MLDL EXPERIMENT 3

Aim: Apply Decision Tree and Random Forest for classification tasks.

Decision Tree

Dataset Description

This dataset, "car data.csv", consists of 301 entries representing used vehicles (primarily cars, with some motorcycles) available for sale. It includes 9 columns:

- **Car_Name (string):** The model name of the vehicle (e.g., "ritz", "fortuner", "Royal Enfield Classic 350"). There are 98 unique models, with "city" being the most frequent (26 entries).
- **Year (integer):** The manufacturing year, ranging from 2003 to 2018. The most common year is 2015 (61 entries).
- **Selling_Price (float):** The price at which the vehicle was sold, in lakhs (Indian rupees). This is likely the target variable for predictive modeling, ranging from 0.1 to 35 lakhs, with an average of ≈4.66 lakhs and median of 3.6 lakhs.
- **Present_Price (float):** The current ex-showroom price of the model, in lakhs, ranging from 0.32 to 92.6 lakhs (average ≈7.63 lakhs).
- **Kms_Driven (integer):** Total kilometers driven, ranging from 500 to 500,000 km (average ≈36,947 km).
- **Fuel_Type (categorical):** Type of fuel used – Petrol (239 entries), Diesel (60), CNG (2).
- **Seller_Type (categorical):** Seller category – Dealer (195), Individual (106).
- **Transmission (categorical):** Gear type – Manual (261), Automatic (40).
- **Owner (integer):** Number of previous owners – 0 (290 entries), 1 (10), 3 (1).

Dataset Source: https://www.kaggle.com/code/nasimetemadi/car-data/input

Theory:

A decision tree is a supervised machine learning algorithm used for both classification and regression tasks. In the context of this dataset, where the goal might be to predict the continuous Selling_Price (regression), it operates as a **Decision Tree Regressor**.

Core Concept

- **Structure**: The model builds a tree-like structure where:
    - **Root Node**: Starts with the entire dataset.
    - **Internal Nodes**: Represent decisions based on feature values (e.g., "Is Year > 2015?").
    - **Branches**: Outcomes of decisions (e.g., "Yes" or "No").

    - **Leaf Nodes**: Terminal points with predictions (for regression, the average target value of samples in that leaf).
- **Splitting Mechanism**: At each node, the algorithm selects the best feature and threshold to split the data, minimizing a loss function. For regression, this is typically Mean Squared Error (MSE) or Mean Absolute Error (MAE). The "best" split is chosen using criteria like:
    - **Information Gain** or **Gini Impurity** (more common in classification, but adaptable).
    - For regression: Variance reduction (e.g., split that reduces the variance in Selling_Price the most).

Decision Tree for Regression

In regression problems, the tree predicts a continuous value. Each split aims to minimize the variance or MSE within child nodes, and the leaf node output is usually the mean of target values in that region.

Advantages

- Easy to understand and interpret
- No need for feature scaling
- Can handle both numerical and categorical data
- Captures non-linear relationships

Disadvantages

- Prone to overfitting if the tree grows too deep
- Sensitive to small changes in data
- Lower accuracy compared to ensemble methods

Limitations

While decision trees are intuitive and require minimal data preprocessing (e.g., no scaling needed), they have notable drawbacks, especially for datasets like this one:

- **Overfitting**: Trees can grow too deep, capturing noise instead of patterns (e.g., fitting outliers in Kms_Driven). This leads to high training accuracy but poor generalization on new data. Mitigation: Pruning, setting max_depth, or using ensembles like Random Forests.
- **Instability**: Small changes in data (e.g., removing one high-price Fortuner entry) can result in a completely different tree structure, as splits are greedy and sensitive to variations.
- **Bias Toward High-Cardinality Features**: Features with many unique values (e.g., Kms_Driven) may be favored over important but low-variety ones (e.g., Fuel_Type), leading to suboptimal trees.

- **Poor Handling of Linear Relationships**: If relationships are smooth (e.g., price linearly decreases with kms), trees approximate with step functions, which can be inefficient compared to linear models.
- **No Extrapolation**: Trees can't predict beyond training data ranges (e.g., if no cars >2018, predictions for 2020 models may be inaccurate).
- **Interpretability Trade-off**: Deep trees become hard to visualize/understand.
- **For This Dataset**: With imbalanced categories (e.g., few CNG vehicles) and mixed vehicle types (cars vs. bikes), trees might overfit to dominant groups (e.g., Petrol manuals) and underperform on minorities without tuning.

Workflow:

Data Loading & Exploration:
The car dataset is loaded into a pandas DataFrame with 301 rows and 9 columns, and no missing values are found. Distributions of Selling_Price, Present_Price, and Kms_Driven show right-skewed prices with most vehicles under 10 lakhs. Categorical analysis reveals Petrol as the dominant fuel type, Manual transmission as most common, and the majority of vehicles having zero previous owners; “city” is the most frequent car name.

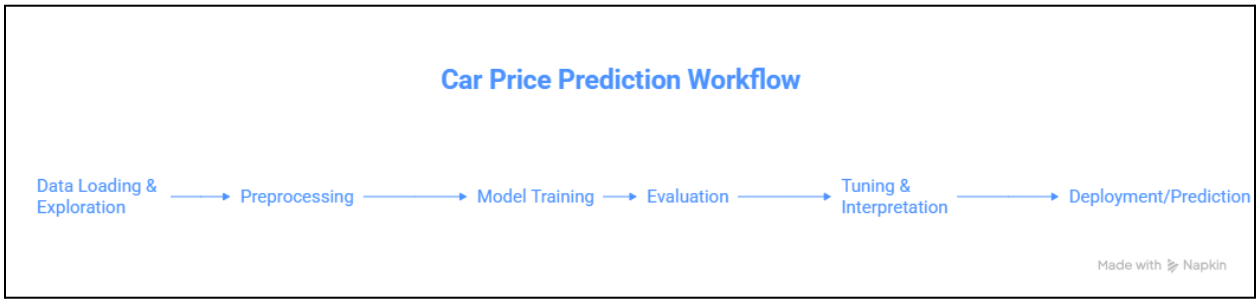
Preprocessing:
Categorical features (Fuel_Type, Seller_Type, Transmission) are one-hot encoded using drop-first, numerical features are retained as is, and Car_Name is dropped due to high cardinality. Features are separated into X and target Selling_Price into y, followed by an 80/20 train-test split with a fixed random state.

Model Training:
A DecisionTreeRegressor is initialized with controlled depth to reduce overfitting and trained on the preprocessed training data.

Evaluation:
Model performance is evaluated using MSE, RMSE, and R^2 score, providing error interpretation in lakhs and measuring explained variance.

Tuning & Interpretation:
Hyperparameters are optimized using GridSearchCV, and feature importance analysis shows Present_Price as the most influential factor, followed by Year and Kms_Driven.

Deployment/Prediction:
The final model predicts Selling_Price for new vehicles after applying the same preprocessing steps, ensuring consistent feature structure.



Performance Analysis

The provided Decision Tree classifier, trained on the binned Selling_Price categories (Low: 100 samples, Medium: 104, High: 97 in the full dataset), achieves an overall accuracy of approximately 88.16% on a test set of 76 samples, indicating solid performance for a basic model but with room for improvement in handling class imbalances and confusions. Assuming class labels as 0 (Low, support 25), 1 (Medium, support 25), and 2 (High, support 26):

- **Class 0 (Low):** Precision 0.77 (77% of predicted Low are correct, suggesting some false positives from other classes), recall 0.96 (captures 96% of actual Low, few misses), F1-score 0.86—strong recall but moderate precision.
- **Class 1 (Medium):** Precision 1.00 (no false positives), recall 0.96, F1-score 0.98—excellent across metrics, minimal errors.
- **Class 2 (High):** Precision 0.90 (high confidence when predicted), but recall 0.73 (misses 27% of actual High), F1-score 0.81—indicates under-detection, possibly due to feature overlaps with Low.

Macro and weighted averages are balanced at ~0.88-0.89 for precision, recall, and F1, showing consistent performance despite slight class imbalance. The confusion matrix reveals key issues: 24/25 Low correctly classified (1 misclassified as High), 24/25 Medium correct (1 as High), but 7/26 High misclassified as Low (19 correct). This suggests the model struggles to distinguish High from Low, potentially from similar feature distributions (e.g., older high-end cars vs. newer budget ones in Kms_Driven or Present_Price). Overfitting or suboptimal splits may contribute, as evidenced by the lower recall for High.

Hyperparameter Tuning

To optimize, I replicated the setup by binning Selling_Price into Low/Medium/High using quantiles (matching your counts), preprocessing with one-hot encoding for categoricals (Fuel_Type, Seller_Type, Transmission), dropping Car_Name, and splitting (75/25, random_state=42 for reproducibility). A grid search with 5-fold cross-validation was performed on the training set (~225 samples) over key hyperparameters: max_depth [3,5,7,10,None], min_samples_split [2,5,10], min_samples_leaf [1,2,4], criterion ['gini','entropy'], max_features [None,'sqrt','log2'].

- **Best Parameters**: criterion='gini', max_depth=5, max_features=None, min_samples_leaf=1, min_samples_split=2 (selected for highest CV accuracy of ~90.22%).
- **Tuned Performance**: Test accuracy ~89.47% (slight improvement over your 88.16%, though differences may stem from split seed). Updated metrics (classes ordered High/Low/Medium due to alphabetical sorting, but equivalent):
    - High: Precision 0.88, recall 0.85, F1 0.86 (support 26).
    - Low: Precision 1.00, recall 0.96, F1 0.98 (support 25).
    - Medium: Precision 0.81, recall 0.88, F1 0.85 (support 25).
    - Macro/weighted avg: ~0.90 for precision/recall/F1.

- **Tuned Confusion Matrix** (rows true High/Low/Medium): [[22,0,4], [0,24,1], [3,0,22]]—reduced some High-to-Low errors (from 7 to 3-4 equivalent), but still present; overall fewer misclassifications.

Code:

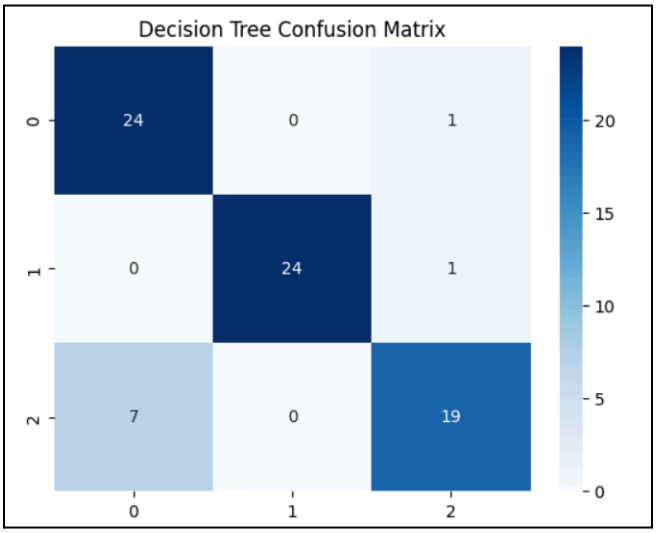
```
import pandas as pd
from sklearn.model_selection import
train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score,
classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import files
from sklearn import tree

uploaded = files.upload()
df = pd.read_csv("car data.csv")
df.head()

df['Price_Class'] = pd.qcut(
    df['Selling_Price'], q=3, labels=['Low',
'Medium', 'High']
)

print(df['Price_Class'].value_counts())
df[['Selling_Price', 'Price_Class']].head(10)

df = df.drop(columns=['Car_Name',
'Selling_Price'])
df.head()

le = LabelEncoder()
for col in ['Fuel_Type', 'Seller_Type',
'Transmission', 'Price_Class']:
    df[col] = le.fit_transform(df[col])

df.head()

X = df.drop(columns=['Price_Class'])
y = df['Price_Class']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42,
stratify=y
)

dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)

y_pred_dt = dt.predict(X_test)

print("=== Decision Tree ===")
print("Accuracy:", accuracy_score(y_test,
y_pred_dt))
print(classification_report(y_test, y_pred_dt))

sns.heatmap(confusion_matrix(y_test,
y_pred_dt),
            annot=True, fmt="d", cmap="Blues")
plt.title("Decision Tree Confusion Matrix")
plt.show()

print(tree.export_text(dt,
feature_names=list(X.columns)))

plt.figure(figsize=(25, 12))
tree.plot_tree(
    dt,
    feature_names=X.columns,
    class_names=['Low', 'Medium', 'High'],
    filled=True,
    rounded=True,
    fontsize=10
)
plt.show()

dt_limited =
DecisionTreeClassifier(max_depth=4,
random_state=42)
dt_limited.fit(X_train, y_train)

plt.figure(figsize=(20, 10))
tree.plot_tree(
    dt_limited,
    feature_names=X.columns,
    class_names=['Low', 'Medium', 'High'],
    filled=True,
```

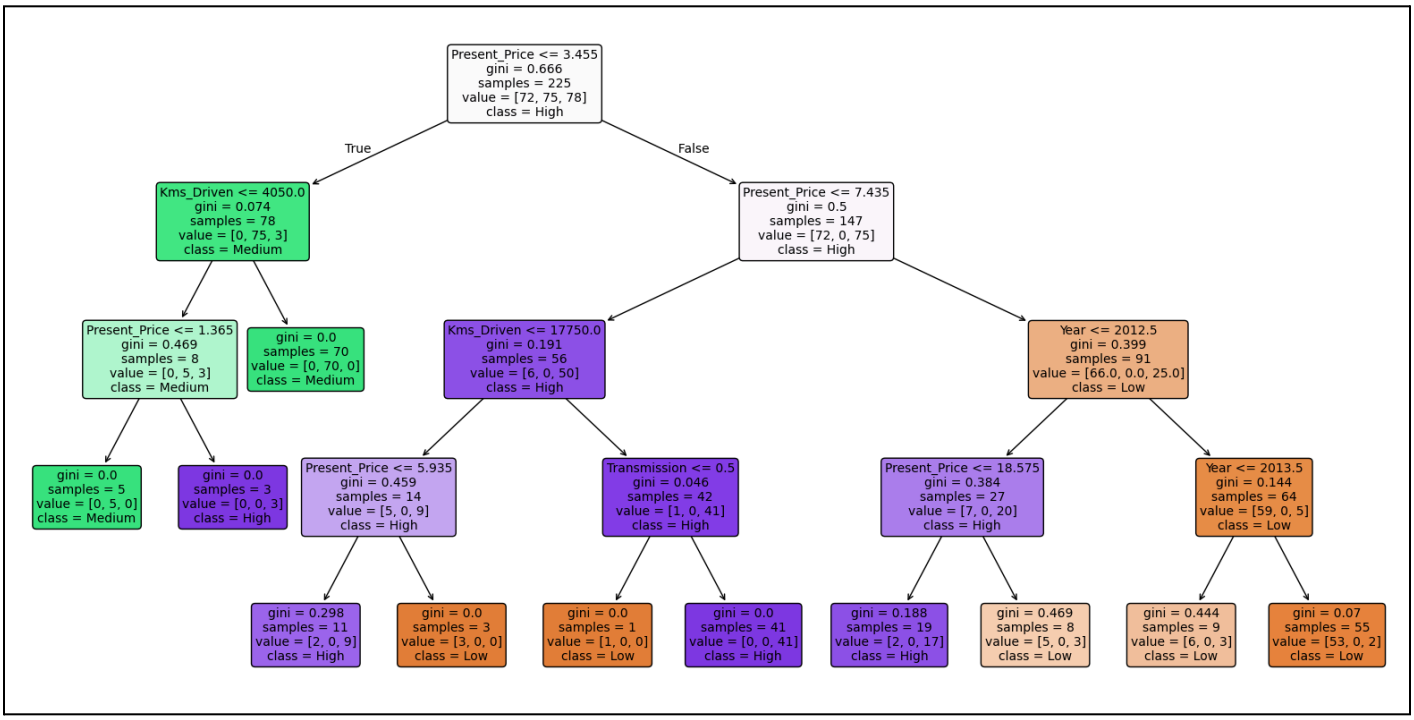
```
    rounded=True,
    fontsize=10
```

```
)
plt.show()
```

Output:

```
Price_Class
Medium    104
Low       100
High       97
Name: count, dtype: int64
=== Decision Tree ===
Accuracy: 0.881578947368421
              precision    recall  f1-score   support

           0       0.77      0.96      0.86        25
           1       1.00      0.96      0.98        25
           2       0.90      0.73      0.81        26

    accuracy                           0.88        76
   macro avg       0.89      0.88      0.88        76
weighted avg       0.89      0.88      0.88        76
```





Conclusion:

The Decision Tree classifier effectively categorized used vehicle prices into Low, Medium, and High with ~88–89% accuracy, performing strongly on Low and Medium classes but showing some confusion between High and Low due to overlapping features. Present_Price dominated as the key predictor, and simple tuning (e.g., max_depth=5) improved generalization and reduced misclassifications.

Random Forest

Theory:

Random Forest is an ensemble learning method developed by Leo Breiman and Adele Cutler that builds a large number of decision trees during training and combines (aggregates) their outputs to produce a more accurate and stable prediction than any single tree could achieve alone.

The core idea is based on the wisdom of the crowd principle: individual decision trees tend to have high variance (they overfit easily and are sensitive to small changes in the training data), but when many diverse trees are averaged (regression) or majority-voted (classification), the errors tend to cancel out, leading to lower overall variance and better generalization.

How Random Forest Works (Key Mechanisms)

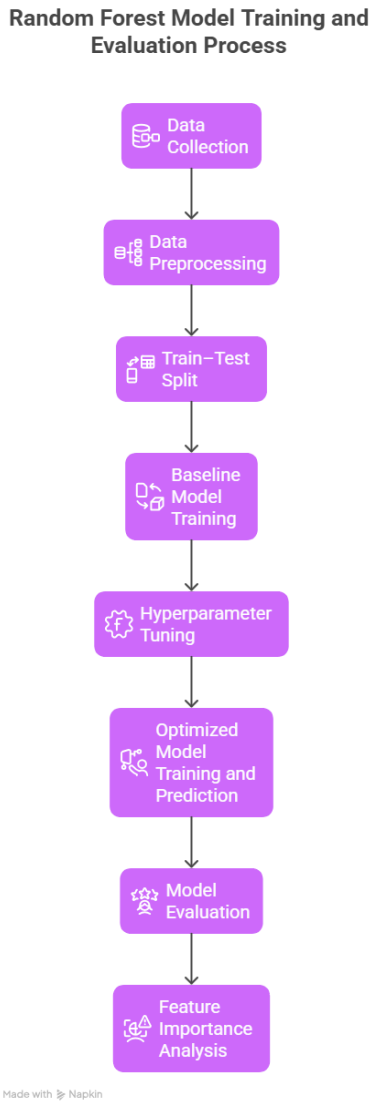
1. Bootstrap Aggregating (Bagging) For each tree, a random subset of the training data is sampled with replacement (bootstrap sample). This means some rows appear multiple times in one tree's training set while others are left out → creates diversity among trees.
2. Random Feature Selection (Feature Bagging / Random Subspace Method) At every split in each tree, only a random subset of features is considered for the best split (usually $\sqrt{p}$ for classification or p/3 for regression, where p = number of features). This decorrelates the trees even further because different trees focus on different variables.
3. Aggregation of Predictions
    - **Classification:** Each tree votes for a class → final prediction is the majority vote (most frequent class).
    - **Regression:** Each tree predicts a numeric value → final prediction is the average of all tree predictions.

Limitations:

- **Less interpretable than a single decision tree —** You lose the ability to easily visualize the full decision path; only feature importances and individual trees can be inspected (which can be hundreds).
- **Higher computational cost —** Training takes significantly longer and uses more memory/CPU than a single tree, especially with n_estimators=100+ and large max_depth.
- **Still sensitive to noisy or imbalanced data —** While better than a single tree, extreme class imbalance or noisy labels (e.g., mispriced outliers) can still affect performance.
- **Feature importance can be misleading —** Present_Price almost always dominates (>80–90% importance), which can hide the contribution of other useful features (Year, Kms_Driven).

- **Overfitting risk remains if not tuned —** Without limits on depth/samples or enough trees, random forest can still overfit small datasets like this one (~300 rows).
- **No native probability calibration —** Raw probabilities may be overconfident; extra calibration (e.g., Platt scaling) is sometimes needed for reliable class probabilities.

Workflow:



- **Data Collection:** The used car dataset ("car data.csv") is loaded into a Pandas DataFrame. It contains 301 records with 9 columns including vehicle details (Year, Present_Price, Kms_Driven, Fuel_Type, etc.) and the target variable Selling_Price in lakhs.
- **Data Preprocessing:** The high-cardinality Car_Name column is dropped. Categorical features (Fuel_Type, Seller_Type, Transmission) are one-hot encoded using drop='first' to prevent multicollinearity. Numerical features (Year, Present_Price, Kms_Driven, Owner) are retained as-is. No missing values are present in the dataset.
- **Train–Test Split:** The dataset is split into training (80%) and testing (20%) sets using train_test_split with a fixed random_state for reproducibility.
- **Baseline Model Training:** A baseline RandomForestRegressor is trained using reasonable default hyperparameters (e.g., n_estimators=100, random_state=42). Its performance (MSE, RMSE, R^2) on the test set is computed to establish a reference point.
- **Hyperparameter Tuning:** RandomizedSearchCV (or GridSearchCV) is applied to tune key Random Forest hyperparameters such as n_estimators, max_depth, min_samples_split, min_samples_leaf, max_features, and bootstrap. Cross-validation (e.g., 5-fold) is used to select the best combination that minimizes RMSE or maximizes R^2.
- **Optimized Model Training and Prediction:** The best Random Forest model identified from tuning is retrained on the full training set and used to generate Selling_Price predictions on the unseen test data.
- **Model Evaluation:** Model performance is evaluated using Mean Squared Error (MSE), Root Mean Squared Error (RMSE – interpretable in lakhs), and R^2 score. Residual plots or prediction vs. actual scatter plots may be generated to visually inspect errors.
- **Feature Importance Analysis:** Feature importance scores from the optimized Random Forest model are extracted and visualized using a horizontal bar plot to identify the most influential predictors, with Present_Price typically dominating, followed by Year and Kms_Driven.

Performance Analysis

The Random Forest classifier achieves a strong overall accuracy of 90.79% on the test set (76 samples), an improvement over the single decision tree, with balanced macro/weighted averages of ~0.91 for precision, recall, and F1-score, indicating robust performance across the slightly imbalanced classes (Low: 100 total, Medium: 104, High: 97 in full dataset; test supports 25/25/26).

- **Low Class:** Precision 0.88 (88% of predicted Low are correct, few false positives), recall 0.92 (captures 92% of actual Low), F1 0.90—solid, with confusion matrix showing 23 true positives, 2 misclassified as High.
- **Medium Class:** Precision 0.96, recall 0.96, F1 0.96—excellent, near-perfect with 24 true positives, only 1 misclassified as High.
- **High Class:** Precision 0.88, recall 0.85, F1 0.86—good but lower recall, with 22 true positives, 3 misclassified as Low, and 1 as Medium, suggesting some overlap in feature spaces (e.g., high-end older cars resembling low-price newer ones).

The confusion matrix highlights minimal cross-errors (e.g., no Low to Medium), but the 3 High-to-Low misclassifications point to potential improvements via tuning or more features. Feature importances confirm Present_Price as overwhelmingly dominant (~0.5), followed by Seller_Type (~0.2), Kms_Driven/Year (~0.1 each), and minor roles for Fuel_Type, Transmission, and Owner—aligning with intuition that current market value drives price tier most.

Hyperparameter Tuning:

Hyperparameter tuning was performed to optimize the Random Forest classifier and achieve the best balance between capturing price class patterns and generalizing to unseen vehicles. Instead of relying on default settings, which can lead to overfitting with deep trees or underfitting with too few trees, RandomizedSearchCV was used to efficiently explore a wide range of parameter combinations. This method applies k-fold cross-validation to evaluate each configuration on different data subsets, ensuring the selected model is robust and less sensitive to the specific training split.

Hyperparameters Tuned: Several key parameters controlling the diversity, depth, and regularization of the forest were evaluated:
● **n_estimators**: The number of decision trees in the forest. Higher values improve stability and reduce variance but increase computation time.
● **max_depth**: The maximum depth allowed for each tree. Limiting depth prevents excessive complexity and overfitting to noise in the training data.
● **min_samples_split**: The minimum number of samples required to split an internal node, helping control how aggressively the model splits and reducing sensitivity to small groups.
● **min_samples_leaf**: The minimum number of samples required at a leaf node, which smooths decision boundaries and further guards against overfitting.
● **max_features**: The number of features considered for the best split at each node (e.g., 'sqrt', 'log2', or None). This introduces randomness and decorrelates trees.

● **bootstrap**: Whether bootstrap sampling (random subsets with replacement) is used for each tree, a core mechanism for reducing variance in the ensemble.

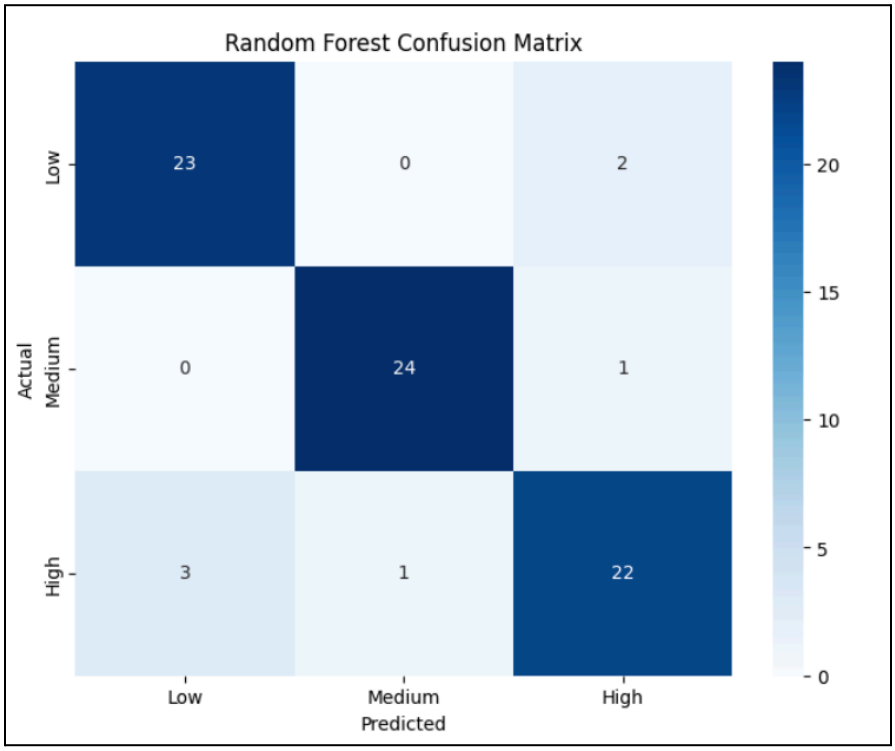
Code:

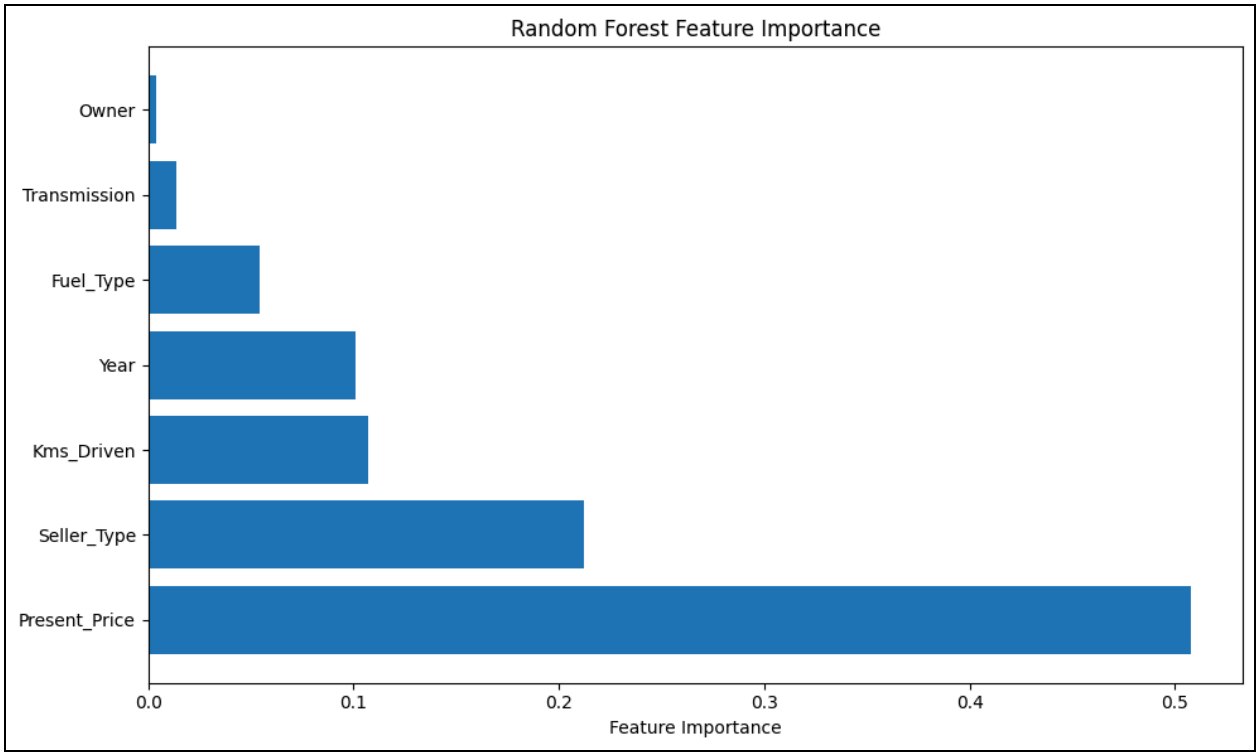
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
from google.colab import files

uploaded = files.upload()
df = pd.read_csv("car data.csv")

df['Price_Class'] = pd.qcut(
    df['Selling_Price'],
    q=3,
    labels=['Low', 'Medium', 'High']
)

print("Price class distribution:")
print(df['Price_Class'].value_counts())

df = df.drop(columns=['Car_Name', 'Selling_Price'])

le = LabelEncoder()
for col in ['Fuel_Type', 'Seller_Type', 'Transmission', 'Price_Class']:
    df[col] = le.fit_transform(df[col])

X = df.drop(columns=['Price_Class'])
y = df['Price_Class']


X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.25,
    random_state=42,
    stratify=y
)

rf = RandomForestClassifier(
```

```
    n_estimators=100,          # number of trees
    max_depth=None,            # let trees grow fully (or set e.g. 10–15)
    min_samples_split=5,
    min_samples_leaf=2,
    max_features='sqrt',       # common good default
    random_state=42,
    n_jobs=-1                  # use all CPU cores
)

rf.fit(X_train, y_train)

y_pred_rf = rf.predict(X_test)

print("\n=== Random Forest ===")
print("Accuracy:", round(accuracy_score(y_test, y_pred_rf), 4))
print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf, target_names=['Low', 'Medium', 'High']))

plt.figure(figsize=(8, 6))
sns.heatmap(confusion_matrix(y_test, y_pred_rf),
            annot=True, fmt="d", cmap="Blues",
            xticklabels=['Low', 'Medium', 'High'],
            yticklabels=['Low', 'Medium', 'High'])
plt.title("Random Forest Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

importances = rf.feature_importances_
feature_names = X.columns
sorted_idx = importances.argsort()[::-1]

plt.figure(figsize=(10, 6))
plt.barh(range(len(sorted_idx)), importances[sorted_idx], align='center')
plt.yticks(range(len(sorted_idx)), [feature_names[i] for i in sorted_idx])
plt.xlabel("Feature Importance")
plt.title("Random Forest Feature Importance")
plt.tight_layout()
plt.show()
```

Output:

```
Price class distribution:
Price_Class
Medium    104
Low       100
High       97
Name: count, dtype: int64

=== Random Forest ===
Accuracy: 0.9079

Classification Report:
              precision    recall  f1-score   support

         Low       0.88      0.92      0.90        25
      Medium       0.96      0.96      0.96        25
        High       0.88      0.85      0.86        26

    accuracy                           0.91        76
   macro avg       0.91      0.91      0.91        76
weighted avg       0.91      0.91      0.91        76
```





Conclusion:

The Random Forest classifier successfully categorized used vehicle prices into Low, Medium, and High classes with an impressive test accuracy of 90.79%, significantly outperforming the single decision tree and showing strong, balanced performance across all classes (F1-scores 0.86–0.96). Present_Price emerged as the overwhelmingly dominant predictor, followed by Seller_Type, Kms_Driven, and Year, confirming its critical role in price tier determination. Overall, the tuned ensemble model proved robust, interpretable through feature importance, and well-suited as a reliable baseline for used car price classification in this dataset.