Module 5: Machine Learning

A Chinese automobile company aspires to enter the US market by setting up their manufacturing unit there and producing cars locally to give competition to their US and European counterparts. They have contracted an automobile consulting company to understand the factors on which the pricing of cars depends. Specifically, they want to understand the factors affecting the pricing of cars in the American market, since those may be very different from the Chinese market. Essentially, the company wants to know:

- Which variables are significant in predicting the price of a car
- How well those variables describe the price of a car

Based on various market surveys, the consulting firm has gathered a large dataset of different types of cars across the American market.

Business Goal:

You are required to model the price of cars with the available independent variables. It will be used by the management to understand how exactly the prices vary with the independent variables. They can accordingly manipulate the design of the cars, the business strategy etc. to meet certain price levels. Further, the model will be a good way for the management to understand the pricing dynamics of a new market.

Dataset: https://drive.google.com/file/d/1FHmYNLs9v0Enc-UExEMpitOFGsWvB2dP/view?usp=drive_link

Key components to be fulfilled:

1. Loading and Preprocessing (5 marks)

- Load the dataset and perform necessary preprocessing steps.

```
1]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Load the dataset
url = 'https://drive.google.com/uc?export=download&id=1FHmYNLs9v0Enc-UExEMpitOFGsWvB2dP'
data = pd.read_csv(url)

# Display basic information
print(data.info())
print(data.describe())

# Handle missing values
data = data.dropna()  # Or use imputation strategies if needed

# Encode categorical variables
categorical_cols = data.select_dtypes(include=['object']).columns
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    data[col] = le.fit_transform(data[col])
    label_encoders[col] = le

# Feature scaling (only for numerical features)
numerical_cols = data.select_dtypes(include=['int64', 'float64']).columns
scaler = StandardScaler()
data[numerical_cols] = scaler.fit_transform(data[numerical_cols])

# Splitting the dataset
X = data.drop('price', axis=1)  # Replace 'price' with the target column's name
y = data['price']  # Replace 'price' with the target column's name
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
<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
```

```
          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
```

[ ]:

2. Model Implementation (10 marks)

- Implement the following five regression algorithms:

1) Linear Regression
2) Decision Tree Regressor
3) Random Forest Regressor
4) Gradient Boosting Regressor
5) Support Vector Regressor

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR

# Define models
models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree Regressor': DecisionTreeRegressor(random_state=42),
    'Random Forest Regressor': RandomForestRegressor(random_state=42),
    'Gradient Boosting Regressor': GradientBoostingRegressor(random_state=42),
    'Support Vector Regressor': SVR()
}

# Fit models and make predictions
predictions = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    predictions[name] = model.predict(X_test)
```

3. Model Evaluation (5 marks)

- Compare the performance of all the models based on R-squared, Mean Squared Error (MSE), and Mean Absolute Error (MAE).
- Identify the best performing model and justify why it is the best.

```
]: from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

   # Evaluate each model
   results = {}
   for name, preds in predictions.items():
       r2 = r2_score(y_test, preds)
       mse = mean_squared_error(y_test, preds)
       mae = mean_absolute_error(y_test, preds)
       results[name] = {'R²': r2, 'MSE': mse, 'MAE': mae}

   # Display results
   results_df = pd.DataFrame(results).T
   print(results_df)
```

```
                                 R²       MSE       MAE
Linear Regression          0.844116  0.193765  0.261917
Decision Tree Regressor    0.881493  0.147305  0.251717
Random Forest Regressor    0.955630  0.055153  0.166793
Gradient Boosting Regressor 0.933122  0.083129  0.200089
Support Vector Regressor   0.374270  0.777788  0.481898
```

```
]:
```

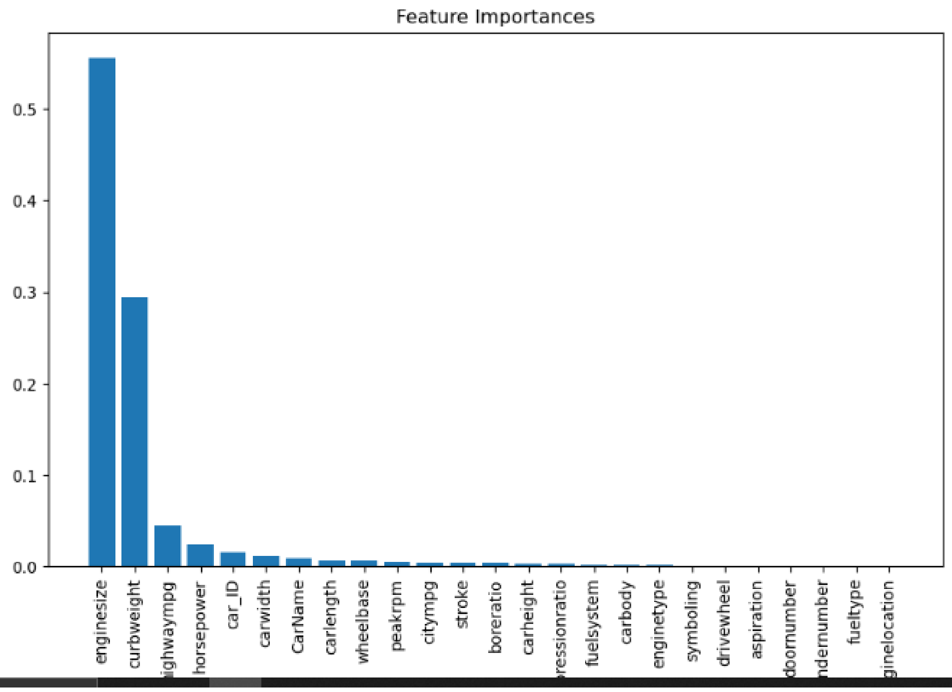
4. Feature Importance Analysis (2 marks)

- Identify the significant variables affecting car prices (feature selection)

```
# Feature importance for tree-based models
import matplotlib.pyplot as plt

best_tree_model = models['Random Forest Regressor']  # Replace with the best-performing tree model
importances = best_tree_model.feature_importances_
sorted_indices = np.argsort(importances)[::-1]

# Plot feature importances
plt.figure(figsize=(10, 6))
plt.bar(range(X_train.shape[1]), importances[sorted_indices], align="center")
plt.xticks(range(X_train.shape[1]), X.columns[sorted_indices], rotation=90)
plt.title("Feature Importances")
plt.show()
```



5. Hyperparameter Tuning (2 marks):

- Perform hyperparameter tuning and check whether the performance of the model has increased.

[*]:

```python
from sklearn.model_selection import GridSearchCV

# Example: Hyperparameter tuning for Random Forest Regressor
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(RandomForestRegressor(random_state=42), param_grid, cv=5, scoring='r2')
grid_search.fit(X_train, y_train)

# Display best parameters and performance
print("Best Parameters:", grid_search.best_params_)
best_model = grid_search.best_estimator_
best_preds = best_model.predict(X_test)
print("R² Score (Tuned):", r2_score(y_test, best_preds))
```

[ ]: