In [1]:
```
# Task: Train a KNN model to predict car availability based on selected features.

# Steps:
# - Use 'Car_Name', 'Number_of_Doors', 'No_of_Cylinder', 'Car_Mileage', 'Car_Age' as features
# - Use 'Available' as the target
# - Handle missing values
# - Encode categorical data
# - Scale features
# - Train/test split, train model, and evaluate with confusion matrix and accuracy score

# Import required libraries
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
```

In [2]:
```
# Load dataset

df = pd.read_csv('Car Data.csv')
```

In [5]:
```
# Select relevant columns

df = df[['Car_Name', 'Number_of_Doors', 'No_of_Cylinder', 'Car_Mileage', 'Car_Age', 'Available']]
```

In [7]:
```
# Clean column names and strip whitespaces

df.columns = df.columns.str.strip()
df['Available'] = df['Available'].str.strip()
```

In [9]:
```
# Fill missing numeric values with mean

imputer = SimpleImputer(strategy='mean')
df[['Number_of_Doors', 'No_of_Cylinder', 'Car_Mileage', 'Car_Age']] = imputer.fit_transform(
    df[['Number_of_Doors', 'No_of_Cylinder', 'Car_Mileage', 'Car_Age']])
```

In [11]:
```
#  Encode categorical variables ('Car_Name' and 'Available') to numerical values

label_encoders = {}
for col in ['Car_Name', 'Available']:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
```

In [13]:
```
# Define input features (X) and target label (y)

X = df[['Car_Name', 'Number_of_Doors', 'No_of_Cylinder', 'Car_Mileage', 'Car_Age']]
y = df['Available']
```

In [15]:
```
# Scale the features for better KNN performance

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [17]:
```
# Split data into training and test sets (80% train, 20% test)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

In [21]:
```
# Train KNN model

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

Out[21]:
```
▾ KNeighborsClassifier

KNeighborsClassifier()
```

In [23]:
```
# Predict and evaluate

y_pred = knn.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
```

In [25]:
```
print("Confusion Matrix:\n", conf_matrix)
print("Accuracy Score:", accuracy)
```

```
Confusion Matrix:
 [[ 6 11]
 [ 9 15]]
Accuracy Score: 0.5121951219512195
```

In [27]:
```
# Predict availability for the test case (Mazda, diesel, std, 4 doors, sedan, fwd, 4 cylinders, 134000 mileage, 22
# Manually encode the new input based on existing encoders and scale

input_data = pd.DataFrame([{
    'Car_Name': label_encoders['Car_Name'].transform(['mazda'])[0],
    'Number_of_Doors': 4,
    'No_of_Cylinder': 4,
    'Car_Mileage': 134000,
    'Car_Age': 22
}])
input_scaled = scaler.transform(input_data)
prediction = knn.predict(input_scaled)
```

In [29]:
```
# Print the prediction

availability = label_encoders['Available'].inverse_transform(prediction)[0]
print("Predicted Availability for test car:", availability)
```

```
Predicted Availability for test car: no
```

In [31]:
```
#  Summary: KNN Model for Predicting Car Availability
#
#  - This model uses K-Nearest Neighbors to predict car availability.
#  - It uses 5 input features: Car Name, Number of Doors, Cylinders, Mileage, and Age.
#  - All missing values are filled using the mean.
#  - Categorical data is encoded numerically and scaled for KNN.
#  - For a test case (Mazda, 4 doors, 4 cylinders, 134000 km, 22 years old),
#       the model predicts whether the car is available or not.
#  - Evaluation includes a confusion matrix and accuracy score.

#  Why This Matters in Finance:
#  - Helps predict inventory or availability in car dealerships.
```

```
#  - Useful in managing fleet, pricing insurance, or loan risk.
#  - Can be extended to other categorical predictions in financial analytics
#
```