Comprehensive Assessment : Machine Learning

In [1]:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

In [2]:

```
# Step 1: Load and Preprocess Data
#Loading and Preprocessing: The dataset is loaded, and initial data exploration is performed. Missing values and categorical
url = 'https://drive.google.com/uc?id=1FHmYNLs9v0Enc-UExEMpitOFGsWvB2dP'  # Direct link
data = pd.read_csv(url)
```

In [3]:

```
# Explore the dataset
print(data.info())
print(data.describe())
print(data.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   car_ID            205 non-null    int64  
 1   symboling         205 non-null    int64  
 2   CarName           205 non-null    object 
 3   fueltype          205 non-null    object 
 4   aspiration        205 non-null    object 
 5   doornumber        205 non-null    object 
 6   carbody           205 non-null    object 
 7   drivewheel        205 non-null    object 
 8   enginelocation    205 non-null    object 
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64  
 14  enginetype        205 non-null    object 
 15  cylindernumber    205 non-null    object 
 16  enginesize        205 non-null    int64  
 17  fuelsystem        205 non-null    object 
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64  
 22  peakrpm           205 non-null    int64  
 23  citympg           205 non-null    int64  
 24  highwaympg        205 non-null    int64  
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
None
           car_ID   symboling   wheelbase   carlength    carwidth   carheight  \
count  205.000000  205.000000  205.000000  205.000000  205.000000  205.000000   
mean   103.000000    0.834146   98.756585  174.049268   65.907805   53.724878   
std     59.322565    1.245307    6.021776   12.337289    2.145204    2.443522   
min      1.000000   -2.000000   86.600000  141.100000   60.300000   47.800000   
25%     52.000000    0.000000   94.500000  166.300000   64.100000   52.000000   
50%    103.000000    1.000000   97.000000  173.200000   65.500000   54.100000   
75%    154.000000    2.000000  102.400000  183.100000   66.900000   55.500000   
max    205.000000    3.000000  120.900000  208.100000   72.300000   59.800000   

        curbweight  enginesize   boreratio      stroke  compressionratio  \
count   205.000000  205.000000  205.000000  205.000000        205.000000   
mean   2555.565854  126.907317    3.329756    3.255415         10.142537   
std     520.680204   41.642693    0.270844    0.313597          3.972040   
min    1488.000000   61.000000    2.540000    2.070000          7.000000   
25%    2145.000000   97.000000    3.150000    3.110000          8.600000   
50%    2414.000000  120.000000    3.310000    3.290000          9.000000   
75%    2935.000000  141.000000    3.580000    3.410000          9.400000   
max    4066.000000  326.000000    3.940000    4.170000         23.000000   

       horsepower      peakrpm     citympg  highwaympg         price  
count  205.000000   205.000000  205.000000  205.000000    205.000000  
mean   104.117073  5125.121951   25.219512   30.751220  13276.710571  
std     39.544167   476.985643    6.542142    6.886443   7988.852332  
min     48.000000  4150.000000   13.000000   16.000000   5118.000000  
25%     70.000000  4800.000000   19.000000   25.000000   7788.000000  
50%     95.000000  5200.000000   24.000000   30.000000  10295.000000  
75%    116.000000  5500.000000   30.000000   34.000000  16503.000000  
max    288.000000  6600.000000   49.000000   54.000000  45400.000000  
car_ID              0
symboling           0
CarName             0
fueltype            0
aspiration          0
doornumber          0
carbody             0
drivewheel          0
enginelocation      0
wheelbase           0
carlength           0
carwidth            0
carheight           0
curbweight          0
enginetype          0
cylindernumber      0
enginesize          0
fuelsystem          0
boreratio           0
stroke              0
compressionratio    0
horsepower          0
peakrpm             0
```

```
citympg             0
highwaympg          0
price               0
dtype: int64
```

In [6]:

```python
print(data.columns)
```

```
Index(['car_ID', 'symboling', 'CarName', 'fueltype', 'aspiration',
       'doornumber', 'carbody', 'drivewheel', 'enginelocation', 'wheelbase',
       'carlength', 'carwidth', 'carheight', 'curbweight', 'enginetype',
       'cylindernumber', 'enginesize', 'fuelsystem', 'boreratio', 'stroke',
       'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
       'price'],
      dtype='object')
```

In [9]:

```python
X = data.drop('price', axis=1)
y = data['price']


```

In [10]:

```python
# One-hot encoding for categorical variables
X = pd.get_dummies(X, drop_first=True)

```

In [11]:

```python
#Model Implementation: Five regression models are instantiated and trained on the training set.
# Step 2: Model Implementation
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


```

In [12]:

```python
# Initialize models
models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree': DecisionTreeRegressor(),
    'Random Forest': RandomForestRegressor(),
    'Gradient Boosting': GradientBoostingRegressor(),
    'Support Vector Regression': SVR()
}


```

In [13]:

```python
# Train and evaluate models
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    results[name] = {
        'R-squared': r2_score(y_test, y_pred),
        'MSE': mean_squared_error(y_test, y_pred),
        'MAE': mean_absolute_error(y_test, y_pred)
    }


```

In [14]:

```python
# Step 3: Model Evaluation
#Model Evaluation: The performance of each model is evaluated using R-squared, MSE, and MAE, and results are summarized.
results_df = pd.DataFrame(results).T
print(results_df)


```

```
                           R-squared           MSE          MAE
Linear Regression          -1.239056  1.767601e+08  7280.667793
Decision Tree               0.887221  8.903252e+06  1881.918707
Random Forest               0.956571  3.428491e+06  1298.078000
Gradient Boosting           0.933773  5.228255e+06  1646.783382
Support Vector Regression  -0.101989  8.699543e+07  5707.167500
```

In [15]:

```python
# Identify the best performing model
best_model = results_df.loc[results_df['R-squared'].idxmax()]
print(f"Best Model: {best_model.name}, R-squared: {best_model['R-squared']}")


```

```
Best Model: Random Forest, R-squared: 0.9565706087055967
```

In [16]:

```
# Step 4: Feature Importance Analysis
Feature Importance Analysis: Feature importances from the Random Forest model are displayed to understand the impact of each

# Feature importance for tree-based models
importances = models['Random Forest'].feature_importances_
feature_importance_df = pd.DataFrame(importances, index=X.columns, columns=['Importance']).sort_values('Importance', ascendi
print(feature_importance_df)
```

```
                                  Importance
enginesize                          0.568230
curbweight                          0.248291
highwaympg                          0.053124
horsepower                          0.042431
car_ID                              0.020533
...                                      ...
CarName_toyota corolla 1600 (sw)    0.000000
CarName_volkswagen rabbit           0.000000
CarName_vokswagen rabbit            0.000000
CarName_nissan nv200                0.000000
CarName_nissan note                 0.000000

[190 rows x 1 columns]
```

In [17]:

```
# Step 5: Hyperparameter Tuning
Hyperparameter Tuning: The Random Forest model is tuned using Grid Search, and the performance of the tuned model is evaluat
# Example for Random Forest Regressor
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10]
}
grid_search = GridSearchCV(RandomForestRegressor(), param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

Out[17]:

```
▸ GridSearchCV
▸ estimator: RandomForestRegressor
    ▸ RandomForestRegressor
```

In [18]:

```
# Evaluate the tuned model
best_rf_model = grid_search.best_estimator_
y_pred_tuned = best_rf_model.predict(X_test)
print(f"Tuned Random Forest R-squared: {r2_score(y_test, y_pred_tuned)}")
```

```
Tuned Random Forest R-squared: 0.9570222346614141
```