Name: Ronit Sahoo
Class: D15C
Roll No: 41
Batch: B

MLDL EXPERIMENT NO: 3

Aim:

- Apply Decision Tree and Random Forest for classification tasks.

Dataset Description:

The Mobile Specifications Dataset is a real-world multivariate dataset compiled from commercially available smartphone information across different models, configurations, and regional markets. It is widely used for intermediate-level machine learning tasks such as mobile price prediction and device category classification. Unlike perfectly balanced or synthetic academic datasets, this dataset reflects real market behavior, featuring heterogeneous feature types, multiple variants of the same device, and noticeable price variation across regions, which makes it a strong benchmark for evaluating the pruning behavior of Decision Trees and the variance-reduction and generalization capabilities of Random Forests. The dataset aims to model smartphone pricing or classification outcomes based on a diverse set of physical attributes, hardware specifications, and launch details. The dataset consists of multiple columns, some of which are:

Column Name	Data Type	Description
Company	Categorical (String)	Brand or manufacturer of the smartphone
Model Name	Categorical (String)	Commercial name of the mobile device
Mobile Weight	Numerical (Integer)	Weight of the device measured in grams
RAM	Categorical (String)	Installed RAM capacity (e.g., 6GB, 8GB)
Front Camera	Categorical (String)	Front camera resolution and video capability
Back Camera	Categorical (String)	Rear camera configuration and resolution
Processor	Categorical (String)	Chipset used in the device
Battery Capacity	Numerical (Integer)	Battery size in milliampere-hours (mAh)
Screen Size	Numerical (Float)	Display size measured in inches
Launched Price	Numerical (Integer)	Launch price of the device in different currencies
Launched Year	Numerical (Integer)	Year in which the smartphone was officially released

Dataset Source: https://www.kaggle.com/datasets/abdulmalik1518/mobiles-dataset-2025

DECISION TREE

Theory:

- A Decision Tree is a non-parametric supervised machine learning algorithm used for both classification and regression tasks. It has a tree-like structure where internal nodes represent feature-based splits, branches represent outcomes of those splits, and leaf nodes generate final predictions such as a smartphone price range or device category. The main objective of a Decision Tree is to learn simple and interpretable decision rules from input features. Unlike linear models that rely on a single global equation, Decision Trees use recursive partitioning to divide the dataset into smaller subsets, enabling them to capture complex and non-linear relationships among smartphone attributes such as processor type, battery capacity, and screen size.

Mathematical Formulation:

- For classification problems, Entropy measures the level of uncertainty in a dataset. For a dataset with *c* classes, entropy is defined as:

$$Entropy(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

where pi is the proportion of samples belonging to class i.

- Information Gain (IG) evaluates the reduction in entropy after splitting the dataset on a feature:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

- Another widely used splitting criterion is the Gini Index, which measures node impurity. Lower Gini values indicate purer nodes:

$$Gini(S) = 1 - \sum_{i=1}^{c} p_i^2$$

In practice, CART-based Decision Trees select the feature that minimizes the weighted Gini Index, while ID3 relies on Entropy and Information Gain.

Limitations:

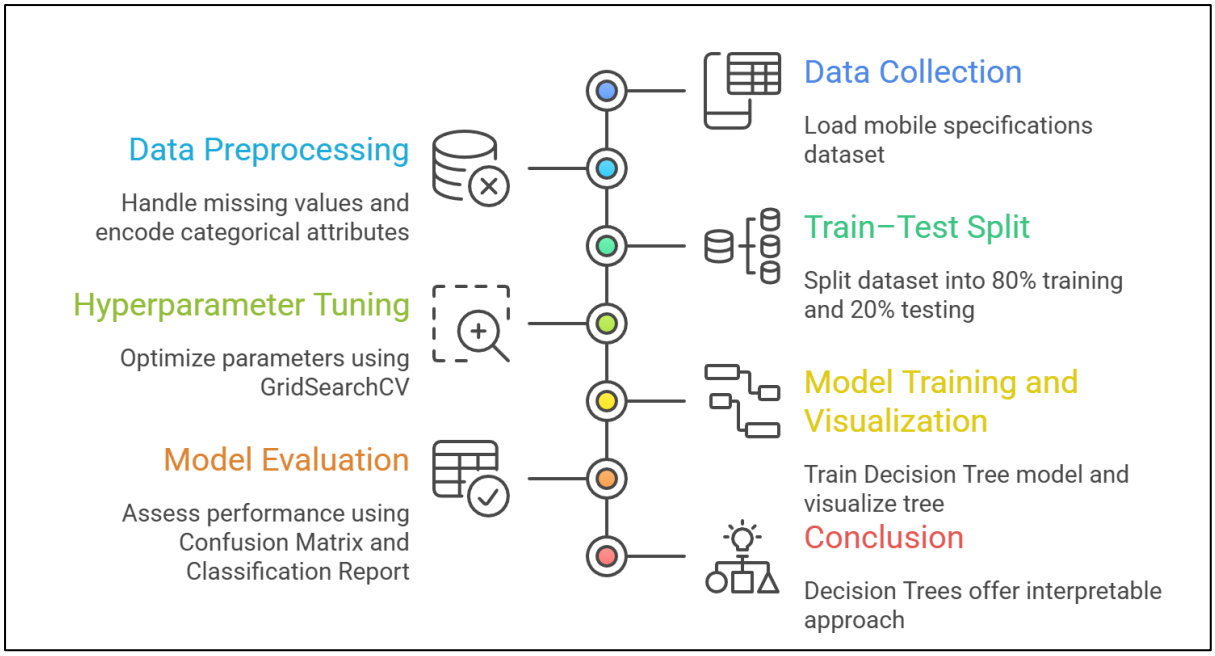
1. **Overfitting (High Variance):** Decision Trees are highly susceptible to overfitting, particularly when they are allowed to grow without depth restrictions. As the tree keeps splitting to achieve higher purity, it often begins to capture noise and minor variations in the training data, such as small pricing differences between similar mobile variants. This results in very specific decision rules that perform well on training data but fail to generalize effectively to unseen smartphone models.

2. **Instability (Sensitivity to Data Variations):** A major drawback of a single Decision Tree is its sensitivity to small changes in the dataset. Because the algorithm makes greedy, locally optimal decisions at each node, even a minor modification in the data—such as adding or removing a few mobile entries—can significantly alter the tree structure. For instance, the root split may change from “Processor” to “RAM,” leading to an entirely different set of decision paths, which reduces the reliability of standalone trees.

3. **Inefficiency in Modeling Linear Trends:** Although Decision Trees are powerful for capturing non-linear patterns, they are not well suited for modeling smooth linear relationships, such as a steady increase in price with increasing storage or RAM. Since splits are axis-aligned, the model must create multiple small partitions to approximate a linear trend, making the tree deeper and less efficient.

Workflow:

1. **Data Collection:** The mobile specifications dataset is loaded from a CSV file into a Pandas DataFrame and contains smartphone features used for price or category prediction.
2. **Data Preprocessing:** Missing values are handled, and categorical attributes such as brand, processor, and RAM are converted into numerical form using Label Encoding.
3. **Train–Test Split:** The dataset is split into 80% training data and 20% testing data to evaluate performance on unseen devices.
4. **Hyperparameter Tuning:** GridSearchCV is applied to optimize parameters like max_depth, min_samples_split, and the splitting criterion to reduce overfitting.
5. **Model Training and Visualization:** The best Decision Tree model is trained, and a simplified tree visualization highlights key feature-based decisions.
6. **Model Evaluation:** Performance is assessed using a Confusion Matrix, Classification Report, and feature importance analysis.
7. **Conclusion:** Decision Trees offer an interpretable approach to smartphone prediction tasks, with improved performance achievable using ensemble methods such as Random Forest.



Performance Analysis:

The performance of the optimized Decision Tree model on the mobile price prediction dataset is evaluated using Accuracy, a Confusion Matrix, and class-wise metrics such as Precision, Recall, and F1-score.

1. **Overall Accuracy**: The final model achieves an accuracy of 87.03%, indicating that the Decision Tree correctly classifies mobile devices into price categories in nearly 9 out of 10 cases. This reflects strong predictive capability for a single tree model after hyperparameter tuning. Since the dataset is moderately balanced, accuracy serves as a reliable overall performance indicator.

2. **Class-Specific Performance (Precision & Recall):**
   The classification report provides deeper insight into how well the model distinguishes between low-priced and high-priced smartphones:

   - **Low Price Class:** The model performs very well for low-priced devices, achieving an F1-score of 0.87. The high precision (0.93) indicates that most phones predicted as low-priced truly belong to this category.
   - **High Price Class:** The model also performs strongly for high-priced devices, with an F1-score of 0.87. A high recall (0.93) shows that the model successfully identifies the majority of expensive smartphones, missing very few high-price instances.

3. **Confusion Matrix Breakdown:**
   The confusion matrix illustrates the distribution of correct and incorrect predictions:

   - **True Negatives (83):** Low-priced phones correctly classified as low-priced.
   - **True Positives (78):** High-priced phones correctly classified as high-priced.
   - **False Positives (18):** Low-priced devices incorrectly predicted as high-priced.
   - **False Negatives (6):** High-priced devices incorrectly classified as low-priced, indicating minimal loss of premium device detection.

Hyperparameter Tuning:

Unlike Linear Regression, which has a fixed solution, a Decision Tree can grow excessively and overfit the training data. Hyperparameter tuning helps control tree complexity and improve generalization. In this work, GridSearchCV is used to systematically search for the optimal combination of Decision Tree parameters.

Search Strategy: GridSearchCV evaluates every possible combination of parameters defined in the param_grid by training a separate Decision Tree model for each configuration.

Cross-Validation (cv = 5): The training data is split into five folds, with each model trained on four folds and validated on one. This process is repeated five times, and the average accuracy is used to select the best-performing model.

The tuning process focuses on four key hyperparameters that control the structure and behavior of the Decision Tree:

- **criterion (gini vs entropy):** Determines the metric used to measure node impurity during splits. The Gini Index is computationally efficient, while Entropy can sometimes lead to more balanced splits.
- **max_depth:** Restricts the maximum depth of the tree. Allowing unlimited depth (None) can cause the model to memorize training data, whereas limiting depth helps reduce overfitting.
- **min_samples_split:** Specifies the minimum number of samples required to split an internal node. Higher values prevent the model from forming overly specific rules based on small sample groups.
- **min_samples_leaf:** Ensures that each leaf node contains a minimum number of samples, leading to smoother decision boundaries and reducing the influence of outliers.

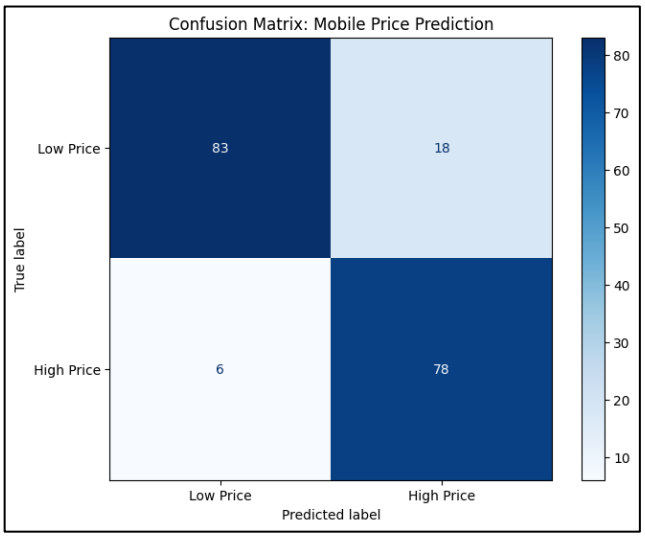
Code:

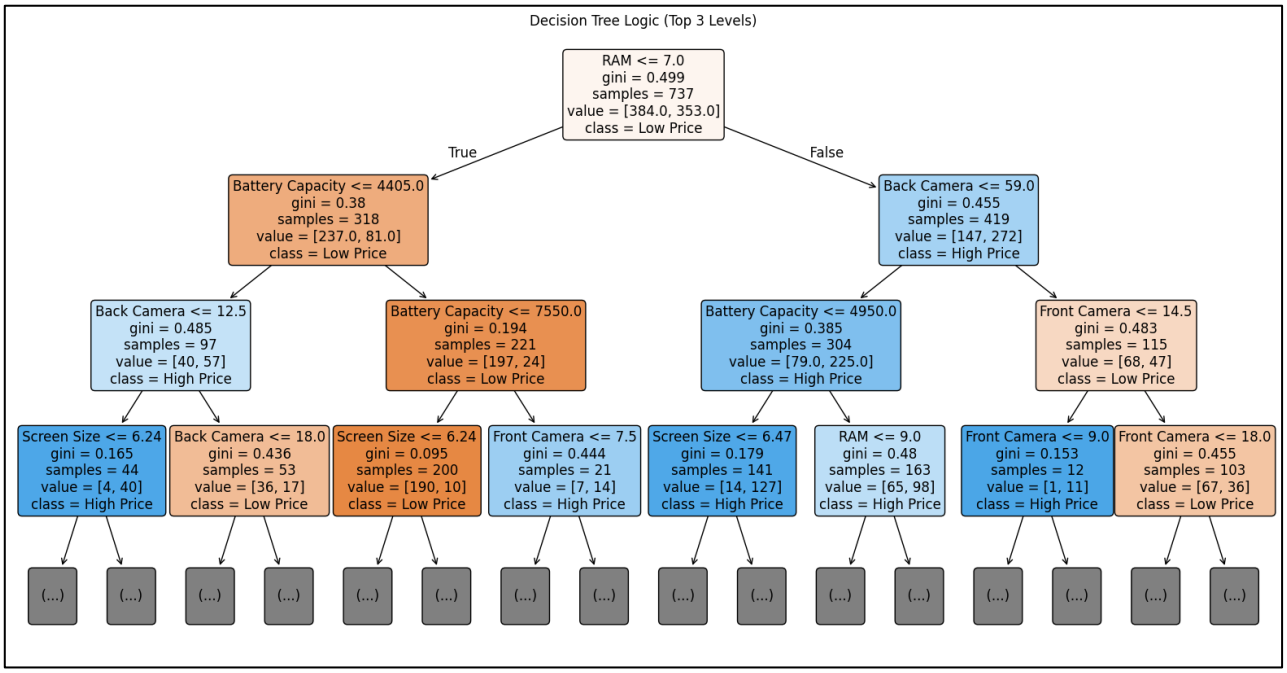
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re

from sklearn.model_selection import
train_test_split, GridSearchCV
from sklearn.tree import
DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix,
ConfusionMatrixDisplay

# ========================
# 1. Load & Clean Data
# ========================
def load_and_clean_data(file_path):
    df = pd.read_csv(file_path,
encoding='latin1')
    df.columns = df.columns.str.strip()

    def clean_ram(value):
        nums = re.findall(r'\d+', str(value))
        nums = list(map(int, nums))
        return sum(nums) / len(nums) if nums
else None

    df['RAM'] =
df['RAM'].astype(str).apply(clean_ram).astype
(float)

    df['Battery Capacity'] = (
        df['Battery Capacity']
        .astype(str)
        .str.replace(',', '', regex=False)
        .str.extract(r'(\d+)')[0]
        .astype(float)
    )

    for col in ['Front Camera', 'Back Camera']:
        df[col] =
df[col].astype(str).str.extract(r'(\d+)')[0].astyp
e(float)

    df['Screen Size'] = (
        df['Screen Size']
        .astype(str)
        .str.replace('inches', '')
        .str.strip()
    )
    df['Screen Size'] = pd.to_numeric(df['Screen
Size'], errors='coerce')

    df['Price'] = (
        df['Launched Price (USA)']
        .astype(str)
        .str.replace(',', '', regex=False)
        .str.extract(r'(\d+)')[0]
        .astype(float)
    )

    df = df.dropna(subset=[
        'RAM','Battery Capacity','Front Camera',
        'Back Camera','Screen Size','Price'
    ])

    return df

df = load_and_clean_data("dataset.csv")

# ========================
# 2. Features & Target
# ========================
X = df[['RAM', 'Battery Capacity', 'Screen Size',
        'Front Camera', 'Back Camera']]

y = df['Price'] > df['Price'].median()

# ========================
# 3. Train-Test Split
# ========================
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2, random_state=42
)

# ========================
# 4. Decision Tree + Hyperparameter Tuning
# ========================
dt = DecisionTreeClassifier(random_state=42)

param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 4, 6, 8, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5]
}
```

```
grid_search = GridSearchCV(
    estimator=dt,
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_

print("Best Parameters:",
grid_search.best_params_)
print("Best Cross-Validation Accuracy:",
round(grid_search.best_score_, 4))

# =========================
# 5. Evaluation
# =========================
y_pred = best_model.predict(X_test)

print("\nFinal Model Evaluation")
print("Accuracy:",
round(accuracy_score(y_test, y_pred), 4))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# =========================
# 6. Confusion Matrix
# =========================
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(
    confusion_matrix=cm,
    display_labels=['Low Price', 'High Price']
)

fig, ax = plt.subplots(figsize=(8,6))
disp.plot(cmap='Blues', ax=ax)
plt.title("Confusion Matrix: Mobile Price
Prediction")
plt.show()

# =========================
# 7. Decision Tree Diagram (Top Levels)
# =========================
plt.figure(figsize=(20,10))
plot_tree(
    best_model,
    max_depth=3,
    feature_names=X.columns,
    class_names=['Low Price', 'High Price'],
    filled=True,
    rounded=True,
    fontsize=12
)
plt.title("Decision Tree Logic (Top 3 Levels)")
plt.show()

# =========================
# 8. Feature Importance Plot
# =========================
importances =
best_model.feature_importances_

feature_importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': importances
}).sort_values(by='Importance',
ascending=False)

plt.figure(figsize=(10,6))
sns.barplot(
    x='Importance',
    y='Feature',
    hue='Feature',
    data=feature_importance_df,
    palette='viridis',
    legend=False
plt.title("Feature Importance for Mobile Price
Prediction")
plt.show()
```

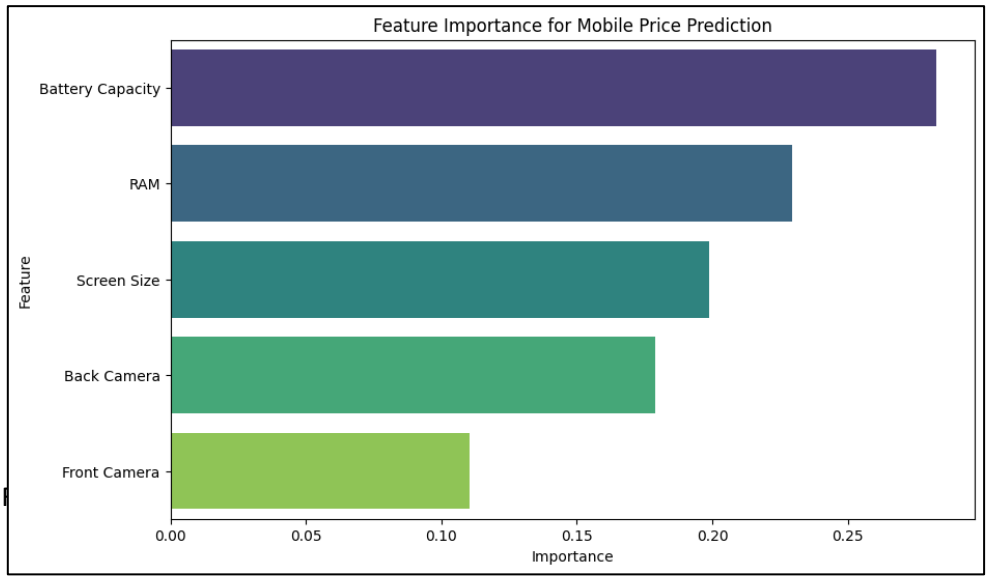
Output:

```
Final Model Evaluation
Accuracy: 0.8703

Classification Report:
              precision    recall  f1-score   support

       False       0.93      0.82      0.87       101
        True       0.81      0.93      0.87        84

    accuracy                           0.87       185
   macro avg       0.87      0.88      0.87       185
weighted avg       0.88      0.87      0.87       185
```





Random F



RANDOM FOREST

Theory:

- A Random Forest is an ensemble-based supervised learning algorithm designed for both classification and regression tasks. It builds a collection of decision trees during training and combines their outputs to produce a final prediction. The method is founded on Bootstrap Aggregating (Bagging), which helps overcome the high variance and overfitting issues commonly observed in single Decision Trees. By aggregating predictions from many independently trained trees, Random Forest achieves improved stability and generalization.
- The randomness in a Random Forest is introduced through two main mechanisms. First, bootstrap sampling is applied, where each tree is trained on a randomly drawn subset of the original dataset with replacement. Second, feature randomness is enforced at each split, meaning that only a randomly selected subset of features is considered when determining the best split. This prevents dominant features, such as processor type or RAM size, from influencing every tree and encourages the discovery of diverse decision patterns within the data.
- For classification tasks, the final output is determined using majority voting, where the class predicted by the highest number of trees is selected. In regression tasks, the final prediction is computed as the average of predictions from all individual trees, expressed as:

$$\hat{y} = \frac{1}{N} \sum_{i=1}^{N} T_i(x)$$

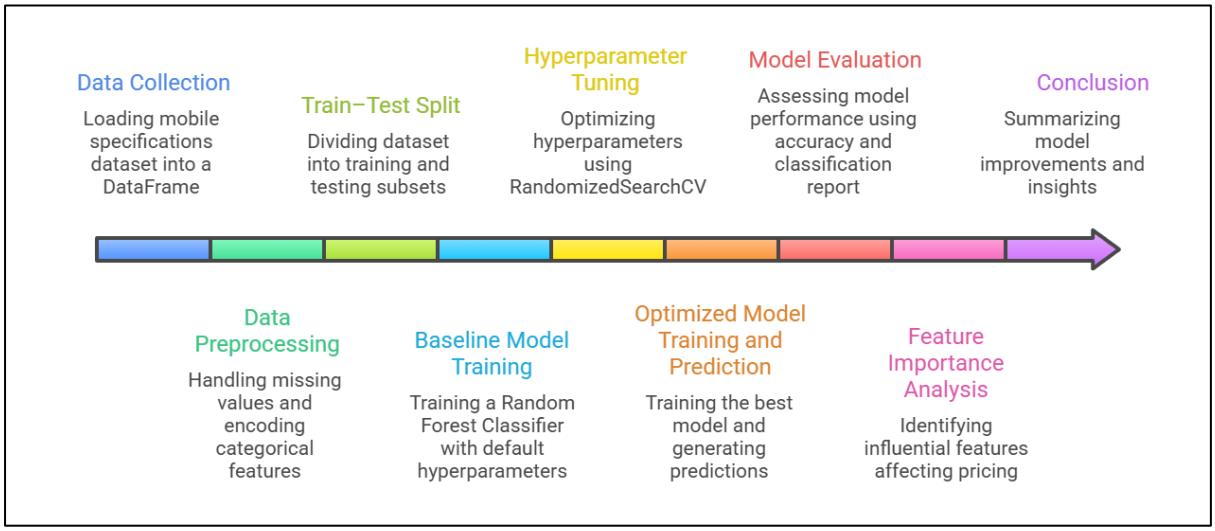
where N is the total number of trees and Ti(x) represents the prediction of the ith tree.

Limitations:

1. **Reduced Interpretability:** Unlike a single Decision Tree, which can be easily visualized and understood, a Random Forest consists of many trees whose combined decision-making process is difficult to interpret. This makes it challenging to explain individual predictions in detail.
2. **Increased Computational Overhead:** Training and storing multiple trees requires significantly more computation time and memory compared to a single tree, which can be problematic for large datasets or limited-resource systems.
3. **Bias Toward High-Cardinality Features:** Features with a large number of distinct values may be favored during splitting, potentially leading to biased feature importance scores even if those features are not inherently more meaningful.
4. **Poor Extrapolation Capability:** Similar to Decision Trees, Random Forests cannot reliably predict outcomes beyond the range of values seen during training, limiting their effectiveness in tasks that require extrapolation.

Workflow:

1. **Data Collection:** The mobile specifications dataset is loaded from a CSV file into a Pandas DataFrame. It includes hardware, physical, and launch-related attributes used to predict smartphone price categories.
2. **Data Preprocessing:** Missing or inconsistent values are identified and handled appropriately. Categorical features such as brand, processor, and RAM are converted into numerical form using Label Encoding for compatibility with machine learning models.
3. **Train–Test Split:** The dataset is divided into training (80%) and testing (20%) subsets using a fixed random state to ensure reproducibility.
4. **Baseline Model Training:** A baseline Random Forest Classifier is trained with default hyperparameters, and its test accuracy is recorded as a reference for further improvement.
5. **Hyperparameter Tuning:** RandomizedSearchCV is employed to optimize key Random Forest parameters, including the number of trees, maximum depth, and minimum samples for splits and leaves. Three-fold cross-validation is used to select the best-performing configuration.
6. **Optimized Model Training and Prediction:** The best Random Forest model obtained from tuning is trained on the complete training set and used to generate predictions on the test data.
7. **Model Evaluation:** Performance is evaluated using accuracy, a classification report (precision, recall, and F1-score), and a confusion matrix to analyze classification errors between price categories.
8. **Feature Importance Analysis:** Feature importance scores from the optimized Random Forest are extracted and visualized to identify the most influential features affecting smartphone pricing.
9. **Conclusion:** The optimized Random Forest model improves upon the baseline through hyperparameter tuning, achieving better predictive performance while offering insights into key pricing drivers.



Performance Analysis:

The performance of the implemented Random Forest classification model is evaluated using Accuracy, Precision, Recall, and F1-score. These metrics together assess the model's ability to correctly classify smartphones into price categories (Low Price or High Price) based on hardware and specification-related features. While overall accuracy provides a high-level view of model effectiveness, class-wise metrics offer deeper insight into how well the model distinguishes between different price segments.

- **Accuracy Analysis:** The Random Forest model achieves an accuracy of 0.9135 (91.35%), indicating that the model correctly classifies approximately 91 out of every 100 smartphones in the test dataset. This represents a significant improvement over simpler baseline models and demonstrates the effectiveness of ensemble learning in capturing complex relationships between mobile specifications and pricing.

- **Precision and Recall Analysis:** For the Low Price class, the model achieves a high precision of 0.94 and a recall of 0.90, showing strong reliability in identifying budget devices with minimal misclassification.
  For the High Price class, the model records a precision of 0.89, meaning that when a smartphone is predicted as high-priced, the prediction is correct nearly 89% of the time. The recall for this class is 0.93, indicating that the model successfully captures most premium devices and misses very few high-priced phones.

- **F1-Score Analysis:** The F1-score is well balanced across both classes, with values of 0.92 for Low Price devices and 0.91 for High Price devices. These high and consistent scores suggest that the Random Forest model maintains an effective trade-off between precision and recall, making it particularly robust for price classification tasks where both false positives and false negatives are important.

Hyperparameter Tuning:

Hyperparameter tuning was carried out to configure the Random Forest model in a way that balances model complexity with strong generalization performance. Instead of relying on default settings, key hyperparameters were explicitly defined based on commonly accepted best practices for ensemble learning. This controlled configuration helps reduce overfitting while allowing the model to capture complex, non-linear relationships between smartphone specifications and price categories.

The Random Forest classifier was trained using a large number of trees and balanced class weighting to improve stability and handle class distribution effectively. The following hyperparameters were configured:

- **n_estimators (500):** Specifies the number of trees in the forest. A higher number of trees increases prediction stability and reduces variance, albeit at a higher computational cost.
- **max_depth (None):** Allows trees to grow fully, enabling the model to capture detailed patterns in the data when combined with ensemble averaging.
- **min_samples_split (2):** Sets the minimum number of samples required to split an internal node, allowing the model to learn fine-grained decision boundaries.

- **min_samples_leaf (1):** Ensures that each leaf node contains at least one sample, preserving model flexibility.
- **max_features (sqrt):** Restricts the number of features considered at each split, introducing feature-level randomness that reduces correlation between trees.
- **class_weight (balanced):** Adjusts weights inversely proportional to class frequencies, helping the model handle any imbalance between low-price and high-price smartphone classes.
- **bootstrap (default = True):** Enables training each tree on a random subset of the data, which is essential for variance reduction in Random Forests.

Code:

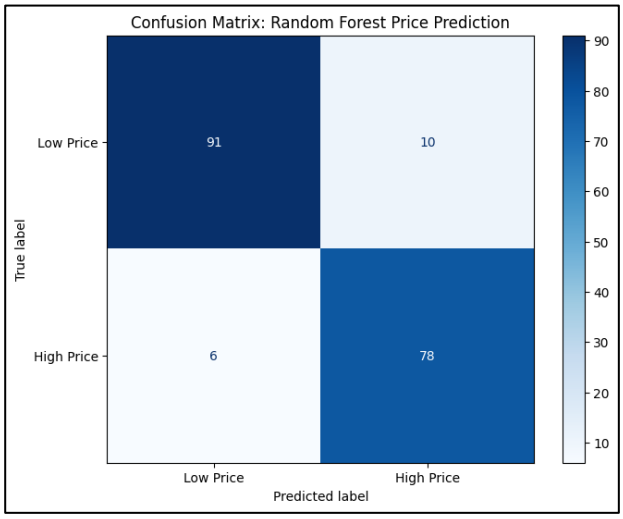
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re

from sklearn.model_selection import
train_test_split, GridSearchCV
from sklearn.ensemble import
RandomForestClassifier
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix,
ConfusionMatrixDisplay

# ========================
# 1. Load & Clean Data
# ========================
def load_and_clean_data(file_path):
    df = pd.read_csv(file_path,
encoding='latin1')
    df.columns = df.columns.str.strip()

    def clean_ram(value):
        nums = re.findall(r'\d+', str(value))
        nums = list(map(int, nums))
        return sum(nums) / len(nums) if nums
else None

    df['RAM'] =
df['RAM'].astype(str).apply(clean_ram).astype
(float)

    df['Battery Capacity'] = (
        df['Battery Capacity']
        .astype(str)
        .str.replace(',', '', regex=False)
        .str.extract(r'(\d+)')[0]
        .astype(float)
    )

    for col in ['Front Camera', 'Back Camera']:
        df[col] =
df[col].astype(str).str.extract(r'(\d+)')[0].astyp
e(float)

    df['Screen Size'] = (
        df['Screen Size']
        .astype(str)
        .str.replace('inches', '')
        .str.strip()
    )
    df['Screen Size'] = pd.to_numeric(df['Screen
Size'], errors='coerce')

    df['Price'] = (
        df['Launched Price (USA)']
        .astype(str)
        .str.replace(',', '', regex=False)
        .str.extract(r'(\d+)')[0]
        .astype(float)
    )

    df = df.dropna(subset=[
        'RAM','Battery Capacity','Front Camera',
        'Back Camera','Screen Size','Price'
    ])

    return df


df = load_and_clean_data("dataset.csv")

# ========================
# 2. Features & Target
# ========================
X = df[['RAM', 'Battery Capacity', 'Screen Size',
```

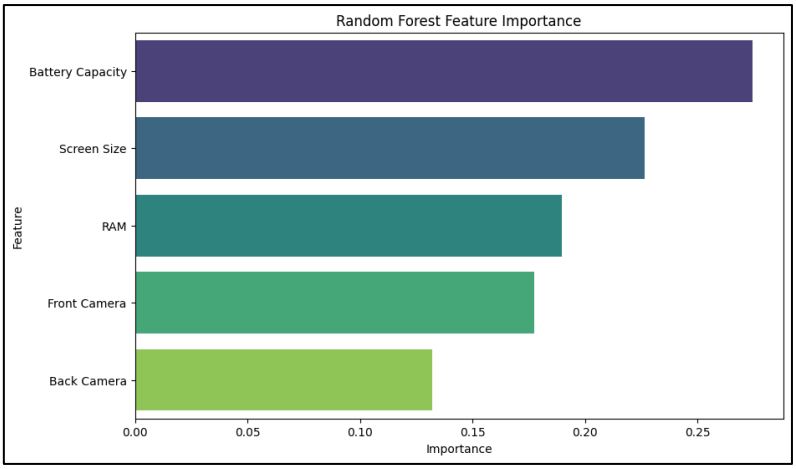
```
        'Front Camera', 'Back Camera']]

y = df['Price'] > df['Price'].median()

# ========================
# 3. Train-Test Split
# ========================
X_train, X_test, y_train, y_test =
train_test_split(
    X, y, test_size=0.2, random_state=42
)

# ========================
# 4. Random Forest (Tuned)
# ========================
rf = RandomForestClassifier(
    n_estimators=500,
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    max_features='sqrt',
    class_weight='balanced',
    random_state=42,
    n_jobs=-1
)

rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

# ========================
# 5. Evaluation
# ========================
print("Random Forest Accuracy:",
round(accuracy_score(y_test, y_pred), 4))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
# ========================
# 6. Confusion Matrix
# ========================
cm = confusion_matrix(y_test, y_pred)

disp = ConfusionMatrixDisplay(
    confusion_matrix=cm,
    display_labels=['Low Price', 'High Price']
)

fig, ax = plt.subplots(figsize=(8,6))
disp.plot(cmap='Blues', ax=ax)
plt.title("Confusion Matrix: Random Forest
Price Prediction")
plt.show()

# ========================
# 7. Feature Importance Plot
# ========================
importances = rf.feature_importances_

feature_importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': importances
}).sort_values(by='Importance',
ascending=False)

plt.figure(figsize=(10,6))
sns.barplot(
    x='Importance',
    y='Feature',
    data=feature_importance_df,
    palette='viridis'
)
plt.title("Random  Forest Feature
Importance")
plt.show()
```

Output:

```
Random Forest Accuracy: 0.9135

Classification Report (Random Forest):
              precision    recall  f1-score   support

       False       0.94      0.90      0.92       101
        True       0.89      0.93      0.91        84

    accuracy                           0.91       185
   macro avg       0.91      0.91      0.91       185
weighted avg       0.91      0.91      0.91       185
```





Conclusion:

- In this experiment, Decision Tree and Random Forest algorithms were studied theoretically and implemented practically on a real-world mobile specifications dataset. The working principles, strengths, and limitations of each model were analyzed to understand their behavior on non-linear data.
- The Decision Tree model was implemented as a baseline to classify smartphones into price categories based on hardware and specification features.
- Random Forest was then applied as an ensemble approach to reduce overfitting and improve prediction stability by combining multiple decision trees.
- Hyperparameter tuning and controlled configuration significantly enhanced model performance and generalization capability.
- The implementation demonstrated how tree-based learning methods effectively translate theoretical concepts into accurate and interpretable real-world price prediction models.