In [3]:
```
# Step 1: Import necessary libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# For preprocessing and model building
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score
```

In [5]:
```
# Step 2: Load the dataset

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

In [7]:
```
# Step 3: Select relevant features (X = independent variables) and target (Y = dependent variable)

X = dataset[['Car_Name', 'Number_of_Doors', 'No_of_Cylinder', 'Car_Mileage', 'Car_Age']].values
Y = dataset[['Available']].values.ravel()  # Flatten Y to 1D array
```

In [9]:
```
# Step 4: Handle missing numeric data (Car_Mileage and Car_Age)

imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(X[:, 3:5])  # Only apply to numeric columns
X[:, 3:5] = imputer.transform(X[:, 3:5])
```

In [11]:
```
# Sample Data

print (X[0:1])
print (Y[0:1]) # 'Yes' means car is avaiable.
```

```
[['alfa-romero' 2 4 130000.0 9.0]]
['yes']
```

In [13]:
```
# Step 5: Encode the 'Car_Name' column (categorical feature)

ct = ColumnTransformer(transformers=[
    ('encoder', OneHotEncoder(sparse=False, handle_unknown='ignore'), [0])  # Encode column 0
], remainder='passthrough')  # Keep other columns unchanged
X = ct.fit_transform(X)
```

```
/Users/Niall/anaconda3/lib/python3.10/site-packages/sklearn/preprocessing/_encoders.py:828: FutureWarning: `sparse`
was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leav
e `sparse` to its default value.
  warnings.warn(
```

In [15]:
```
# Sample, coloumn 'Car_name' is encoded now
print (X[0:1])
```

```
[[1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
  0.0 0.0 0.0 0.0 2 4 130000.0 9.0]]
```

In [17]:
```
# Step 6: Encode the target variable 'Available' (Yes/No → 1/0)

le = LabelEncoder()
Y = le.fit_transform(Y)  #Converts 'yes' to 1 and 'no' to 0
```

In [19]:
```
# Y is encoded now, sample data for 25 rows
print (Y[0:25])
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1]
```

In [21]:
```
# Step 7: Check for missing data in target

print('Missing data in target column "Available":', dataset['Available'].isna().sum())
```

```
Missing data in target column "Available": 0
```

In [23]:
```
# Step 8: Split data into training and testing sets

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, random_state=0)
```

In [25]:
```
# Step 9: Train logistic regression model
```

```
reg = LogisticRegression(random_state=0)
reg.fit(X_train, Y_train)
```

Out[25]:

```
▾ LogisticRegression
LogisticRegression(random_state=0)
```

In [27]:

```
# Step 10: Predict test set results

Y_pred = reg.predict(X_test)
```

In [29]:

```
# Step 11: Display predictions and true values

print("Predicted values:", Y_pred)
print("Actual values:   ", Y_test)
```

```
Predicted values: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1]
Actual values:    [1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 1 0 1 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1 1 0 0
 1 1 1 1]
```

In [31]:

```
# Step 12: Predict availability of a specific car

# Prepare input as a DataFrame (with same structure)
car_input = pd.DataFrame([['mazda', 4, 4, 134000.0, 22.0]],
                         columns=['Car_Name', 'Number_of_Doors', 'No_of_Cylinder', 'Car_Mileage', 'Car_Age'])

# Impute missing values (if any) — only for numeric columns
car_input.iloc[:, 3:5] = imputer.transform(car_input.iloc[:, 3:5])

# Transform the input using the already fitted column transformer
car_input_transformed = ct.transform(car_input)

# Make prediction
prediction = reg.predict(car_input_transformed)
predicted_label = 'YES' if prediction[0] == 1 else 'NO'

print("Availability prediction for Mazda car:", predicted_label)
```

```
Availability prediction for Mazda car: YES
```

```
/Users/Niall/anaconda3/lib/python3.10/site-packages/sklearn/base.py:413: UserWarning: X has feature names, but Simpl
eImputer was fitted without feature names
  warnings.warn(
/Users/Niall/anaconda3/lib/python3.10/site-packages/sklearn/base.py:413: UserWarning: X has feature names, but OneHo
tEncoder was fitted without feature names
  warnings.warn(
```

In [33]:

```python
# Step 12: Evaluate the model using a Confusion Matrix and Accuracy Score

from sklearn.metrics import confusion_matrix, accuracy_score

# Generate the confusion matrix, # Rows: actual values, Columns: predicted values
# For binary classification: [[TN, FP], [FN, TP]]
cm = confusion_matrix(Y_test, Y_pred)

# Display the confusion matrix
print("Confusion Matrix:\n", cm)

# Interpretation:
# cm[0][0] = True Negatives (car not available and predicted not available)
# cm[0][1] = False Positives (car not available but predicted available)
# cm[1][0] = False Negatives (car available but predicted not available)
# cm[1][1] = True Positives (car available and predicted available)
print() # bank line space

# Calculate the overall accuracy of the model, # Accuracy = (TP + TN) / total predictions
acc = accuracy_score(Y_test, Y_pred)

# Print the accuracy score as a percentage
print("Accuracy Percentage: {:.2f}%".format(acc * 100))
```

```
Confusion Matrix:
 [[ 0 14]
 [ 0 27]]

Accuracy Percentage: 65.85%
```

In [35]:

```python
# Model Summary and Business Use

# This model predicts whether a car is available based on details like its name, doors, mileage, and age.
#
```

```
# In banking and finance, we can use similar models to:
# - Predict if a customer will get approved for a loan.
# - Estimate the chance someone might miss a payment.
# - Identify who might be interested in new products or services.
#
# Using these predictions helps banks make smarter decisions, offer better service,
# and reduce risks — all leading to happier customers and a healthier business outcomes
#
```

In [ ]:

```
```

In [ ]:

```
```