance. This ensures that:

- All features contribute equally to distance-based models (e.g., KNN, SVM)
- SHAP explanations are numerically stable and consistent
- XGBoost's internal handling of features doesn't disproportionately favor large-scale inputs

We split the dataset into **70% training and 30% testing**, using stratified sampling to preserve class ratios across splits.

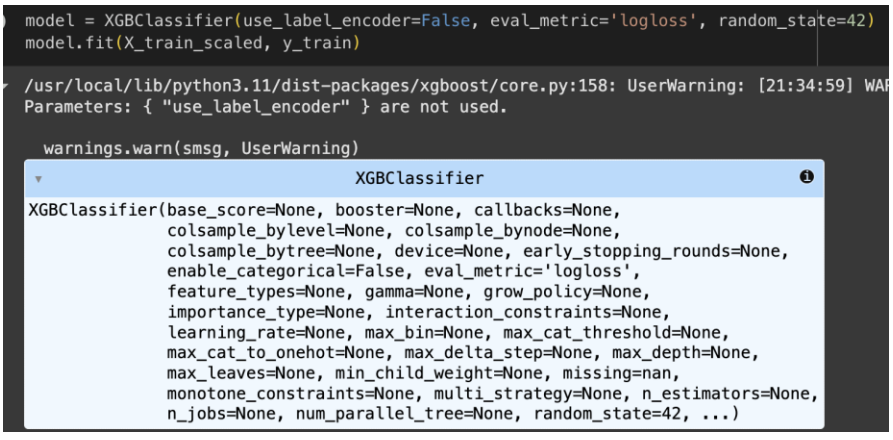
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Tip: Always apply scaling **after splitting** and **fit the scaler only on training data** to prevent data leakage.

Model Training + SHAP Explainability: Full Walkthrough

Step 5: Train the XGBoost Classifier

We begin by training an **XGBoost Classifier**, which is known for its high performance and compatibility with SHAP explanations through its built-in TreeSHAP algorithm (Lundberg et al., 2020). We use `use_label_encoder=False` and `eval_metric='logloss'` to suppress warnings and provide clean evaluation. The classifier is trained on the standardized training data (`X_train_scaled`) to ensure feature consistency.



This model, once trained, serves as our black-box predictor — and we now seek to explain how it arrived at each decision using SHAP values.

Step 6: Shape Explainer initiated:

SHAP uses a model-specific explainer tailored for tree-based models like XGBoost. The `shap.Explainer()` wrapper detects the appropriate algorithm automatically (TreeSHAP in this case), and computes **Shapley values** for each sample in the test set.

```
explainer = shap.Explainer(model, X_train_scaled)
shap_values = explainer(X_test_scaled)
```

Each `shap_value` represents the **contribution of each feature** to the prediction, **relative to the expected value** (i.e., the average model output). This allows us to explain predictions at both global and local levels — a key strength of SHAP (Lundberg & Lee, 2017).

Step 7: SHAP Summary Bar Plot (Global Importance)

The first visual tool we use is the **summary bar plot**, which aggregates the **mean absolute SHAP value** of each feature. This shows how much each feature contributes, on average, to model predictions — regardless of direction.

```
shap.summary_plot(shap_values, X_test, plot_type='bar')
```

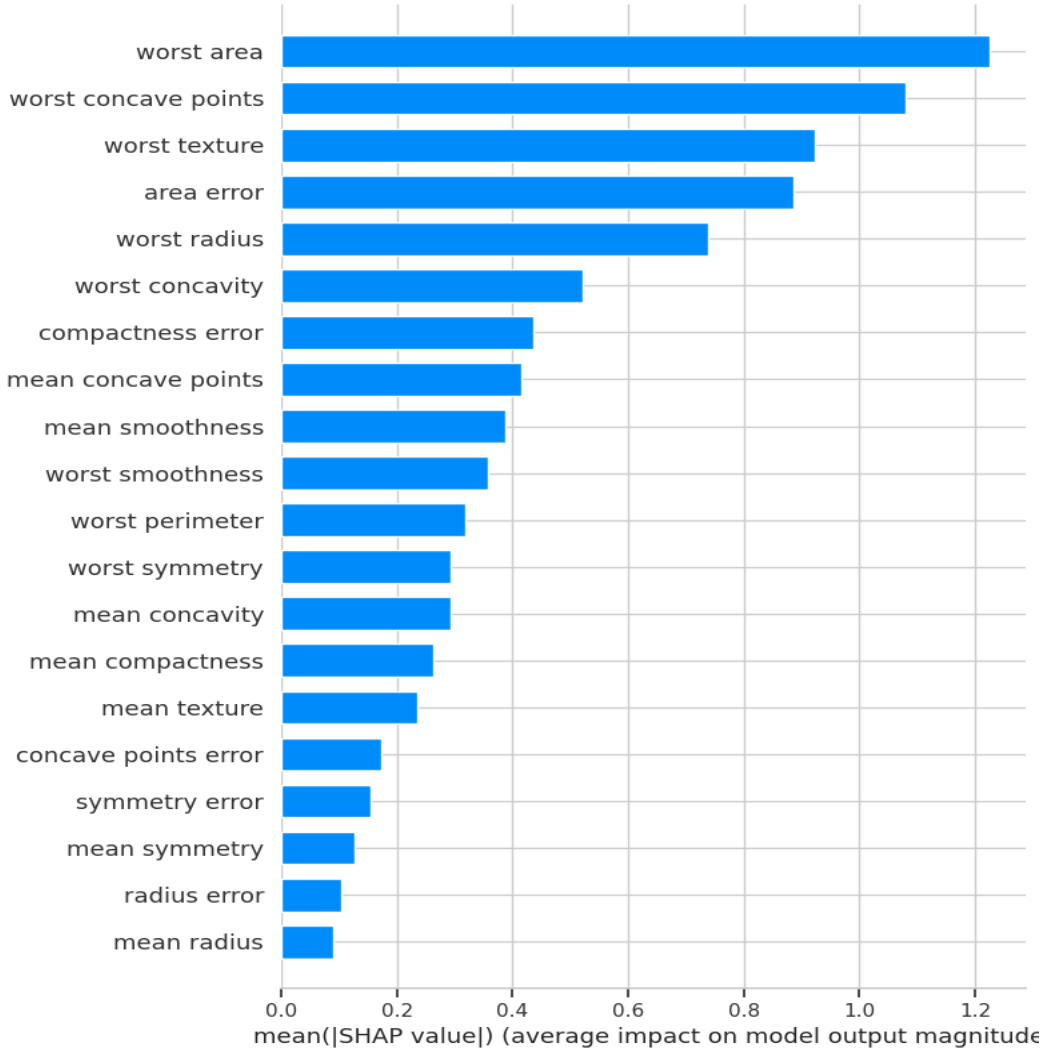


Figure 4 SHAP summary bar plot showing average absolute impact of each feature on the model output.

From this, students can identify the most influential features overall — such as "worst concave points" or "mean area" — and compare them with feature importances from other methods like Gini importance.

Step 8: SHAP Beeswarm Plot (Distribution and Direction)

While the bar plot shows magnitude, the **beeswarm plot** adds depth by showing **direction and distribution**. Each point is a feature's SHAP value for a sample, color-coded by the feature's value (e.g., high = red, low = blue). This lets us observe how **low or high values push predictions up or down**.

```
shap.summary_plot(shap_values, X_test)
```

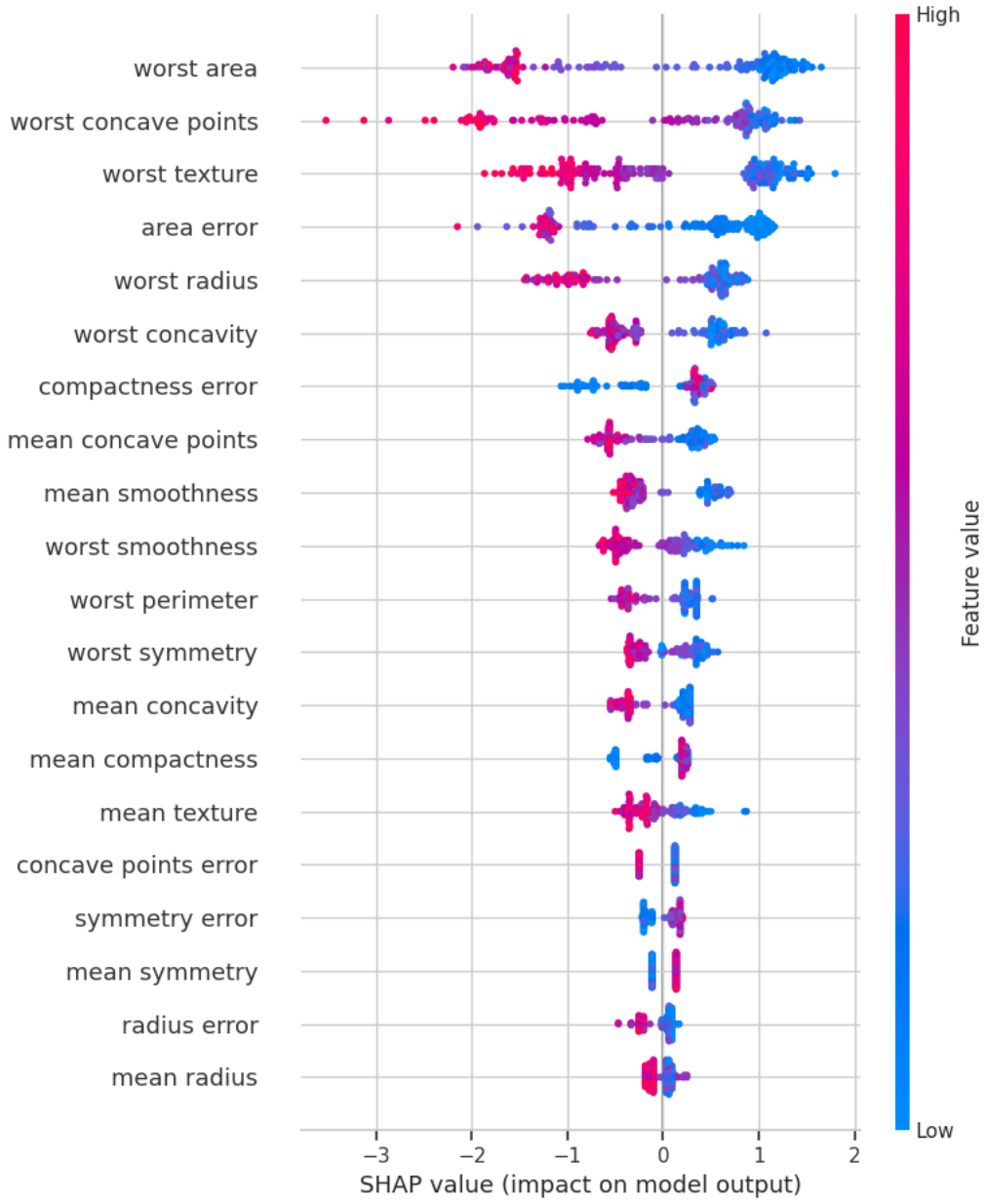


Figure 5 SHAP beeswarm plot showing the distribution and direction of feature effects across all test samples.

This plot is extremely informative: students can see not just what is important, but how it behaves.

Step 9: Individual Prediction by Force plot

Lastly, we create a **force plot** for a single prediction. This visual decomposes the predicted probability into a **baseline (expected value)** and the **contributions of each feature**, pushing the final output higher or lower.

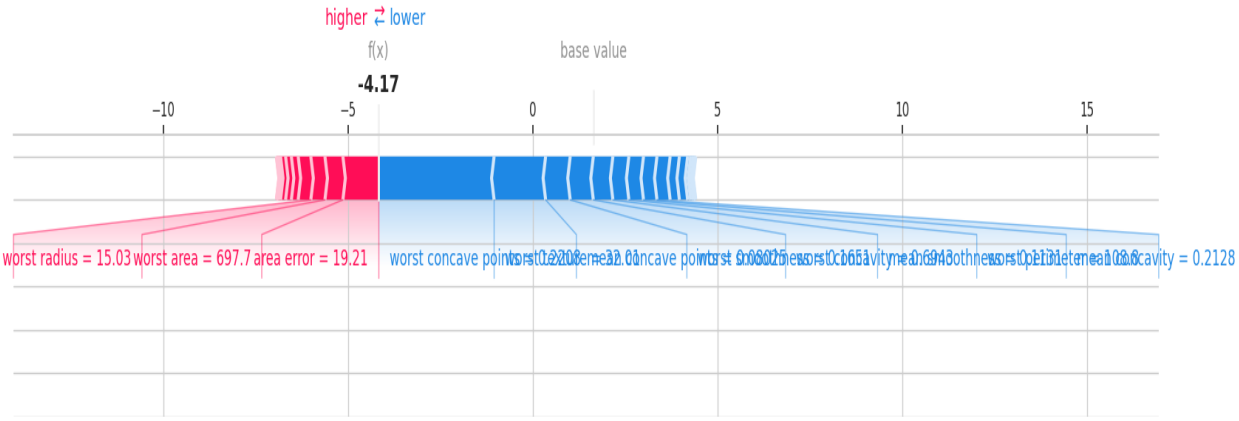


Figure 6 SHAP force plot explaining a single test prediction as a sum of feature contributions.

The force plot allows model builders to "zoom in" and answer: *why did the model think this specific tumor was benign or malignant?* This is invaluable in medical applications, where interpretability and transparency are as critical as accuracy.

Conclusion: Interpreting Black-Box Models with SHAP

In this tutorial, we explored one of the most powerful frameworks in interpretable machine learning — **SHAP (SHapley Additive exPlanations)**. We began by grounding ourselves in the motivation for explainability: the need to understand and trust model predictions, particularly in sensitive domains like healthcare and finance. We then introduced SHAP's theoretical foundation in Shapley values, offering a principled way to **attribute prediction outcomes to individual features**.

Using the Breast Cancer Wisconsin dataset, we demonstrated how to train a high-performing XGBoost classifier and explain its predictions using multiple SHAP visualizations — including **summary bar plots**, **beeswarm plots**, and **force plots**. Along the way, we practiced exploratory data analysis, performed scaling to stabilize feature influence, and discussed the trade-offs and limitations of SHAP. Importantly, we emphasized that **interpretability does not imply causality**, and that students must critically assess what these visual tools reveal about the model and the data.

By completing this tutorial, students now have a robust conceptual and practical understanding of:

- Why model interpretability matters
- How SHAP values offer local and global insights
- What responsible model explanation looks like in real workflows

Explainability is no longer a luxury in machine learning — it is a necessity. And with SHAP, we now have the tools to illuminate even the most complex prediction logic, turning **black boxes into glass boxes** that can be trusted, audited, and improved.

References:

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*, 281–305. http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, *55*(10), 78–87. https://doi.org/10.1145/2347736.2347755

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems (NeurIPS)*, *30*. https://papers.nips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., ... & Lee, S.-I. (2020). From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence*, *2*(1), 56–67. https://doi.org/10.1038/s42256-019-0138-9

Molnar, C. (2022). *Interpretable Machine Learning* (2nd ed.). Leanpub. https://christophm.github.io/interpretable-ml-book/

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830. http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

Tukey, J. W. (1977). *Exploratory Data Analysis*. Addison-Wesley.

Wolberg, W. H., Street, W. N., & Mangasarian, O. L. (1995). Breast Cancer Wisconsin (Diagnostic) Data Set. *UCI Machine Learning Repository*. https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29

⇒ Notebook (.ipynb):
https://github.com/Pavithiraseenivasagan/SHAP/blob/main/shap_explainability_breast_cancer%20(1).ipynb
⇒README.md:
https://github.com/Pavithiraseenivasagan/SHAP/blob/main/README_SHAP_Explainability.md
⇒ requirements.txt:
https://github.com/Pavithiraseenivasagan/SHAP/blob/main/requirements_SHAP_Explainability.txt
⇒ Report (.pdf):