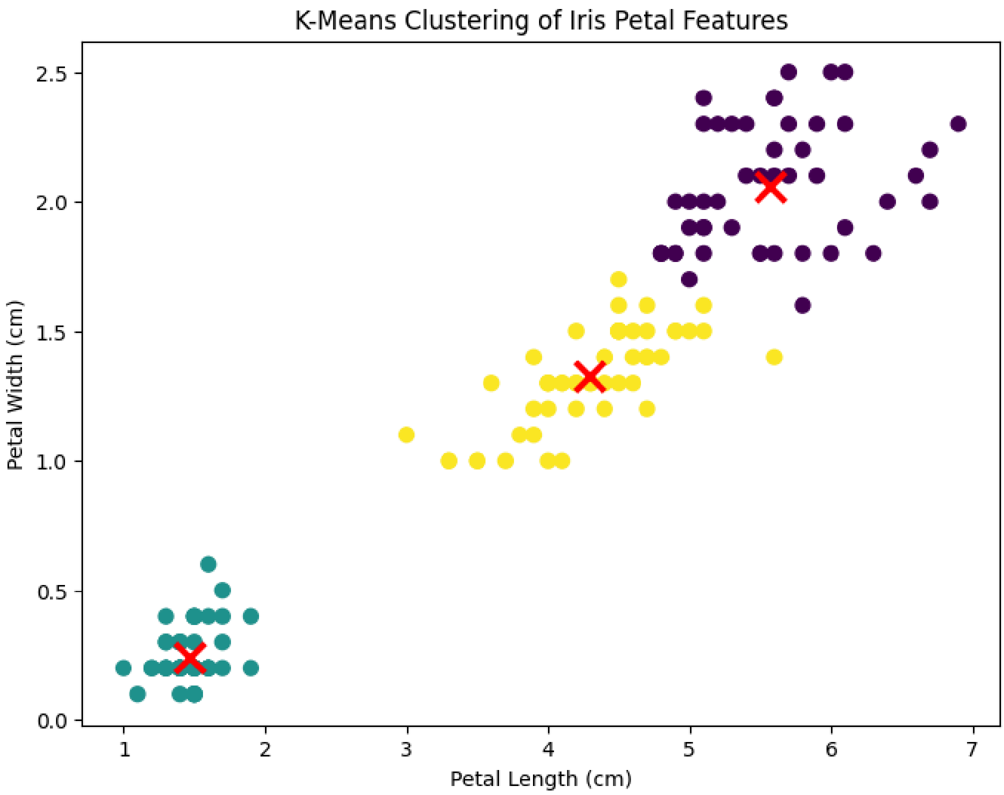
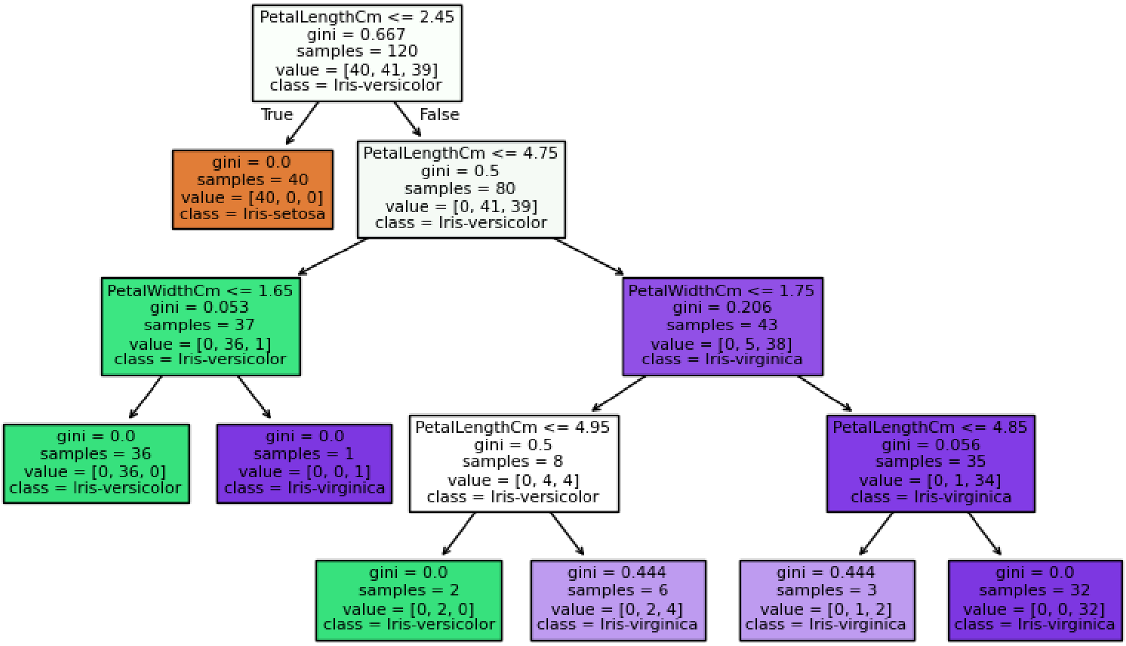
In [1]:
```python
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load and preprocess data
df = pd.read_csv('Iris.csv')
df.columns = df.columns.str.strip().str.lower()
X = df[['petallengthcm', 'petalwidthcm']]

# Scale and cluster
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
kmeans = KMeans(n_clusters=3, random_state=42).fit(X_scaled)

# Plot clusters and centroids
plt.figure(figsize=(8, 6))
plt.scatter(X['petallengthcm'], X['petalwidthcm'], c=kmeans.labels_, cmap='viridis', s=50)
centers = scaler.inverse_transform(kmeans.cluster_centers_)
plt.scatter(centers[:, 0], centers[:, 1], c='red', marker='x', s=200, linewidths=3)
plt.xlabel('Petal Length (cm)')
plt.ylabel('Petal Width (cm)')
plt.title('K-Means Clustering of Iris Petal Features')
plt.show()
```



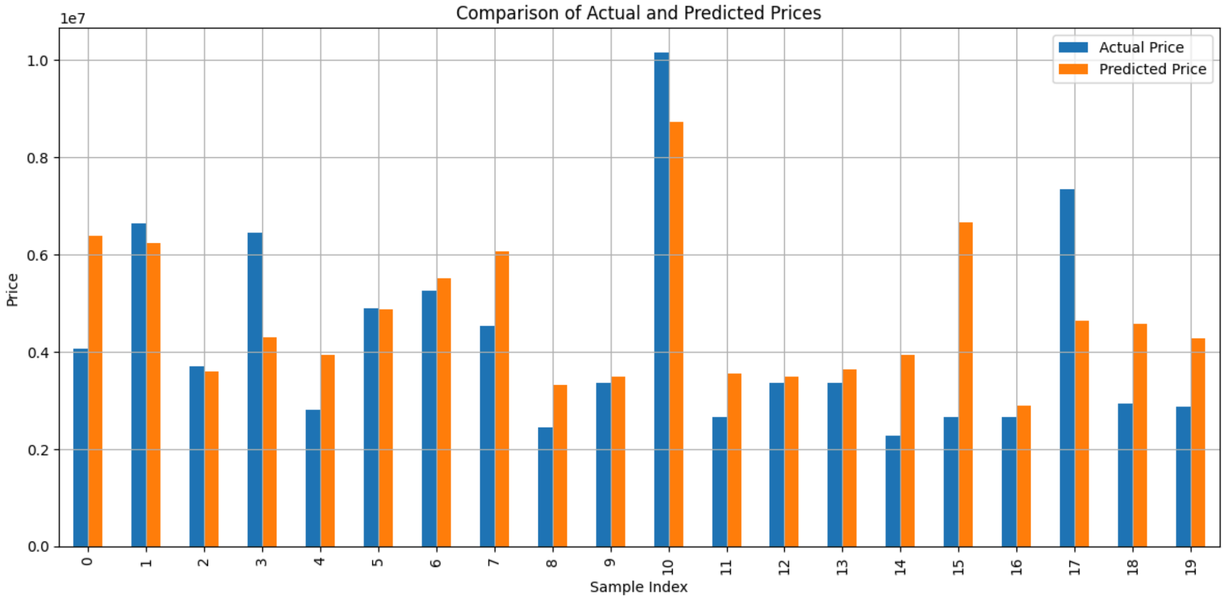
In [2]:
```python
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split, GridSearchCV
import matplotlib.pyplot as plt

df = pd.read_csv('Iris.csv')
X = df[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
y = df['Species']

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

grid = GridSearchCV(DecisionTreeClassifier(random_state=42), {
    'max_depth': [2, 3, 4, None],
    'min_samples_split': [2, 3, 4]
}, cv=5).fit(X_train, y_train)

plt.figure(figsize=(10, 6))
plot_tree(grid.best_estimator_, feature_names=X.columns, class_names=grid.best_estimator_.classes_, filled=True)
plt.show()
```



In [5]:

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv('housing.csv')  # Replace with your dataset path

# Select features and target
X = df[['area', 'bedrooms', 'bathrooms']]  # Replace with actual feature column names
y = df['price']  # Replace with actual target column name

# Split 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)

# Train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Display evaluation metrics
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))

# Compare actual vs predicted prices
comparison = pd.DataFrame({'Actual Price': y_test.values, 'Predicted Price': y_pred})
print(comparison.head())

# Visualize actual vs predicted
comparison.head(20).plot(kind='bar', figsize=(12, 6))
plt.title("Comparison of Actual and Predicted Prices")
plt.xlabel("Sample Index")
plt.ylabel("Price")
plt.grid(True)
plt.tight_layout()
plt.show()
```

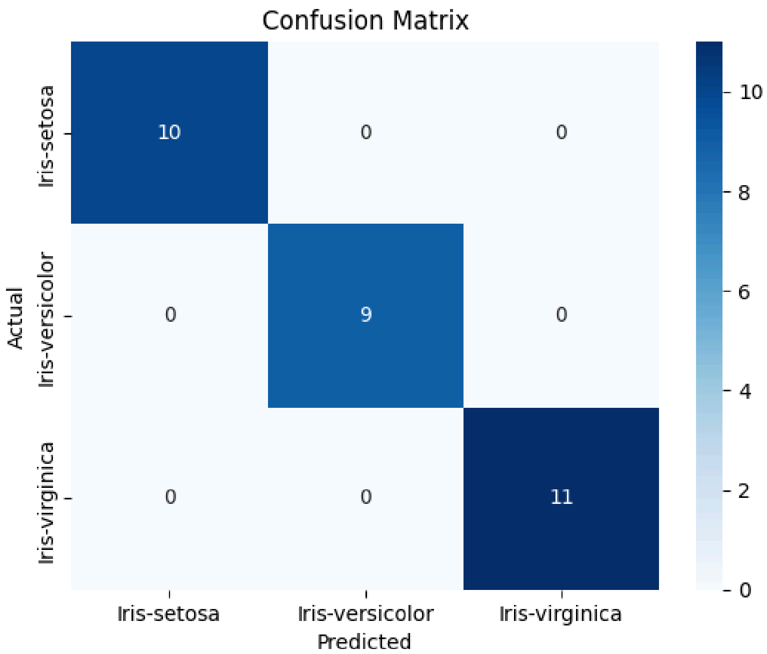
```
Mean Squared Error: 2750040479309.052
R² Score: 0.45592991188724463
   Actual Price  Predicted Price
0       4060000     6.383168e+06
1       6650000     6.230250e+06
2       3710000     3.597885e+06
3       6440000     4.289731e+06
4       2800000     3.930446e+06
```



In [8]:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv('Iris.csv')

# Select features and target
X = df[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
y = df['Species']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the Logistic Regression model
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluation
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix visualization
conf_matrix = confusion_matrix(y_test, y_pred, labels=model.classes_)
sns.heatmap(conf_matrix, annot=True, fmt='d', xticklabels=model.classes_, yticklabels=model.classes_, cmap='Blues'
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00        11

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30
```



In [12]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt

# Load and preprocess data
df = pd.read_csv('Iris.csv')
df.columns = df.columns.str.strip().str.lower()
X = df[['petallengthcm', 'petalwidthcm']]
y = df['species']

# Split and scale data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Fit KNN model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)

# Predict and evaluate
y_pred = knn.predict(X_test_scaled)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy: 1.0
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       1.00      1.00      1.00        11

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30
```

In [13]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler # Essential for KNN

# 1. Load the Iris dataset
df = pd.read_csv('Iris.csv')

# 2. Separate features (X) and target (y)
# 'Id' column is dropped as it's not a feature, and 'Species' is the target.
X = df.drop(['Id', 'Species'], axis=1)
y = df['Species']

# 3. Split data into training and testing sets
```

```python
# 30% of data for testing, 70% for training. stratify=y ensures class proportions are maintained.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# 4. Feature Scaling (Crucial for KNN!)
# KNN is distance-based, so scaling prevents features with larger values from dominating.
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train) # Fit and transform on training data
X_test_scaled = scaler.transform(X_test)     # Transform test data using the same scaler

# 5. Initialize and Train the KNN Classifier
# n_neighbors (k) is set to 5 as a common starting point.
k = 5
knn_classifier = KNeighborsClassifier(n_neighbors=k)
knn_classifier.fit(X_train_scaled, y_train) # Train the model on scaled training data

# 6. Make Predictions
y_pred = knn_classifier.predict(X_test_scaled) # Make predictions on scaled test data

# Outputting a snippet of results
print("KNN Model Training and Prediction Completed.")
print(f"Number of Neighbors (k) used: {k}")
print("\nSample of Actual Values (first 5 from test set):")
print(y_test.head())
print("\nSample of Predicted Values (first 5 from test set):")
print(y_pred[:5])
```

```
KNN Model Training and Prediction Completed.
Number of Neighbors (k) used: 5

Sample of Actual Values (first 5 from test set):
107     Iris-virginica
63     Iris-versicolor
133     Iris-virginica
56     Iris-versicolor
127     Iris-virginica
Name: Species, dtype: object

Sample of Predicted Values (first 5 from test set):
['Iris-virginica' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-virginica']
```

In [14]:
```python
import statistics

# Sample data
data = [12, 15, 21, 15, 19, 15, 18, 21, 17]

# Central Tendency Measures
mean = statistics.mean(data)
median = statistics.median(data)
mode = statistics.mode(data)

# Measures of Dispersion
variance = statistics.variance(data)
std_deviation = statistics.stdev(data)

# Display results
print("Central Tendency Measures:")
print(f"Mean: {mean}")
print(f"Median: {median}")
print(f"Mode: {mode}")

print("\nMeasures of Dispersion:")
print(f"Variance: {variance}")
print(f"Standard Deviation: {std_deviation}")
```

```
Central Tendency Measures:
Mean: 17
Median: 17
Mode: 15

Measures of Dispersion:
Variance: 9.25
Standard Deviation: 3.0413812651491097
```

In [16]:
```python
import pandas as pd

pd.read_csv("iris.csv")           # Load data from a CSV file
df.head()                         # View the first 5 rows of a DataFrame
df.describe()                     # Summary statistics
df.dropna()                       # Remove missing values
```

Out[16]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

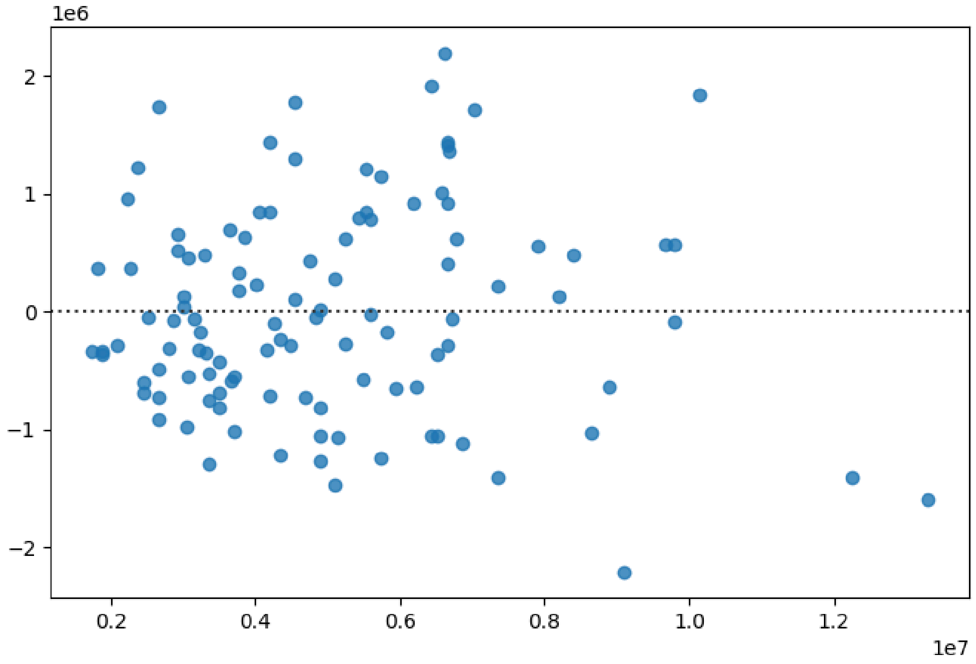
150 rows × 6 columns

In [20]:

```python
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
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_absolute_error

# Step 1: Load and Encode Data
df = pd.read_csv("Housing.csv")
df_encoded = pd.get_dummies(df, drop_first=True)

# Step 2: Split Features and Target
X = df_encoded.drop('price', axis=1)
y = df_encoded['price']

# Step 3: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 4: Model Training
model = LinearRegression()
model.fit(X_train, y_train)

# Step 5: Predictions
y_pred = model.predict(X_test)

# Step 6: Evaluation
print("R² Score:", r2_score(y_test, y_pred))
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred))

# Step 7: Visualization - Residual Plot
plt.figure(figsize=(8,5))
sns.residplot(x=y_test, y=y_pred, lowess=True, line_kws={'color': 'red'})
plt.xlabel("Actual Price")
plt.ylabel("Residuals")
plt.title("Residuals vs Actual Price")
plt.tight_layout()
plt.show()

# Step 8: Predicting New House Price
# Format: [area, bedrooms, bathrooms, stories, mainroad_yes, guestroom_yes, basement_yes, hotwaterheating_yes,
# airconditioning_yes, parking, prefarea_yes, furnishingstatus_semi-furnished, furnishingstatus_unfurnished]
new_data = np.array([[7500, 3, 2, 2, 1, 0, 1, 0, 1, 2, 1, 1, 0]]).reshape(1, -1)
new_price = model.predict(new_data)[0]
print(f"Predicted price for the new house: ₹{new_price:,.0f}")
```

```
R² Score: 0.6529242642153184
Mean Absolute Error: 970043.4039201637
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
Cell In[20], line 34
     32 # Step 7: Visualization - Residual Plot
     33 plt.figure(figsize=(8,5))
---> 34 sns.residplot(x=y_test, y=y_pred, lowess=True, line_kws={'color': 'red'})
     35 plt.xlabel("Actual Price")
     36 plt.ylabel("Residuals")

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\regression.py:939, in residplot(data, x, y, x_partial, y_partial, lowess, order, robust, dropna, label, color, scatter_kws, line_kws, ax)
    937 scatter_kws = {} if scatter_kws is None else scatter_kws.copy()
    938 line_kws = {} if line_kws is None else line_kws.copy()
--> 939 plotter.plot(ax, scatter_kws, line_kws)
    940 return ax

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\regression.py:384, in _RegressionPlotter.plot(self, ax, scatter_kws, line_kws)
    381     self.scatterplot(ax, scatter_kws)
    383 if self.fit_reg:
--> 384     self.lineplot(ax, line_kws)
    386 # Label the axes
    387 if hasattr(self.x, "name"):

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\regression.py:429, in _RegressionPlotter.lineplot(self, ax, kws)
    427 """Draw the model."""
    428 # Fit the regression model
--> 429 grid, yhat, err_bands = self.fit_regression(ax)
    430 edges = grid[0], grid[-1]
    432 # Get set default aesthetics

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\regression.py:198, in _RegressionPlotter.fit_regression(self, ax, x_range, grid)
    196 def fit_regression(self, ax=None, x_range=None, grid=None):
    197     """Fit the regression model."""
--> 198     self._check_statsmodels()
    200     # Create the grid for the regression
    201     if grid is None:

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\seaborn\regression.py:194, in _RegressionPlotter._check_statsmodels(self)
    192 for option in options:
    193     if getattr(self, option) and not _has_statsmodels:
--> 194         raise RuntimeError(err.format(option))

RuntimeError: `lowess=True` requires statsmodels, an optional dependency, to be installed.
```



In [21]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load dataset
```

```python
data = pd.read_csv('Housing.csv')  # Replace with your actual file path

# Select features and target
X = data[['area', 'bedrooms', 'bathrooms']]  # Example features
y = data['price']  # Target variable

# Split 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)

# Train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
print("R² Score:", r2_score(y_test, y_pred))

# Model coefficients
print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)
```

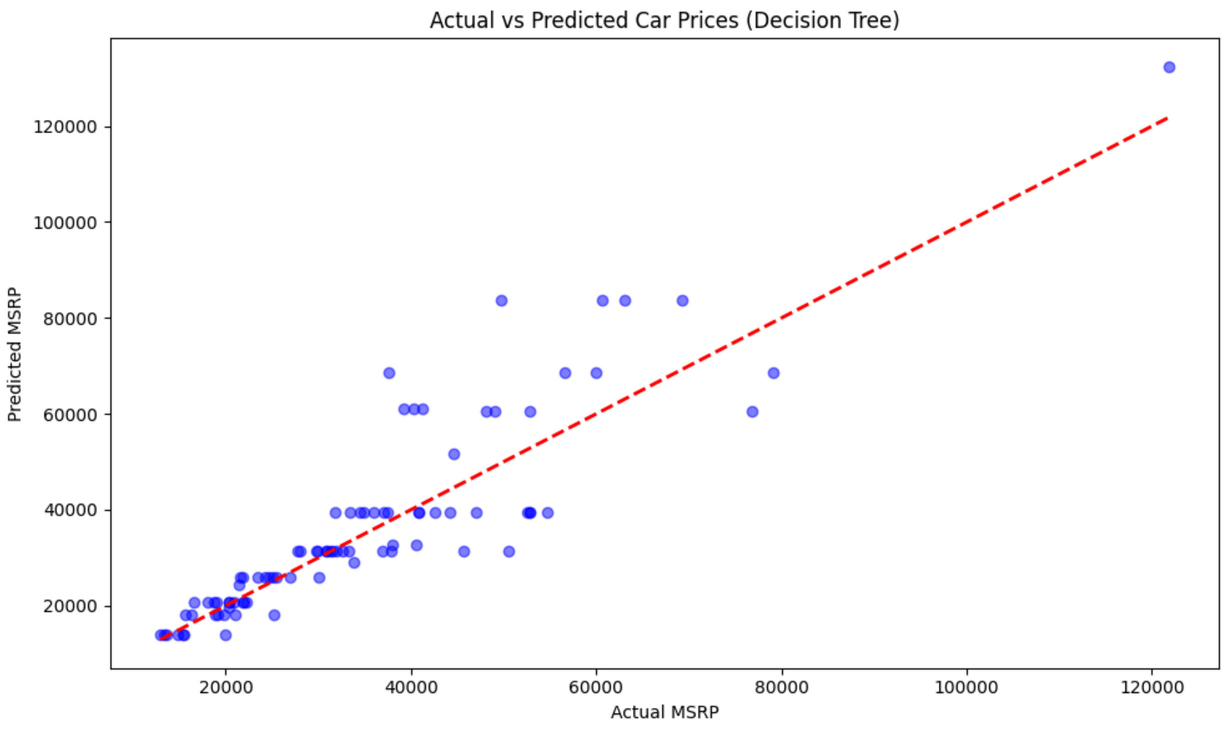
```
Mean Squared Error: 2750040479309.052
R² Score: 0.45592991188724463
Intercept: 59485.379208717495
Coefficients: [3.45466570e+02 3.60197650e+05 1.42231966e+06]
```

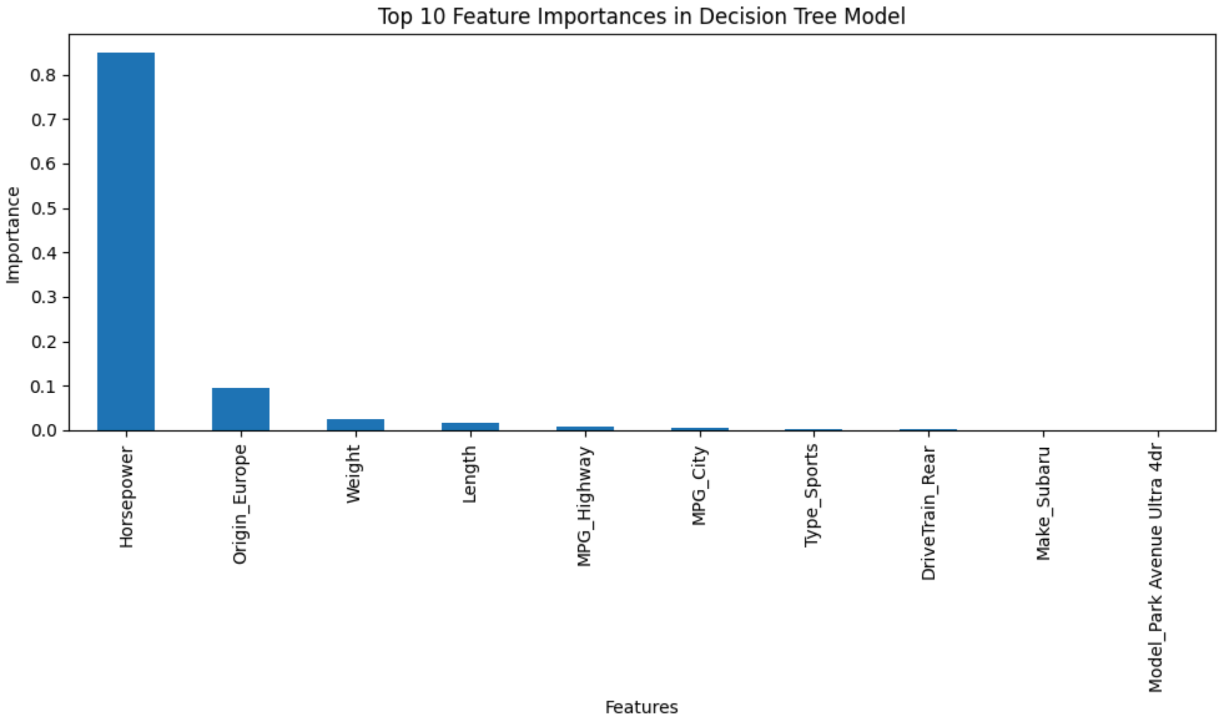
In [26]:

```python
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import re

# Function to convert MSRP to numeric
def convert_msrp(price):
    if isinstance(price, str):
        return float(re.sub(r'[^\d.]', '', price))
    return float(price)

# Load the dataset
data = pd.read_csv('cars_data.csv')

# Convert MSRP to numeric
data['MSRP'] = data['MSRP'].apply(convert_msrp)

# Define feature columns and target
numerical_features = ['EngineSize', 'Cylinders', 'Horsepower', 'MPG_City', 'MPG_Highway', 'Weight', 'Wheelbase', '
categorical_features = ['Make', 'Model', 'Type', 'Origin', 'DriveTrain']
target_column = 'MSRP'

# Handle missing values
data = data.dropna(subset=numerical_features + categorical_features + [target_column])

# Encode categorical variables
data = pd.get_dummies(data, columns=categorical_features, drop_first=True)

# Prepare features (X) and target (y)
feature_columns = numerical_features + [col for col in data.columns if col.startswith(tuple(categorical_features))
X = data[feature_columns]
y = data[target_column]

# Scale numerical features
scaler = StandardScaler()
X_scaled = X.copy()
X_scaled[numerical_features] = scaler.fit_transform(X[numerical_features])

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Initialize the Decision Tree model
dt = DecisionTreeRegressor(random_state=42)

# Define parameter grid for tuning
param_grid = {
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

```
# Perform GridSearchCV
grid_search = GridSearchCV(dt, param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Get the best model
best_dt = grid_search.best_estimator_
print(f'Best Parameters: {grid_search.best_params_}')

# Make predictions
y_pred = best_dt.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse:.2f}')
print(f'R² Score: {r2:.2f}')

# Feature importance
feature_importance = pd.Series(best_dt.feature_importances_, index=feature_columns).sort_values(ascending=False)
print('Top 5 Feature Importances:')
print(feature_importance.head())

# Visualize actual vs predicted prices
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.xlabel('Actual MSRP')
plt.ylabel('Predicted MSRP')
plt.title('Actual vs Predicted Car Prices (Decision Tree)')
plt.tight_layout()
plt.show()

# Visualize feature importance
plt.figure(figsize=(10, 6))
feature_importance.head(10).plot(kind='bar')
plt.title('Top 10 Feature Importances in Decision Tree Model')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.tight_layout()
plt.show()
```

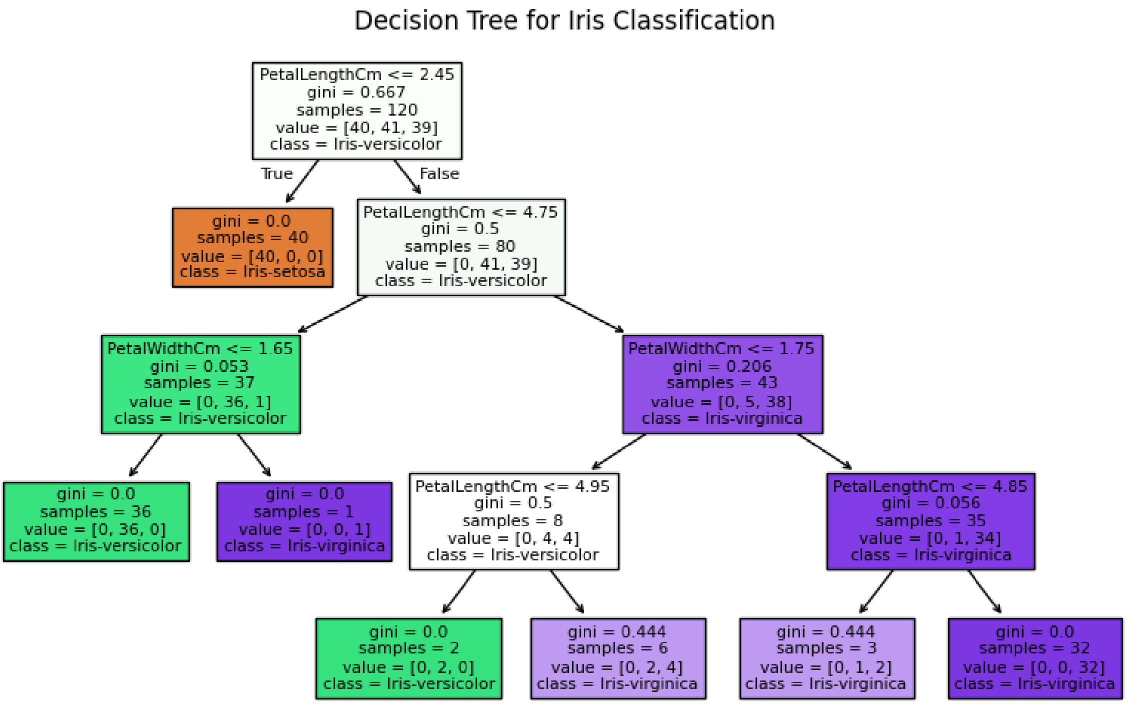
```
Best Parameters: {'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split': 10}
Mean Squared Error: 89434646.44
R² Score: 0.71
Top 5 Feature Importances:
Horsepower       0.847816
Origin_Europe    0.093511
Weight           0.025455
Length           0.015857
MPG_Highway      0.009184
dtype: float64
```





In [27]:
```python
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('Iris.csv')
X = df[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
y = df['Species']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Perform GridSearchCV
grid = GridSearchCV(DecisionTreeClassifier(random_state=42), {
    'max_depth': [2, 3, 4, None],
    'min_samples_split': [2, 3, 4]
}, cv=5).fit(X_train, y_train)

# Get the best model
best_model = grid.best_estimator_
print(f'Best Parameters: {grid.best_params_}')

# Predict class probabilities for MSE and R² calculation
y_pred_proba = best_model.predict_proba(X_test)

# Convert true labels to one-hot encoding for probability-based MSE and R²
y_test_one_hot = pd.get_dummies(y_test).values
mse = mean_squared_error(y_test_one_hot, y_pred_proba)
r2 = r2_score(y_test_one_hot, y_pred_proba)

# Print evaluation metrics
print(f'Mean Squared Error (based on probabilities): {mse:.4f}')
print(f'R² Score (based on probabilities): {r2:.4f}')

# Visualize the decision tree
plt.figure(figsize=(10, 6))
plot_tree(best_model, feature_names=X.columns, class_names=best_model.classes_, filled=True)
plt.title('Decision Tree for Iris Classification')
plt.show()
```

```
Best Parameters: {'max_depth': 4, 'min_samples_split': 2}
Mean Squared Error (based on probabilities): 0.0000
R² Score (based on probabilities): 1.0000
```

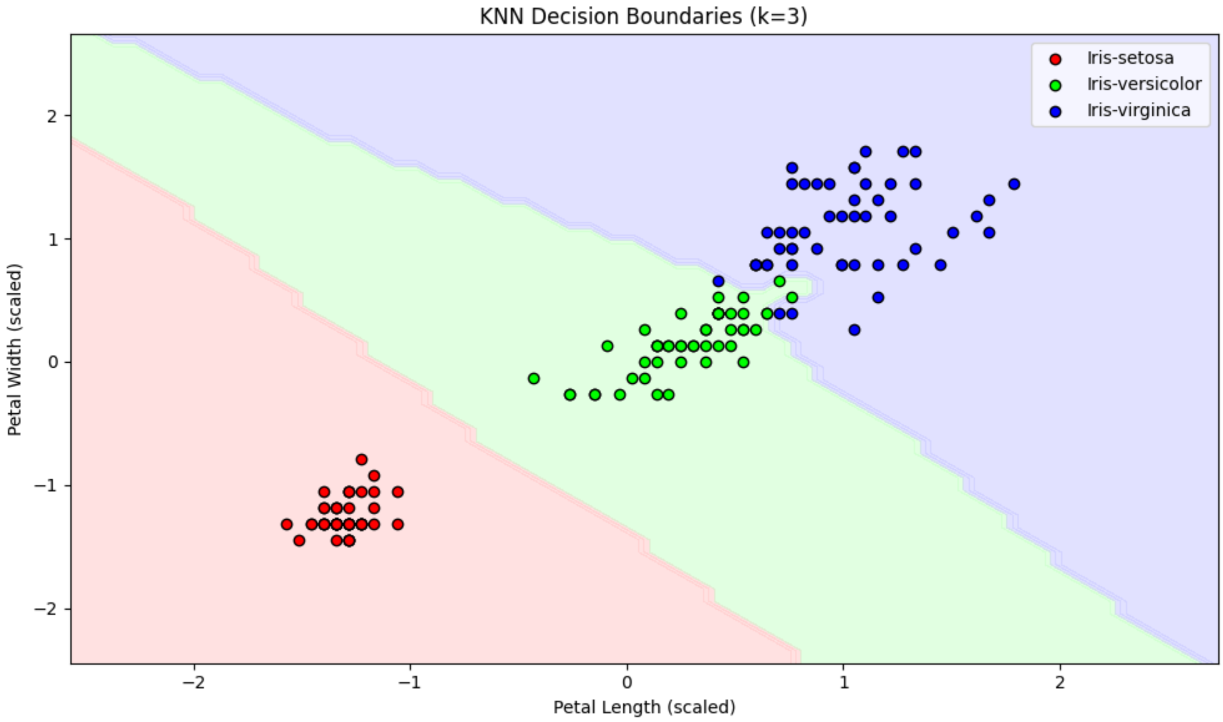


In [28]:

```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# Load the dataset
df = pd.read_csv('Iris.csv')
X = df[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
y = df['Species']

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Initialize KNN classifier
knn = KNeighborsClassifier()

# Define parameter grid for tuning
param_grid = {'n_neighbors': [3, 5, 7, 9, 11]}

# Perform GridSearchCV
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Get the best model
best_knn = grid_search.best_estimator_
print(f'Best Parameters: {grid_search.best_params_}')

# Make predictions
y_pred = best_knn.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.4f}')

# Calculate MSE and R² based on predicted probabilities
y_pred_proba = best_knn.predict_proba(X_test)
y_test_one_hot = pd.get_dummies(y_test).values
mse = mean_squared_error(y_test_one_hot, y_pred_proba)
r2 = r2_score(y_test_one_hot, y_pred_proba)
print(f'Mean Squared Error (based on probabilities): {mse:.4f}')
print(f'R² Score (based on probabilities): {r2:.4f}')
```

```python
# Visualize decision boundaries using two features (PetalLengthCm and PetalWidthCm)
X_subset = X_scaled[:, [2, 3]]  # PetalLengthCm and PetalWidthCm
X_train_subset = X_train[:, [2, 3]]
X_test_subset = X_test[:, [2, 3]]

# Train KNN on the subset for visualization
knn_subset = KNeighborsClassifier(n_neighbors=grid_search.best_params_['n_neighbors'])
knn_subset.fit(X_train_subset, y_train)

# Create mesh grid for decision boundary
x_min, x_max = X_subset[:, 0].min() - 1, X_subset[:, 0].max() + 1
y_min, y_max = X_subset[:, 1].min() - 1, X_subset[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 0.1))
Z = knn_subset.predict(np.c_[xx.ravel(), yy.ravel()])
Z = pd.Categorical(Z, categories=best_knn.classes_).codes
Z = Z.reshape(xx.shape)

# Plot decision boundaries
plt.figure(figsize=(10, 6))
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ['#FF0000', '#00FF00', '#0000FF']
plt.contourf(xx, yy, Z, cmap=cmap_light, alpha=0.3)
for idx, species in enumerate(best_knn.classes_):
    plt.scatter(X_subset[y == species, 0], X_subset[y == species, 1],
                c=cmap_bold[idx], label=species, edgecolor='k')
plt.xlabel('Petal Length (scaled)')
plt.ylabel('Petal Width (scaled)')
plt.title(f'KNN Decision Boundaries (k={grid_search.best_params_["n_neighbors"]})')
plt.legend()
plt.tight_layout()
plt.show()
```

```
Best Parameters: {'n_neighbors': 3}
Accuracy: 1.0000
Mean Squared Error (based on probabilities): 0.0049
R² Score (based on probabilities): 0.9776
```



In [31]:

```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Decision boundary visualization (2D for petal length & width)
import numpy as np

# Use only two features for 2D plot
X_vis = X[:, 2:4]
X_train_vis, X_test_vis, _, _ = train_test_split(X_vis, y, test_size=0.3, random_state=42)

# Fit model again on reduced features
model_vis = LogisticRegression()
model_vis.fit(X_train_vis, y_train)
```

```
# Plotting
x_min, x_max = X_vis[:, 0].min() - 1, X_vis[:, 0].max() + 1
y_min, y_max = X_vis[:, 1].min() - 1, X_vis[:, 1].max() + 1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 200),
                     np.linspace(y_min, y_max, 200))
Z = model_vis.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure(figsize=(8, 6))
plt.contourf(xx, yy, Z, alpha=0.3, cmap='RdBu')
sns.scatterplot(x=X_vis[:, 0], y=X_vis[:, 1], hue=y, palette='Set1', edgecolor='k')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.title('Logistic Regression Decision Boundary')
plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\pandas\core\indexes\base.py:3791, in Index.get_loc
(self, key)
   3790 try:
-> 3791     return self._engine.get_loc(casted_key)
   3792 except KeyError as err:

File index.pyx:152, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:158, in pandas._libs.index.IndexEngine.get_loc()

TypeError: '(slice(None, None, None), slice(2, 4, None))' is an invalid key

During handling of the above exception, another exception occurred:

InvalidIndexError                         Traceback (most recent call last)
Cell In[31], line 9
      6 import numpy as np
      8 # Use only two features for 2D plot
----> 9 X_vis = X[:, 2:4]
     10 X_train_vis, X_test_vis, _, _ = train_test_split(X_vis, y, test_size=0.3, random_state=42)
     12 # Fit model again on reduced features

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\pandas\core\frame.py:3893, in DataFrame.__getitem_
_(self, key)
   3891 if self.columns.nlevels > 1:
   3892     return self._getitem_multilevel(key)
-> 3893 indexer = self.columns.get_loc(key)
   3894 if is_integer(indexer):
   3895     indexer = [indexer]

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\pandas\core\indexes\base.py:3803, in Index.get_loc
(self, key)
   3798     raise KeyError(key) from err
   3799 except TypeError:
   3800     # If we have a listlike key, _check_indexing_error will raise
   3801     #  InvalidIndexError. Otherwise we fall through and re-raise
   3802     #  the TypeError.
-> 3803     self._check_indexing_error(key)
   3804     raise

File ~\AppData\Local\Programs\Python\Python311\Lib\site-packages\pandas\core\indexes\base.py:5975, in Index._check_
indexing_error(self, key)
   5971 def _check_indexing_error(self, key):
   5972     if not is_scalar(key):
   5973         # if key is not a scalar, directly raise an error (the code below
   5974         # would convert to numpy arrays and raise later any way) - GH29926
-> 5975         raise InvalidIndexError(key)

InvalidIndexError: (slice(None, None, None), slice(2, 4, None))
```

In [33]:

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load Iris dataset
dataset = load_iris()
X, y = dataset.data, dataset.target

# For binary classification (e.g., Setosa vs. not Setosa)
y = (y == 0).astype(int)

# Split dataset
```

```python
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=40
)

# Initialize logistic regression model
model = LogisticRegression(
    penalty='l2',
    C=2.0,
    solver='liblinear',
    max_iter=1000
)

# Train model
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

In [ ]: