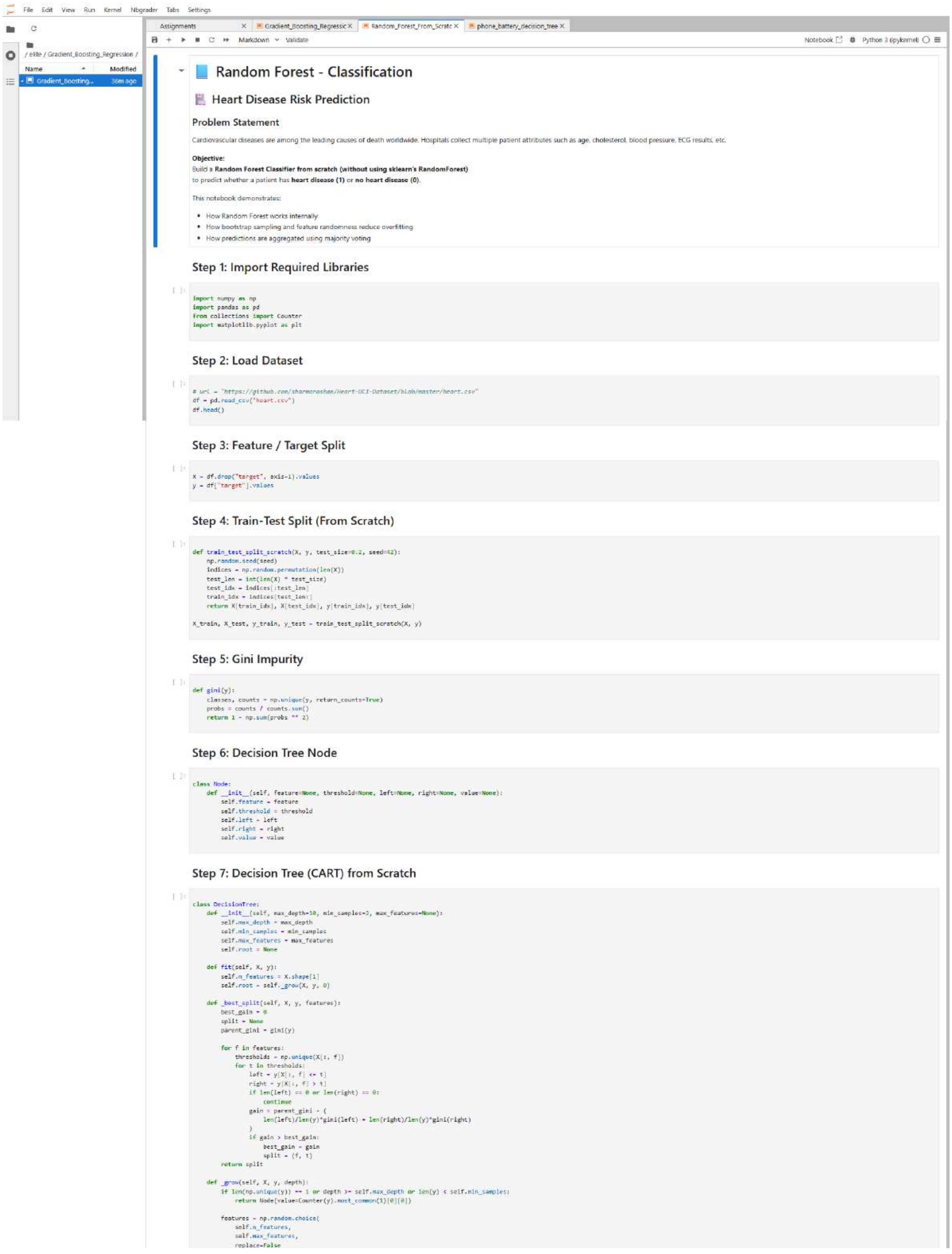

Random Forest - Classification

Heart Disease Risk Prediction

Problem Statement

Cardiovascular diseases are among the leading causes of death worldwide. Hospitals collect multiple patient attributes such as age, cholesterol, blood pressure, ECG results, etc.

Objective:
Build a **Random Forest Classifier from scratch (without using sklearn's RandomForest)**
to predict whether a patient has **heart disease (1)** or **no heart disease (0)**.

This notebook demonstrates:

- How Random Forest works internally
- How bootstrap sampling and feature randomness reduce overfitting
- How predictions are aggregated using majority voting

Step 1: Import Required Libraries

```python
import numpy as np
import pandas as pd
from collections import Counter
import matplotlib.pyplot as plt
```

Step 2: Load Dataset

```python
# url = "https://github.com/sharmaroshan/Heart-UCI-Dataset/blob/master/heart.csv"
df = pd.read_csv("heart.csv")
df.head()
```

Step 3: Feature / Target Split

```python
X = df.drop("target", axis=1).values
y = df["target"].values
```

Step 4: Train-Test Split (From Scratch)

```python
def train_test_split_scratch(X, y, test_size=0.2, seed=42):
    np.random.seed(seed)
    indices = np.random.permutation(len(X))
    test_len = int(len(X) * test_size)
    test_idx = indices[:test_len]
    train_idx = indices[test_len:]
    return X[train_idx], X[test_idx], y[train_idx], y[test_idx]

X_train, X_test, y_train, y_test = train_test_split_scratch(X, y)
```

Step 5: Gini Impurity

```python
def gini(y):
    classes, counts = np.unique(y, return_counts=True)
    probs = counts / counts.sum()
    return 1 - np.sum(probs ** 2)
```

Step 6: Decision Tree Node

```python
class Node:
    def __init__(self, feature=None, threshold=None, left=None, right=None, value=None):
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value
```

Step 7: Decision Tree (CART) from Scratch

```python
class DecisionTree:
    def __init__(self, max_depth=10, min_samples=2, max_features=None):
        self.max_depth = max_depth
        self.min_samples = min_samples
        self.max_features = max_features
        self.root = None

    def fit(self, X, y):
        self.n_features = X.shape[1]
        self.root = self._grow(X, y, 0)

    def _best_split(self, X, y, features):
        best_gain = 0
        split = None
        parent_gini = gini(y)

        for f in features:
            thresholds = np.unique(X[:, f])
            for t in thresholds:
                left = y[X[:, f] <= t]
                right = y[X[:, f] > t]
                if len(left) == 0 or len(right) == 0:
                    continue
                gain = parent_gini - (
                    len(left)/len(y)*gini(left) + len(right)/len(y)*gini(right)
                )
                if gain > best_gain:
                    best_gain = gain
                    split = (f, t)
        return split

    def _grow(self, X, y, depth):
        if len(np.unique(y)) == 1 or depth >= self.max_depth or len(y) < self.min_samples:
            return Node(value=Counter(y).most_common(1)[0][0])

        features = np.random.choice(
            self.n_features,
            self.max_features,
            replace=False
        )

        split = self._best_split(X, y, features)
        if split is None:
            return Node(value=Counter(y).most_common(1)[0][0])

        f, t = split
        left_idx = X[:, f] <= t
        right_idx = X[:, f] > t

        return Node(
            feature=f,
            threshold=t,
            left=self._grow(X[left_idx], y[left_idx], depth+1),
            right=self._grow(X[right_idx], y[right_idx], depth+1)
        )

    def predict_row(self, x, node):
        if node.value is not None:
            return node.value
        if x[node.feature] <= node.threshold:
            return self.predict_row(x, node.left)
        return self.predict_row(x, node.right)

    def predict(self, X):
        return np.array([self.predict_row(x, self.root) for x in X])
```

Step 8: Random Forest from Scratch

[ ]:
```python
class RandomForest:
    def __init__(self, n_trees=20, max_depth=10):
        self.n_trees = n_trees
        self.max_depth = max_depth
        self.trees = []

    def fit(self, X, y):
        self.trees = []
        for _ in range(self.n_trees):
            # YOUR CODE HERE
            raise NotImplementedError()

    def predict(self, X):
        # Collect predictions from all trees
        predictions = np.array([tree.predict(X) for tree in self.trees])

        # YOUR CODE HERE
        raise NotImplementedError()
```

Step 9: Train Random Forest

[ ]:
```python

rf = RandomForest(n_trees=25, max_depth=12)
rf.fit(X_train, y_train)
```

Step 10: Evaluation

[ ]:
```python

y_pred = rf.predict(X_test)
accuracy = np.mean(y_pred == y_test)
print("Accuracy:", accuracy)
```

✅ Conclusion

In this notebook we implemented **Random Forest from scratch** using:

- Bootstrap sampling
- Random feature selection at each split
- Majority voting for prediction

This approach significantly reduces overfitting compared to a single Decision Tree, making Random Forest a powerful ensemble learning technique.

[ ]:

[ ]:

[ ]:

Final Remarks

These asset-based test cases validate:

- Mathematical correctness (Gini)
- Structural correctness (tree and forest)
- Algorithmic behavior (bootstrap randomness)
- Predictive sanity (minimum accuracy threshold)

Simple 3 Python 3 (ipykernel) | Idle Mode: Command Ln 1, Col 1 Random_Forest_From_Scratch_Heart_Disease.ipynb 1