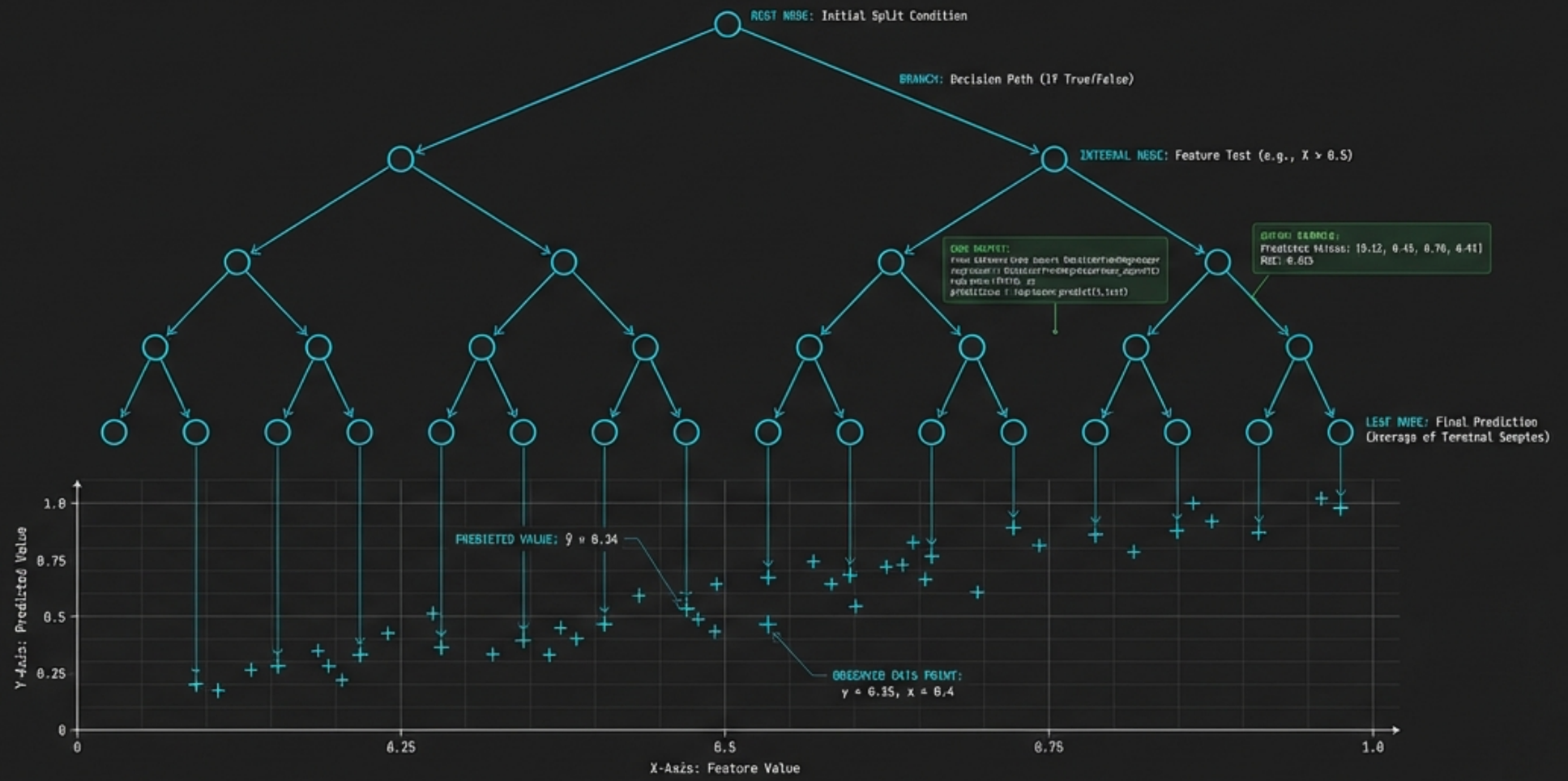
DECISION TREE REGRESSION

A Blueprint for Predicting Continuous Values

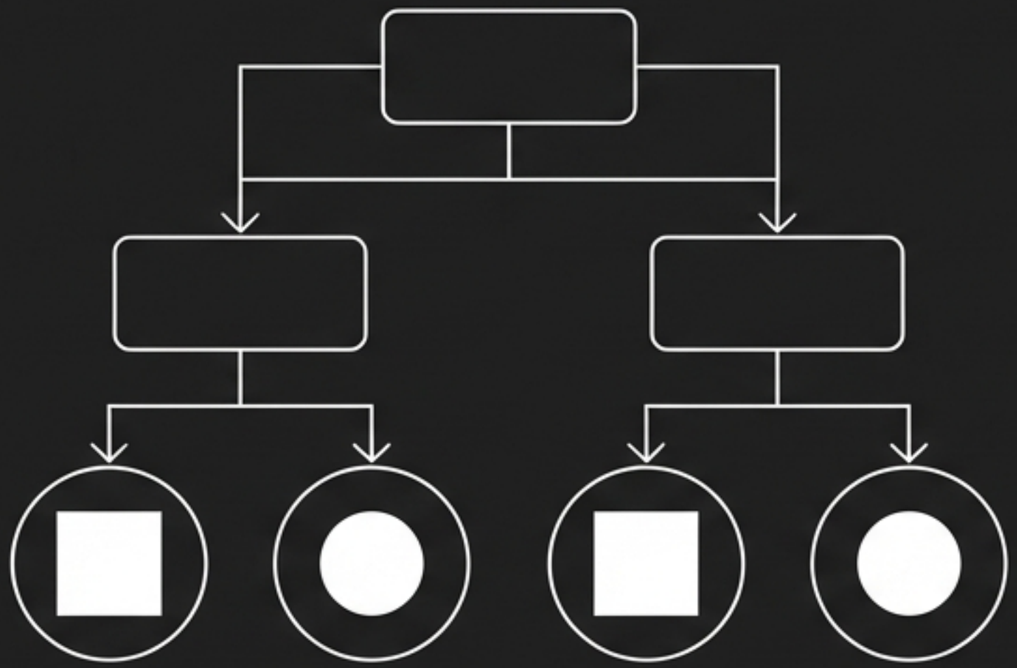


An architectural guide to the logic, mathematics, and code implementation of regression trees.

Predicting Quantities, Not Categories.

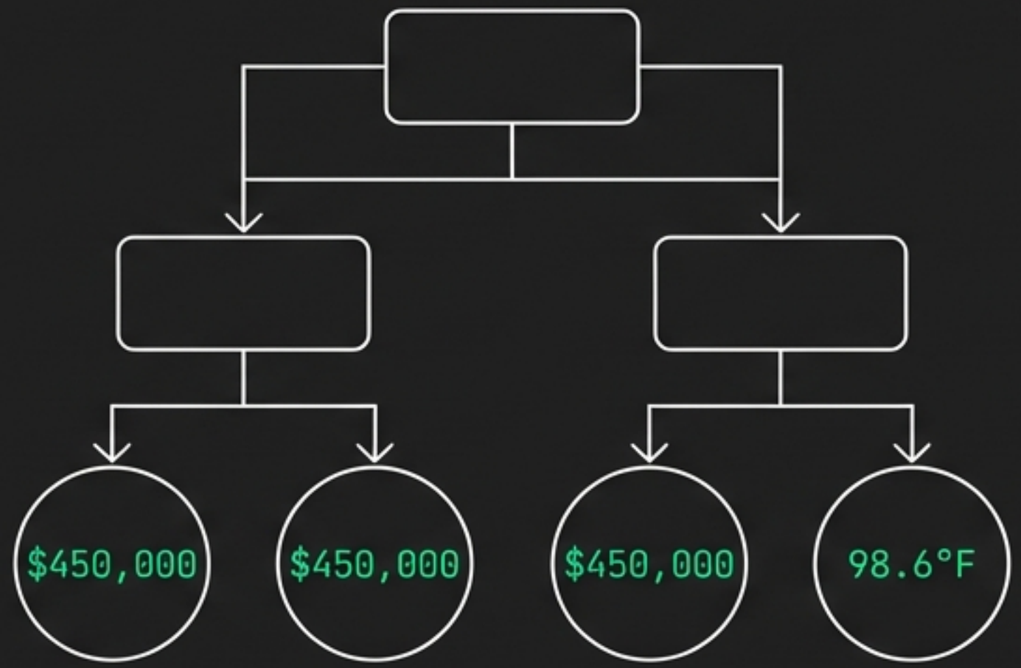
Regression trees output numerical data, distinct from the categorical output of classification trees.

CLASSIFICATION



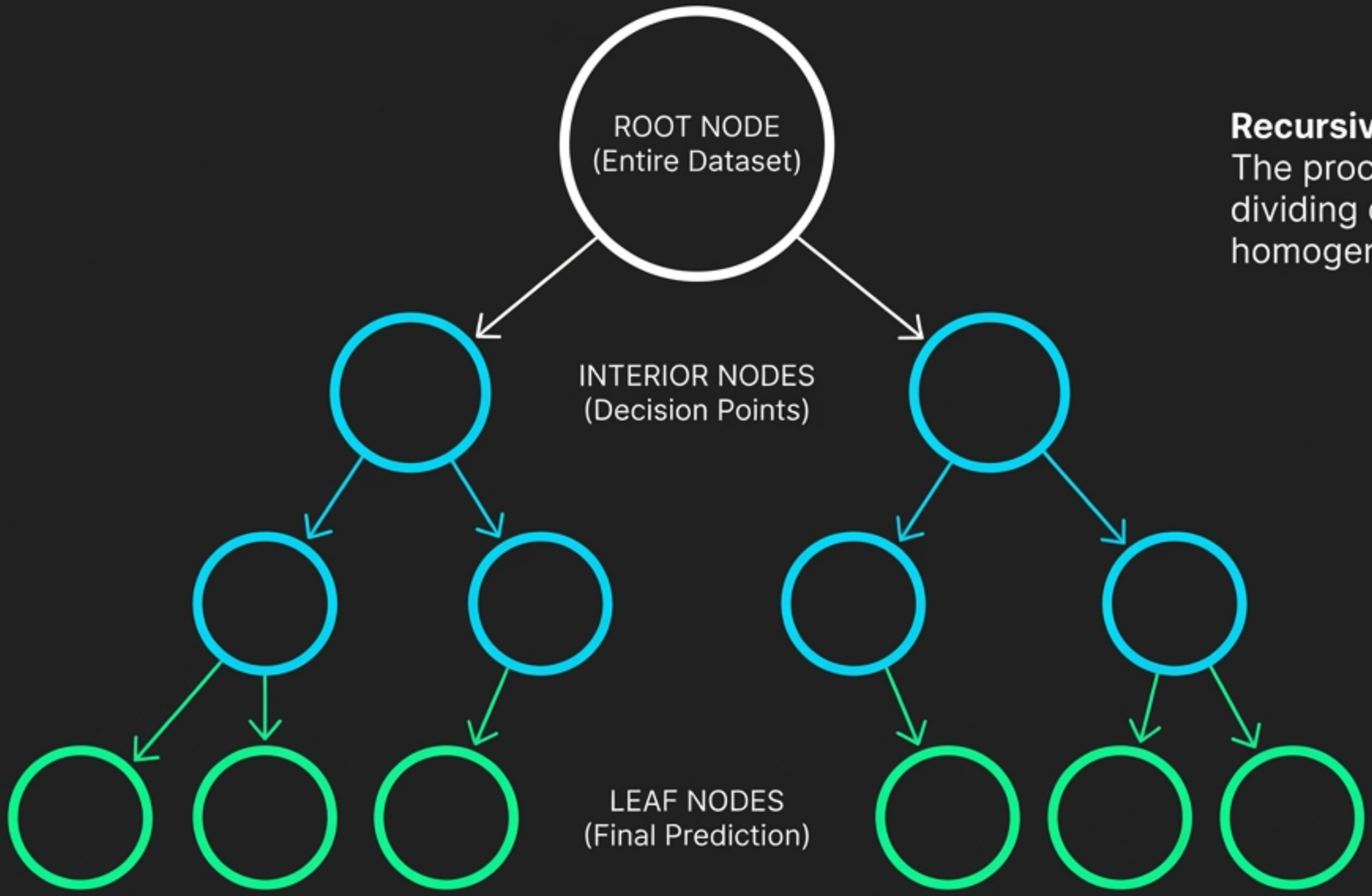
Predicts Categories.

REGRESSION



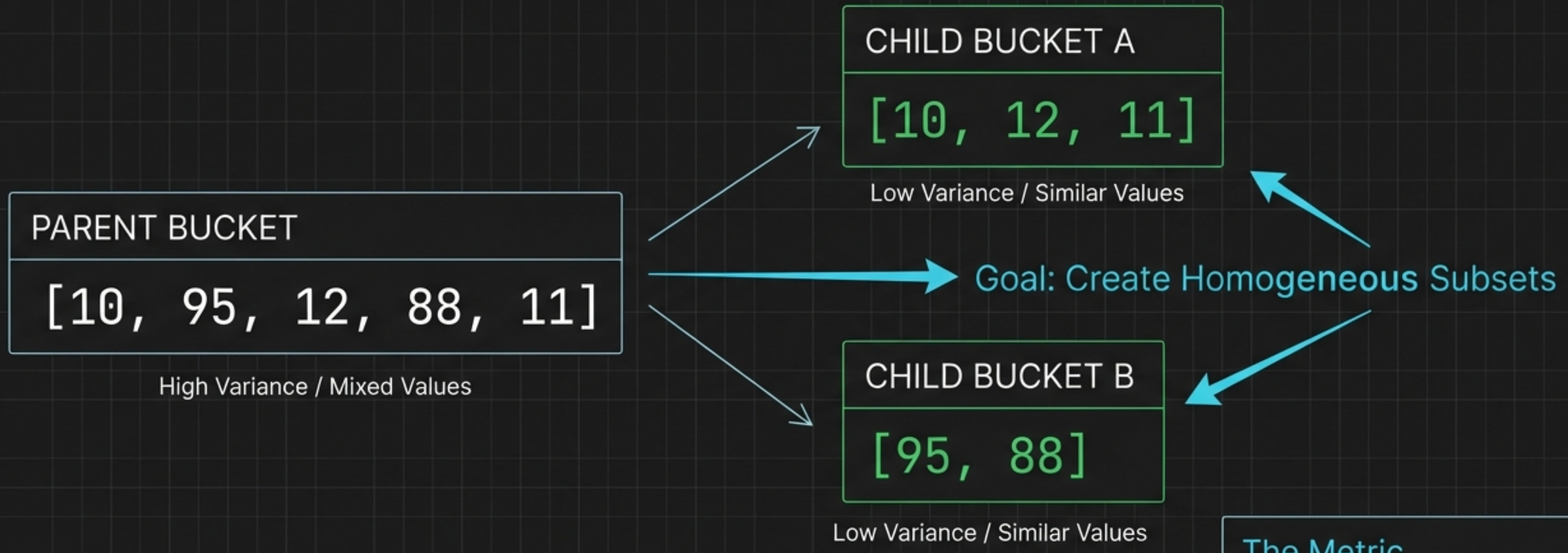
Predicts Continuous Quantities.

The Anatomy of the Algorithm.



Recursive Splitting:
The process of repeatedly dividing data into smaller, homogeneous subsets.

The Mechanics of the Split



The Metric

Unlike Classification (which uses Gini/Entropy), Regression uses Variance Reduction to find the best split.

Calculating Variance Reduction

$$\text{Variance Reduction} = Var(root) - \sum \left(\frac{n_i}{N} * Var(child_i) \right)$$

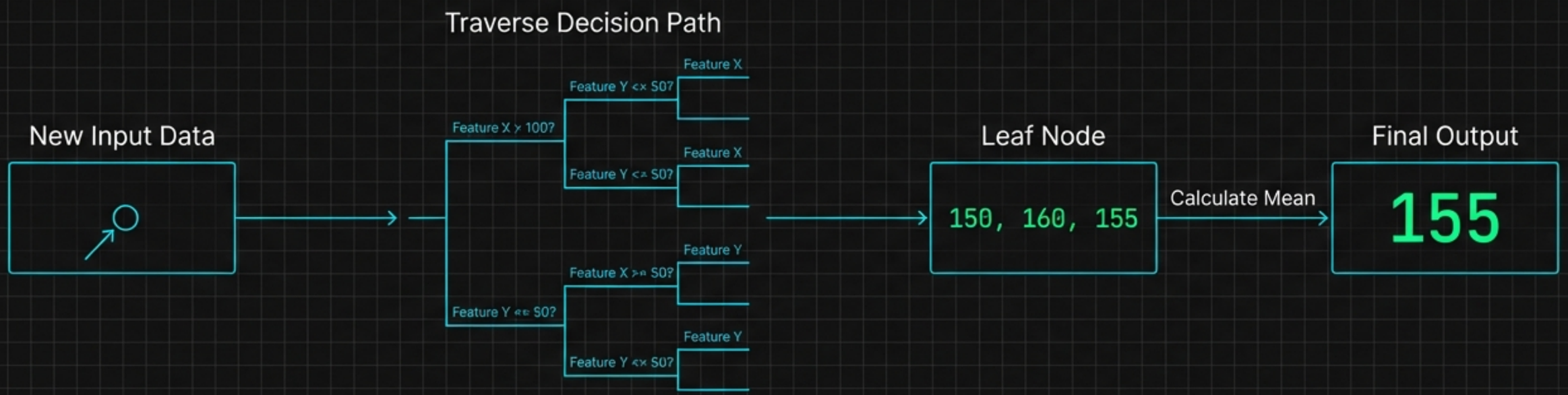
Variance of the parent node.
Variant of the parent node where JetBrains Mono.

Weight of the child node (Size relative to total).
The variance of child node enters JetBrains Mono.

Variance of the child node.
Variance of the child node in JetBrains Mono.

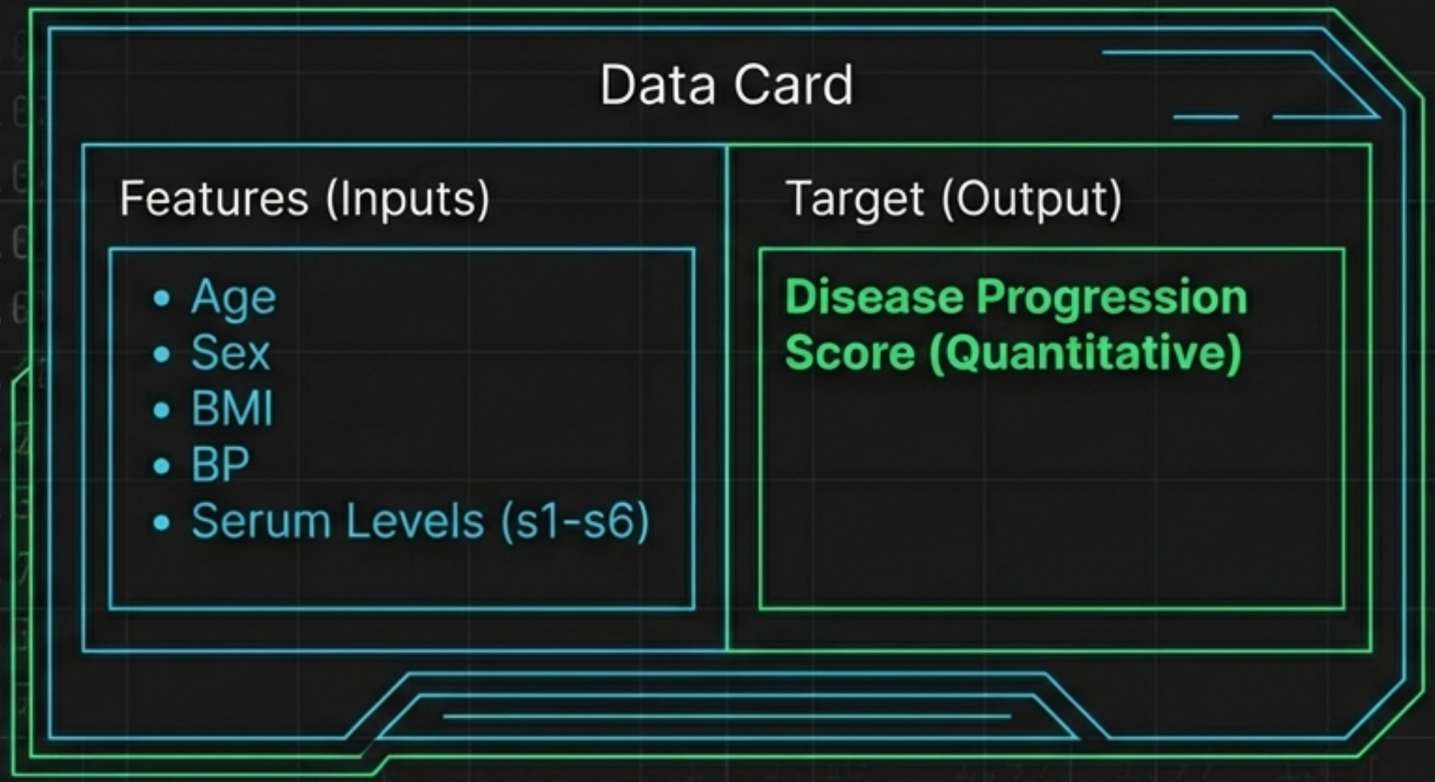
The algorithm calculates this for every possible split and chooses the highest value.

Deriving the Prediction



The final prediction is the average (mean) of all training samples in the destination leaf.

The Scenario: Diabetes Progression.



Data Note: JetBrains Mono Source: Scikit-Learn Diabetes Dataset (n=442). Features are mean-centered and scaled.

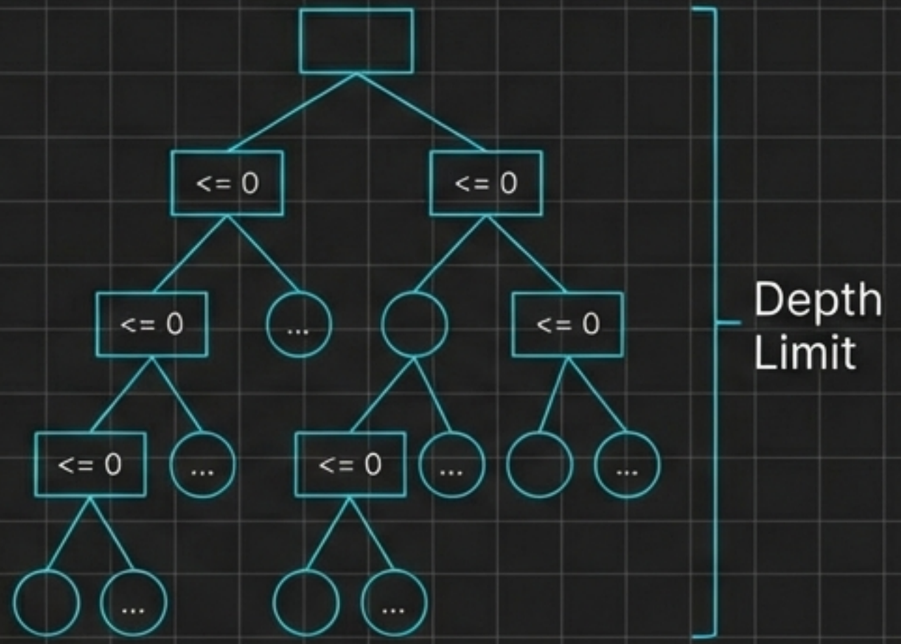
Phase 1: Data Preparation

```python
import pandas as pd
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split

data = load_diabetes()
X, y = pd.DataFrame(data['data'], columns=...), data['target']

# Split: 75% Train, 25% Test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

Crucial Step: Segregating data to evaluate performance on unseen inputs.

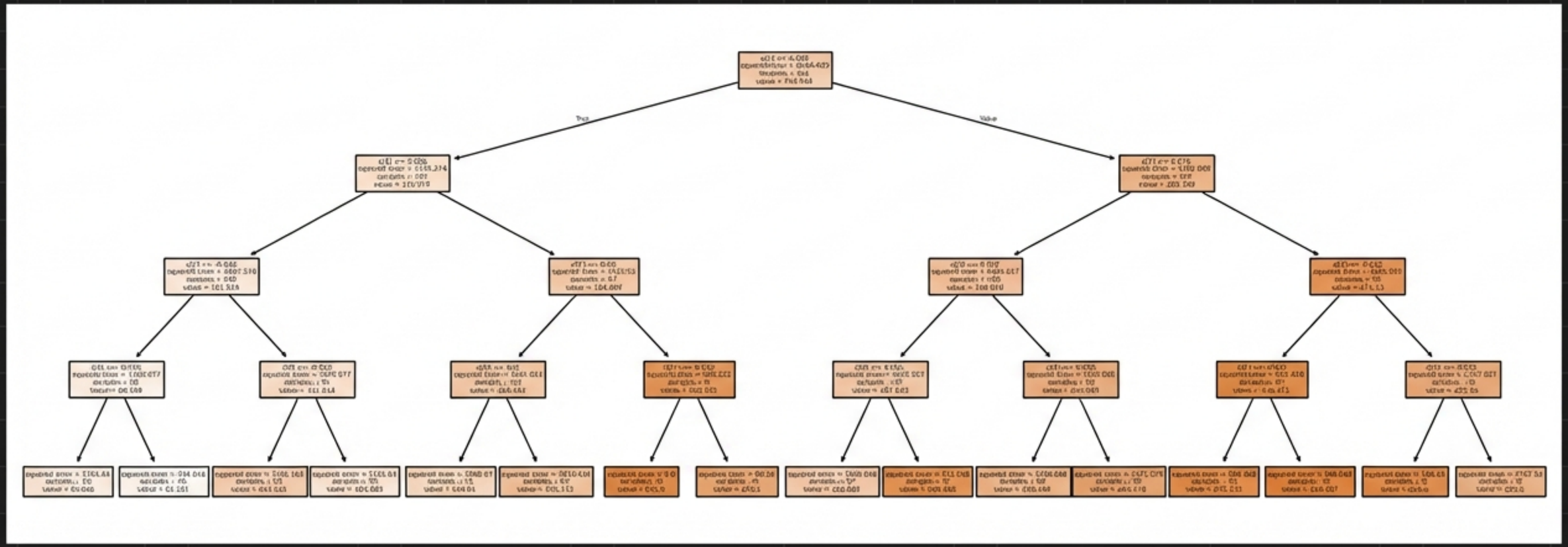
Phase 2: Training the Model.

```python
from sklearn.tree import DecisionTreeRegressor

# Initialize with depth limit
decision_tree_reg = DecisionTreeRegressor(max_depth=4)

# The math happens here
decision_tree_reg.fit(X_train, y_train)
decision_tree_reg.fit(X_train, y_train)
```



Prevents Overfitting: Limits complexity so the model learns patterns, not noise.

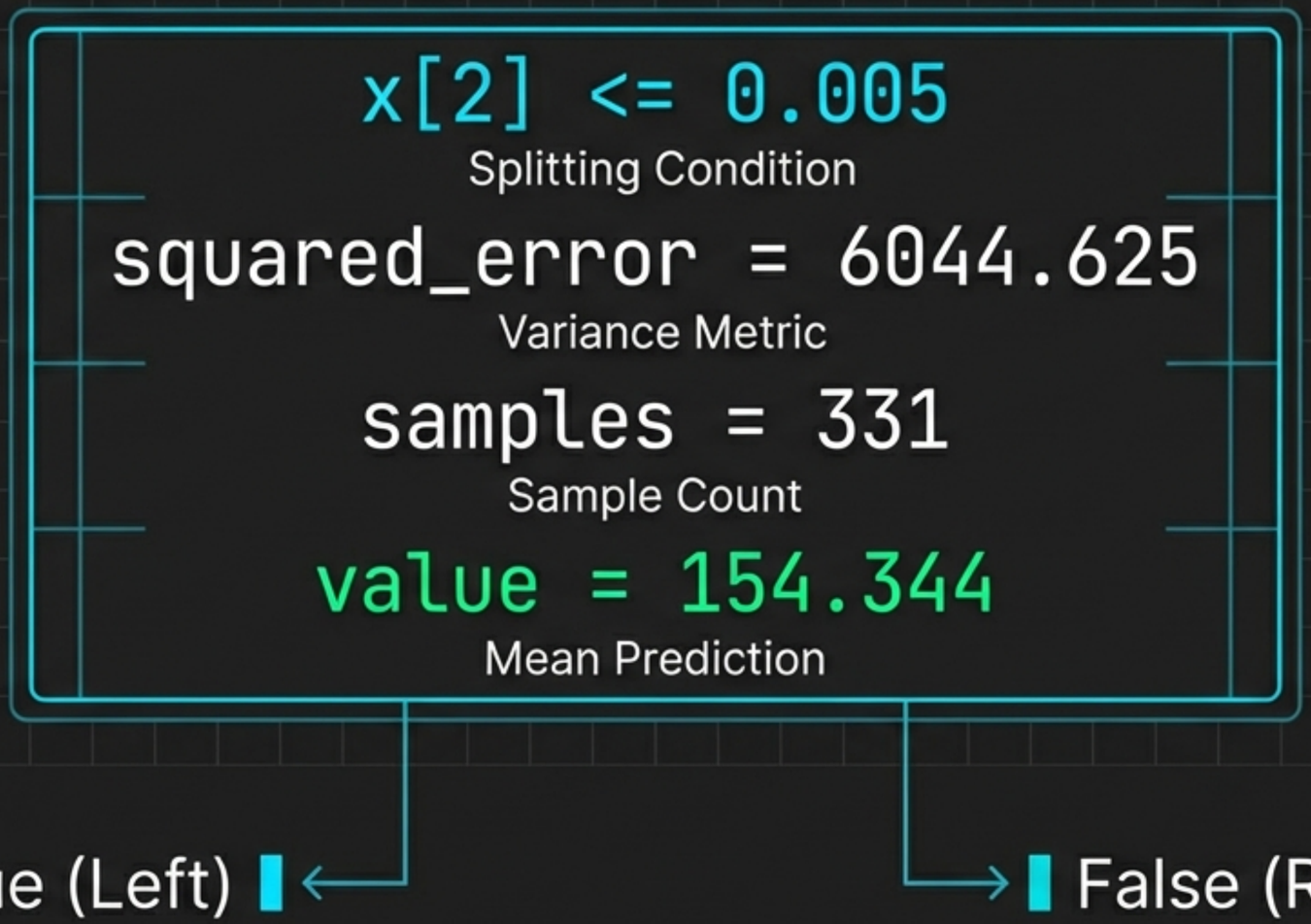
Visualizing the Decision Logic.



```
tree.plot_tree(decision_tree_reg, filled=True)
```

The resulting logic structure. Darker orange nodes indicate higher predicted values.

Decoding a Single Node.



Phase 3: Making Predictions.

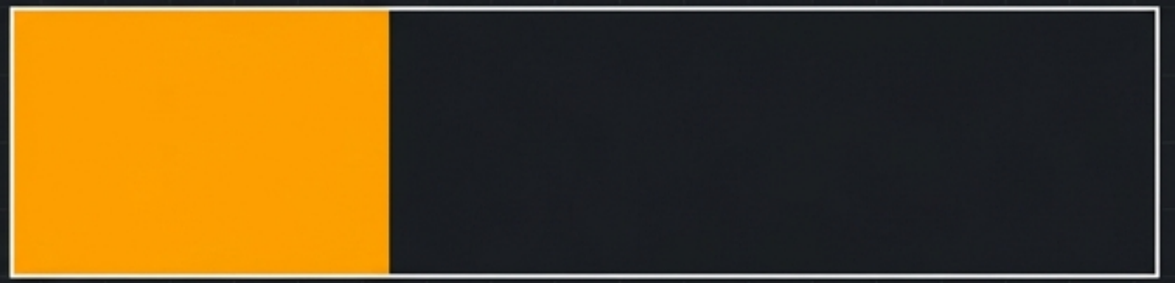
```
y_pred = decision_tree_reg.predict(X_test)
```

```
[ 175.33, 199.29, 175.33, 244.77, 104.96 ... ]
```

Predicted disease progression for Patient X.

Phase 4: Evaluation.

```
from sklearn.metrics import r2_score

score = r2_score(y_pred=y_pred, y_true=y_test)
```



R2 Score: 0.3401

The model explains 34% of the variance. This baseline score suggests the need for hyperparameter tuning or more complex ensemble methods.

Key Takeaways.

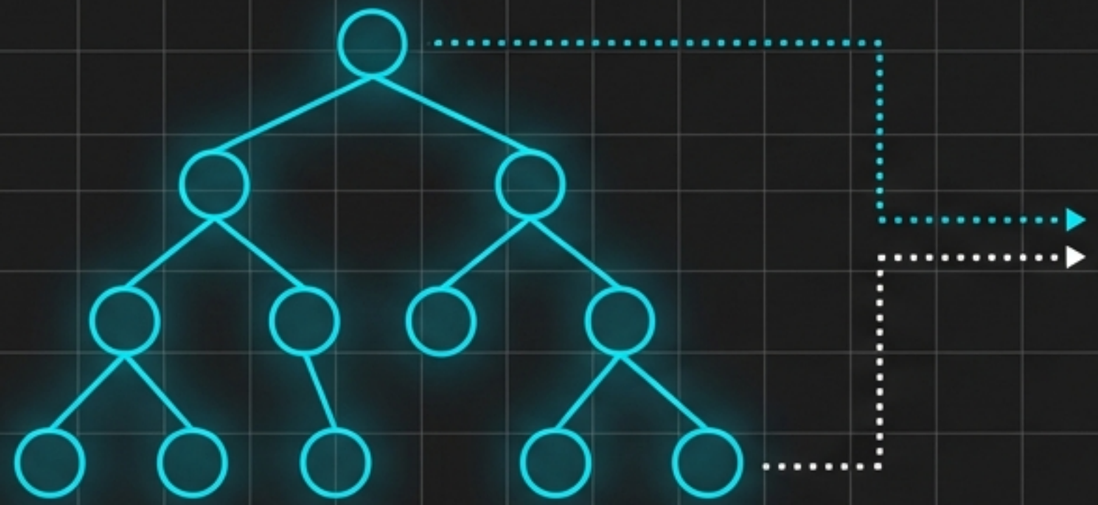
Regression predicts continuous numbers, not categories.

Splits are determined by Variance Reduction (Minimizing error).

Leaf Prediction = Mean value of the samples in that node.

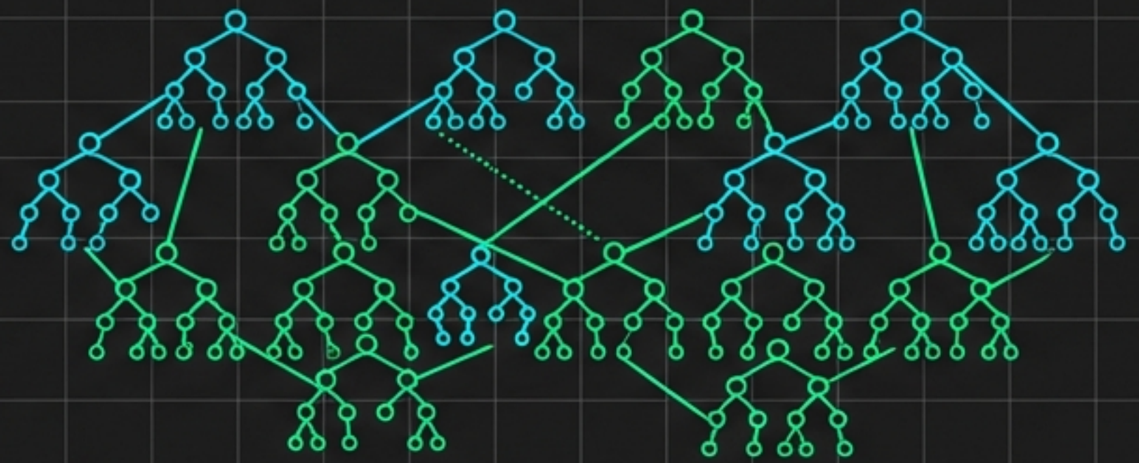
Max_Depth controls model complexity to prevent overfitting.

Beyond the Single Tree



Interpretable, but prone to high variance (0.34 R2 Score)

Ensemble Methods



Next Steps:

1. Hyperparameter Tuning (min_samples_split).
2. Random Forests & Gradient Boosting.

“The tree is just the beginning of the forest.”