In [3]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
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 r2_score, mean_squared_error, mean_absolute_error
```

In [5]:
```
# Load the dataset
url = "https://drive.google.com/uc?export=download&id=1FHmYNLs9v0Enc-UExEMpitOFGsWv
df = pd.read_csv(url)
```

In [7]:
```
df
```

Out[7]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	carbody	drivew
0	1	3	alfa-romero giulia	gas	std	two	convertible	
1	2	3	alfa-romero stelvio	gas	std	two	convertible	
2	3	1	alfa-romero Quadrifoglio	gas	std	two	hatchback	
3	4	2	audi 100 ls	gas	std	four	sedan	
4	5	2	audi 100ls	gas	std	four	sedan	
...	...	...	...	...	...	...	...	
200	201	-1	volvo 145e (sw)	gas	std	four	sedan	
201	202	-1	volvo 144ea	gas	turbo	four	sedan	
202	203	-1	volvo 244dl	gas	std	four	sedan	
203	204	-1	volvo 246	diesel	turbo	four	sedan	
204	205	-1	volvo 264gl	gas	turbo	four	sedan	

205 rows × 26 columns

In [9]:
```
# Display basic information
df.info()
df.describe()
```

```
<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
```

Out[9]:

	car_ID	symboling	wheelbase	carlength	carwidth	carheight	curbweight
count	205.000000	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	2555.565854
std	59.322565	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204
min	1.000000	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000
25%	52.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000
50%	103.000000	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000
75%	154.000000	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000
max	205.000000	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000

In [11]:
```
# Handle missing values
df.dropna(inplace=True)
```

In [13]:
```
# Convert categorical variables to numerical
label_encoders = {}
for column in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le
```

In [25]:
```
# Splitting features and target
X = df.drop(columns=['price'])
y = df['price']
X
y
```

Out[25]:
```
0      13495.0
1      16500.0
2      16500.0
3      13950.0
4      17450.0
        ...
200    16845.0
201    19045.0
202    21485.0
203    22470.0
204    22625.0
Name: price, Length: 205, dtype: float64
```

In [17]:
```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [19]:
```
# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

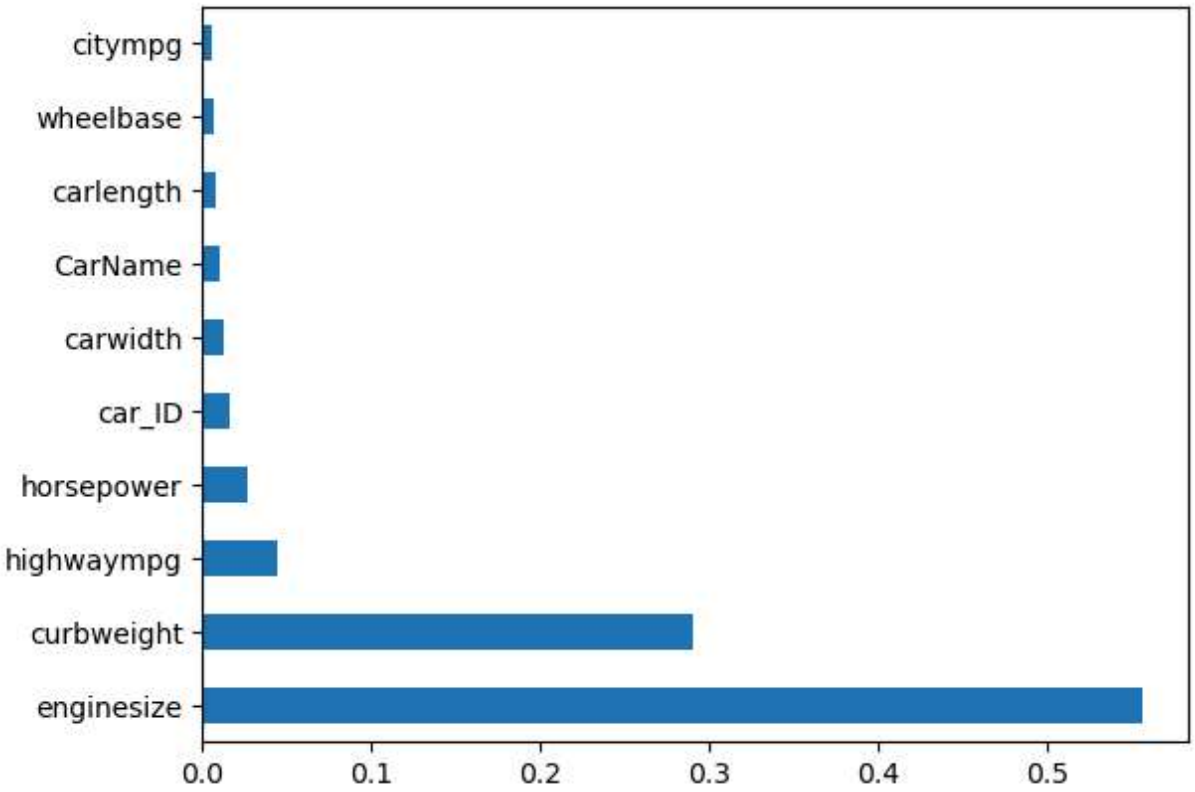
In [27]:
```
# Model initialization
models = {
    'Linear Regression': LinearRegression(),
    'Decision Tree': DecisionTreeRegressor(random_state=42),
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42),
    'Gradient Boosting': GradientBoostingRegressor(n_estimators=100, random_state=4
    'Support Vector Regressor': SVR()
}
```

In [31]:
```
# Model evaluation
def evaluate_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return {
        'R-squared': r2_score(y_test, y_pred),
        'MSE': mean_squared_error(y_test, y_pred),
        'MAE': mean_absolute_error(y_test, y_pred)
    }
results = {name: evaluate_model(model, X_train, X_test, y_train, y_test) for name,
```

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

```
                           R-squared           MSE          MAE
Linear Regression           0.844116  1.230612e+07  2087.306212
Decision Tree               0.879253  9.532216e+06  2090.699195
Random Forest               0.957168  3.381318e+06  1303.187512
Gradient Boosting           0.933037  5.286350e+06  1600.962646
Support Vector Regressor   -0.100146  8.684995e+07  5696.573042
```

In [33]:
```
# Feature Importance (Random Forest)
feature_importances = pd.Series(models['Random Forest'].feature_importances_, index
feature_importances.nlargest(10).plot(kind='barh')
plt.show()
```



In [ ]:
```
```

In [35]:
```
# Hyperparameter Tuning for Random Forest
param_grid = {'n_estimators': [50, 100, 200], 'max_depth': [None, 10, 20], 'min_sam
gs = GridSearchCV(RandomForestRegressor(random_state=42), param_grid, cv=3, scoring
gs.fit(X_train, y_train)
```

Out[35]:

▸ GridSearchCV

▸ estimator: RandomForestRegressor

▸ RandomForestRegressor

In [36]:
```
print("Best parameters:", gs.best_params_)
best_model = gs.best_estimator_
y_pred_tuned = best_model.predict(X_test)
print("Tuned Model R-squared:", r2_score(y_test, y_pred_tuned))
```

```
Best parameters: {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 200}
Tuned Model R-squared: 0.9562656644447429
```