Random Forest Classifier: A Hands-On Guide to Interpretable, Robust Machine Learning

Name: VISHAL KUMAR SENTHIL KUMAR
STUDENT ID: 23076841
GitHub: https://github.com/vishalkumar041298/random-forest-classifier-tutorial

1. Introduction

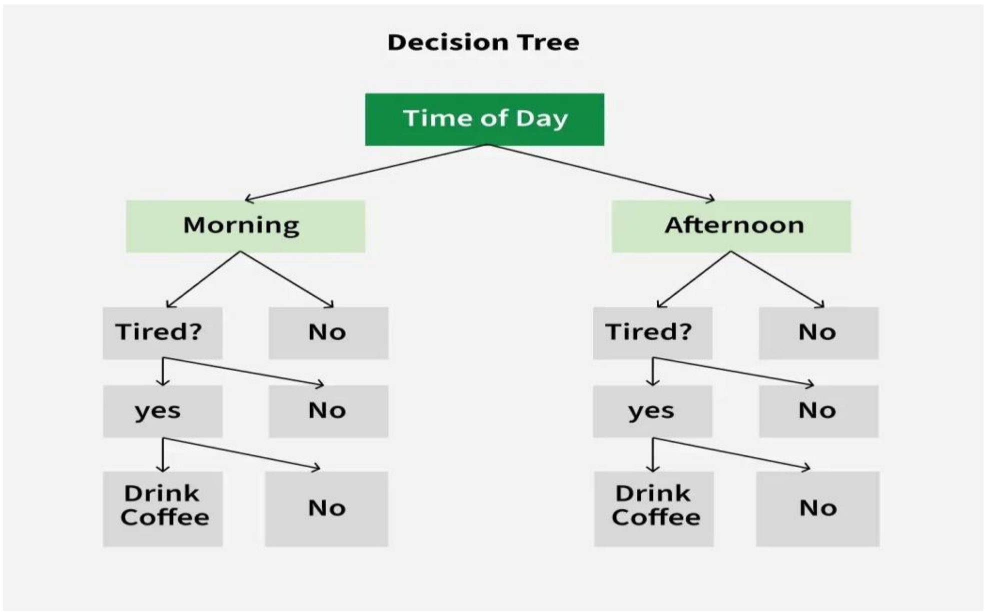
In today's data-driven world, choosing the right machine learning algorithm can be the difference between useful insights and misleading predictions. One of the most reliable, beginner-friendly, and yet powerful algorithms is the **Random Forest Classifier**. If you've ever trained a decision tree and found that it worked well on training data but struggled on new data, you're not alone this is a common challenge. Random Forest helps solve that problem.

It does so by combining the predictions of multiple decision trees, each trained slightly differently. This makes it **less likely to overfit**, and generally more **accurate and stable** than individual trees.
What makes Random Forest especially appealing is that it's a fantastic balance between performance and interpretability. You can still get a sense of what features matter most, unlike many black-box models like deep neural networks.

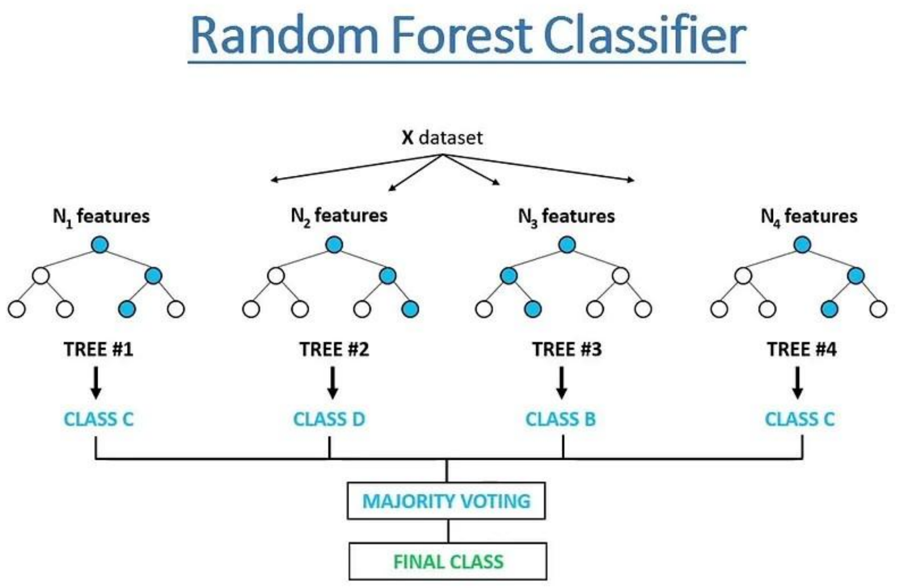
In this tutorial, we'll take a closer look at how Random Forest works, why it performs so well, and how to implement it step by step. We'll use a real dataset to make the process feel practical and approachable even if you're just getting started with machine learning. By the end, you should feel confident in using Random Forest for your own classification problems.

2. How Random Forest Works?

To understand why Random Forest is so powerful, we first need to revisit its building block: the **Decision Tree**. A decision tree is like a flowchart that asks a series of yes/no questions to classify data. It's simple, intuitive, and easy to visualize but unfortunately, it's also quite prone to overfitting.



This is where Random Forest shines. Instead of relying on one tree, it creates a "forest" of trees, each one slightly different, and combines their predictions. The result? Better generalization and a more reliable model. Typically, increasing the number of trees in a random forest enhances its robustness. Similarly, in a Random Forest Classifier, having more trees often leads to improved accuracy in predictions.



2.1. Bagging (Bootstrap Aggregation)

Random Forest uses a technique called bagging, short for bootstrap aggregation. Here's how it works:
It randomly samples the training data with replacement to create multiple datasets (called bootstrap samples).
A separate decision tree is trained on each of these datasets.
When it's time to make a prediction, the forest takes a vote (for classification) or an average (for regression) across all the trees.
The idea is simple but powerful: combining multiple weak models can lead to a strong one. The ensemble smooths out individual errors and reduces the risk of overfitting.

2.2. Random Feature Selection

In addition to sampling the data, Random Forest adds another layer of randomness: when building each tree, it selects only a random subset of features at each split. This prevents all the trees from looking too similar and ensures they learn different patterns from the data.
It's like assembling a diverse group of people to solve a problem, the more perspectives, the better the final decision.

2.3. Out-of-Bag (OOB) Error Estimation

An elegant bonus of bagging is the out-of-bag error estimate. Since each tree is trained on a bootstrap sample, about one-third of the data is left out (not used in training). This "left out" data can be used to test the model's performance, giving you a reliable accuracy estimate without needing a separate validation set.

Pseudocode for Random Forest

1. Like bagging, we begin by drawing multiple bootstrap samples from the original training dataset.
2. For each bootstrap sample, a decision tree is trained. These trees are allowed to grow to full depth (i.e., without pruning).

The key distinction in Random Forest lies in how the splits within each tree are determined. Unlike bagging, where all features are considered for splitting at each node, Random Forest introduces randomness by selecting a subset of features.

At each split in a tree:

- A random subset of m features is selected from the total p features ($m << p$).
- The best split is chosen only from this subset.

Why this randomness?

If a single predictor among the p is particularly strong, it might dominate the split decisions in every tree, leading to similar trees across the forest. This results in high correlation among trees, which undermines the goal of variance reduction through averaging.

By restricting the split decisions to a random subset of features:

- Diversity among trees is increased.
- Correlation is reduced.
- The ensemble becomes more robust and accurate.

Typically, $m \approx \sqrt{p}$ for classification problems.

If $m = p$, Random Forest behaves just like bagging.

Random Forest Tree Construction Steps

3. From the m selected features, identify the best split at a node.
4. Split the node into child (daughter) nodes using this optimal feature and split point.
5. Repeat steps 1–4 recursively until a stopping condition is met (e.g., a maximum number of nodes l or pure leaf nodes).
6. Repeat steps 1–5 to construct n different decision trees, each built on a different bootstrap sample with different feature subsets at splits.

Thus, a Random Forest is essentially an ensemble of n such trees, each grown with random feature selection at each split.

Random Forest Prediction Pseudocode

To make predictions with a trained Random Forest model:

1. Pass the test features through each of the n trained decision trees.
2. Each tree makes its own prediction (target output).
3. Collect all the predictions and determine the most frequently predicted target — this is known as **majority voting**.

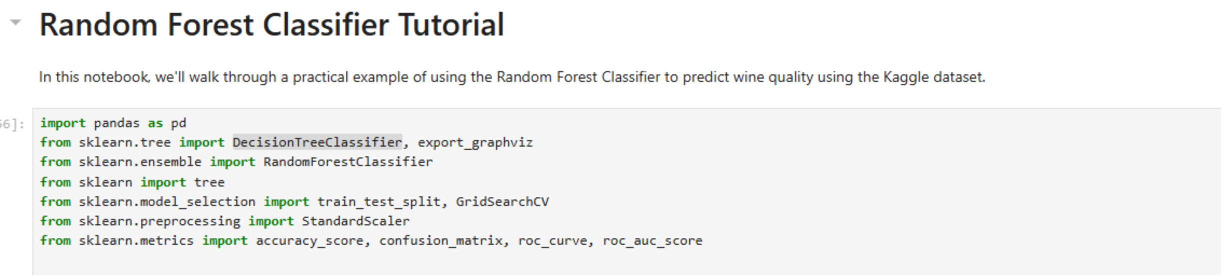
For example:

- If 100 trees are used and the possible predictions are x, y, and z:
    - Suppose 60 trees predict x, 25 predict y, and 15 predict z.
    - The final prediction by the Random Forest is x as it received the majority vote.

3. Practical Demonstration: Predicting Wine Quality

The dataset is taken from Kaggle: https://www.kaggle.com/datasets/rajyellow46/wine-quality

Step 1: Import Necessary libraries

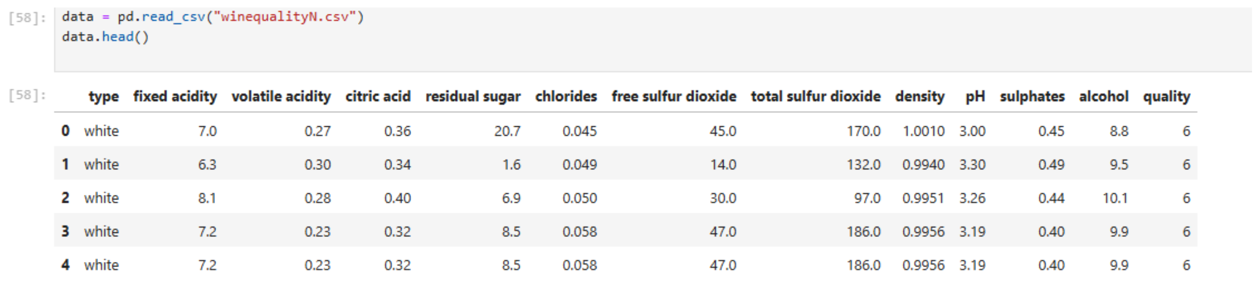


Random Forest Classifier Tutorial

In this notebook, we'll walk through a practical example of using the Random Forest Classifier to predict wine quality using the Kaggle dataset.

```
[56]: import pandas as pd
      from sklearn.tree import DecisionTreeClassifier, export_graphviz
      from sklearn.ensemble import RandomForestClassifier
      from sklearn import tree
      from sklearn.model_selection import train_test_split, GridSearchCV
      from sklearn.preprocessing import StandardScaler
      from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score
```

Step 2: Read the data

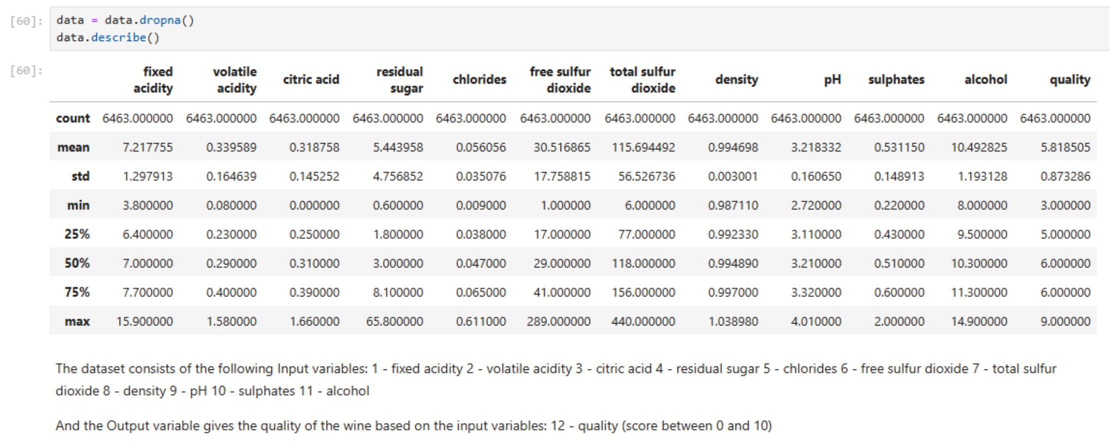


```
[58]: data = pd.read_csv("winequalityN.csv")
      data.head()
```

[58]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	white	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

Step 3: Remove null values and describe the data



```
[60]: data = data.dropna()
      data.describe()
```

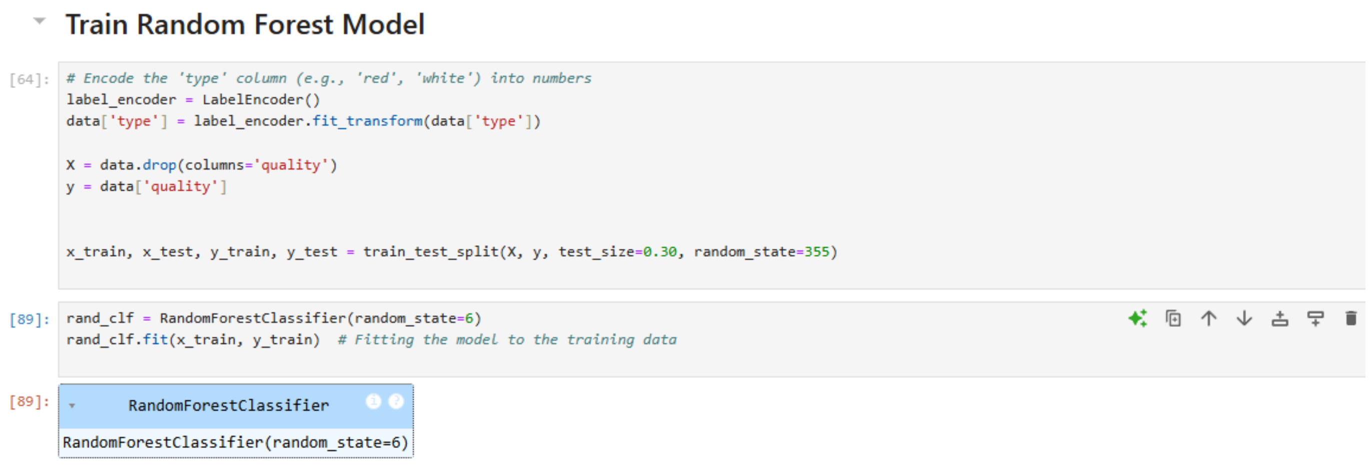
[60]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000	6463.000000
mean	7.217755	0.339589	0.318758	5.443958	0.056056	30.516865	115.694492	0.994698	3.218332	0.531150	10.492825	5.818505
std	1.297913	0.164639	0.145252	4.756852	0.035076	17.758815	56.526736	0.003001	0.160650	0.148913	1.193128	0.873286
min	3.800000	0.080000	0.000000	0.600000	0.009000	1.000000	6.000000	0.987110	2.720000	0.220000	8.000000	3.000000
25%	6.400000	0.230000	0.250000	1.800000	0.038000	17.000000	77.000000	0.992330	3.110000	0.430000	9.500000	5.000000
50%	7.000000	0.290000	0.310000	3.000000	0.047000	29.000000	118.000000	0.994890	3.210000	0.510000	10.300000	6.000000
75%	7.700000	0.400000	0.390000	8.100000	0.065000	41.000000	156.000000	0.997000	3.320000	0.600000	11.300000	6.000000
max	15.900000	1.580000	1.660000	65.800000	0.611000	289.000000	440.000000	1.038980	4.010000	2.000000	14.900000	9.000000

The dataset consists of the following Input variables: 1 - fixed acidity 2 - volatile acidity 3 - citric acid 4 - residual sugar 5 - chlorides 6 - free sulfur dioxide 7 - total sulfur dioxide 8 - density 9 - pH 10 - sulphates 11 - alcohol

And the Output variable gives the quality of the wine based on the input variables: 12 - quality (score between 0 and 10)

Step 4: Encode the data and train the random forest classifier model



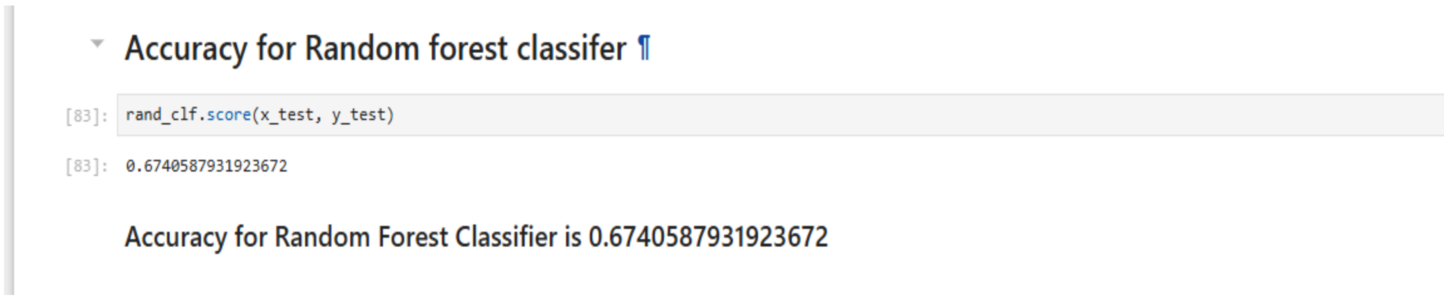
Train Random Forest Model

```
[64]: # Encode the 'type' column (e.g., 'red', 'white') into numbers
      label_encoder = LabelEncoder()
      data['type'] = label_encoder.fit_transform(data['type'])

      X = data.drop(columns='quality')
      y = data['quality']


      x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=355)
```

```
[89]: rand_clf = RandomForestClassifier(random_state=6)
      rand_clf.fit(x_train, y_train)  # Fitting the model to the training data
```

```
[89]: RandomForestClassifier
      RandomForestClassifier(random_state=6)
```

Step 5: Check the Accuracy score for Random Forest Classifier



Accuracy for Random forest classifer ¶

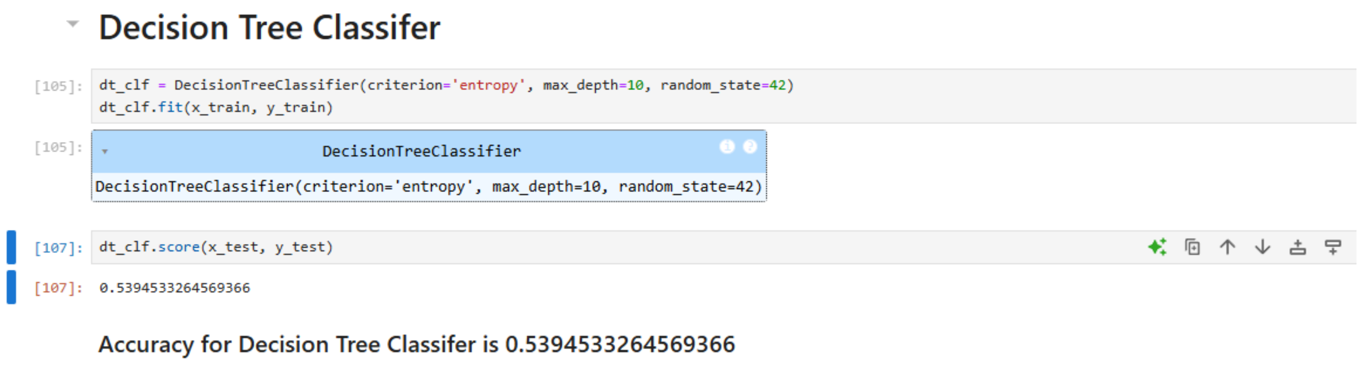
```
[83]: rand_clf.score(x_test, y_test)
```

```
[83]: 0.6740587931923672
```

Accuracy for Random Forest Classifier is 0.6740587931923672

The Accuracy for Random Forest Classifier is 0.6740587931923672

Step 6: Do the same for Decision Tree Classifier to compare.



Decision Tree Classifer

```
[105]: dt_clf = DecisionTreeClassifier(criterion='entropy', max_depth=10, random_state=42)
       dt_clf.fit(x_train, y_train)
```

```
[105]: DecisionTreeClassifier
       DecisionTreeClassifier(criterion='entropy', max_depth=10, random_state=42)
```

```
[107]: dt_clf.score(x_test, y_test)
```

```
[107]: 0.5394533264569366
```

Accuracy for Decision Tree Classifer is 0.5394533264569366

Result

The Random Forest Classifier achieved **a higher accuracy score** compared to the Decision Tree Classifier, indicating better overall performance on the wine quality prediction task.

Key Hyperparameters of the Random Forest Classifier:

1. **max_depth**: Defines the maximum depth of each decision tree in the forest, representing the longest path from the root node to any leaf node.
2. **min_samples_split**: Specifies the minimum number of data points required in a node before it's allowed to split. The default value is 2.
3. **max_leaf_nodes**: Limits the number of leaf nodes in each tree by controlling how many times nodes can split, thus preventing excessive tree growth.
4. **min_samples_leaf**: Sets the minimum number of samples that must be present in a leaf node after a split. By default, this is set to 1.
5. **n_estimators**: Indicates the total number of individual decision trees that make up the random forest.
6. **max_samples**: Controls the proportion of the original training data used to train each individual tree.
7. **max_features**: Refers to the maximum number of features considered when looking for the best split at each node.
8. **bootstrap**: Determines whether sampling of training data is done with replacement (True by default) or not.
9. **criterion**: The metric used to evaluate the quality of a split, commonly either "gini" for Gini impurity or "entropy" for information gain.

4. Why Use Random Forest?

Advantages

- **High Accuracy:** Often delivers excellent results without heavy tuning.
- **Reduced Overfitting**: The ensemble approach smooths out noise and variance.
- **Works Out-of-the-Box:** Great for prototyping and first-pass models.
- **Handles Nonlinear Relationships:** No need to transform data or assume distributions.
- **Feature Importance:** You can see which features matter most — helpful in model interpretation.
- **Robust to Missing Data:** Can handle datasets where not all features are present in every row.

Disadvantages

- **More Computationally Expensive:** Training hundreds of trees can take time, especially on large datasets.
- **Model Size:** The saved model is much larger than a single tree.
- **Can be Overfit with Noisy Labels:** Especially if too many trees are used without constraint**.**

5. Interpretability and Real-World Use

One reason people love Random Forest is because it’s not a black box. Yes, it’s more complex than a single tree but it still gives you feature importance, and tools like SHAP and LIME can make individual predictions more explainable.
This is vital in fields like healthcare, law, or finance, where understanding why a prediction was made is just as important as the prediction itself.
For instance, if a model predicts someone is at high risk of heart disease, doctors want to know if that’s because of cholesterol, age, or other factors. Random Forest gives us that level of insight without sacrificing accuracy.

6. Conclusion

Random Forest is one of those rare machine learning models that’s both easy to use and highly effective. It improves on decision trees by adding randomness and ensemble, which leads to better generalization and less overfitting.
Through our heart disease prediction example, we saw how Random Forest:

- Outperformed a single tree on accuracy.
- Identified important features driving the prediction.
- Produced strong evaluation metrics.

7. References

- **Scikit-learn Documentation**: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- **Towards Data Science**: https://medium.com/analytics-vidhya/random-forest-classifier-and-its-hyperparameters-8467bec755f6
- **Kaggle Wine Dataset**: https://www.kaggle.com/datasets/rajyellow46/wine-quality
- **SHAP for Tree-Based Models:** https://shap.readthedocs.io