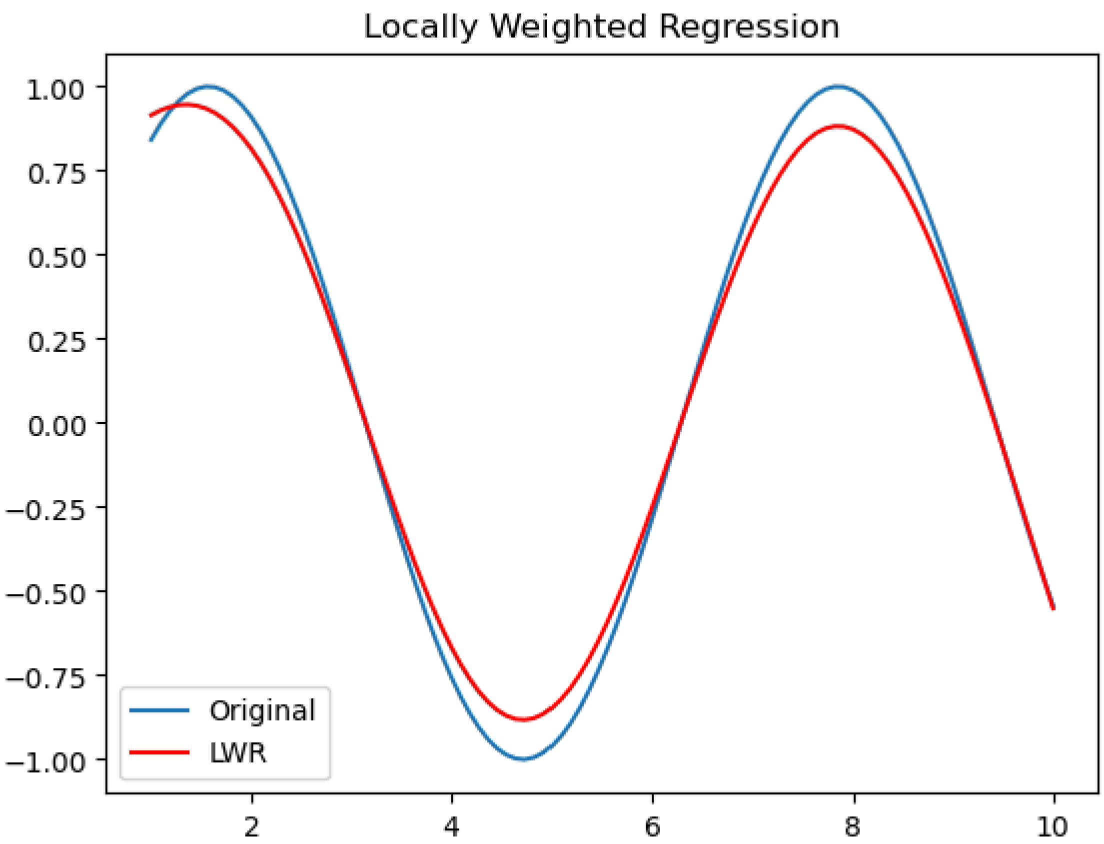
6

In [16]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split

print("\n--- Locally Weighted Regression ---")
def kernel(x, xi, tau):
    return np.exp(-np.sum((x - xi) ** 2) / (2 * tau ** 2))

def predict_lwr(x, X, y, tau=0.5):
    W = np.array([kernel(x, xi, tau) for xi in X])
    W = np.diag(W)
    theta = np.linalg.pinv(X.T @ W @ X) @ X.T @ W @ y
    return x @ theta

X = np.linspace(1, 10, 100).reshape(-1, 1)
y = np.sin(X).ravel()
X_ = np.c_[np.ones(X.shape[0]), X]
preds = [predict_lwr(xi, X_, y) for xi in X_]
plt.plot(X, y, label='Original')
plt.plot(X, preds, label='LWR', color='r')
plt.legend()
plt.title('Locally Weighted Regression')
plt.show()
```

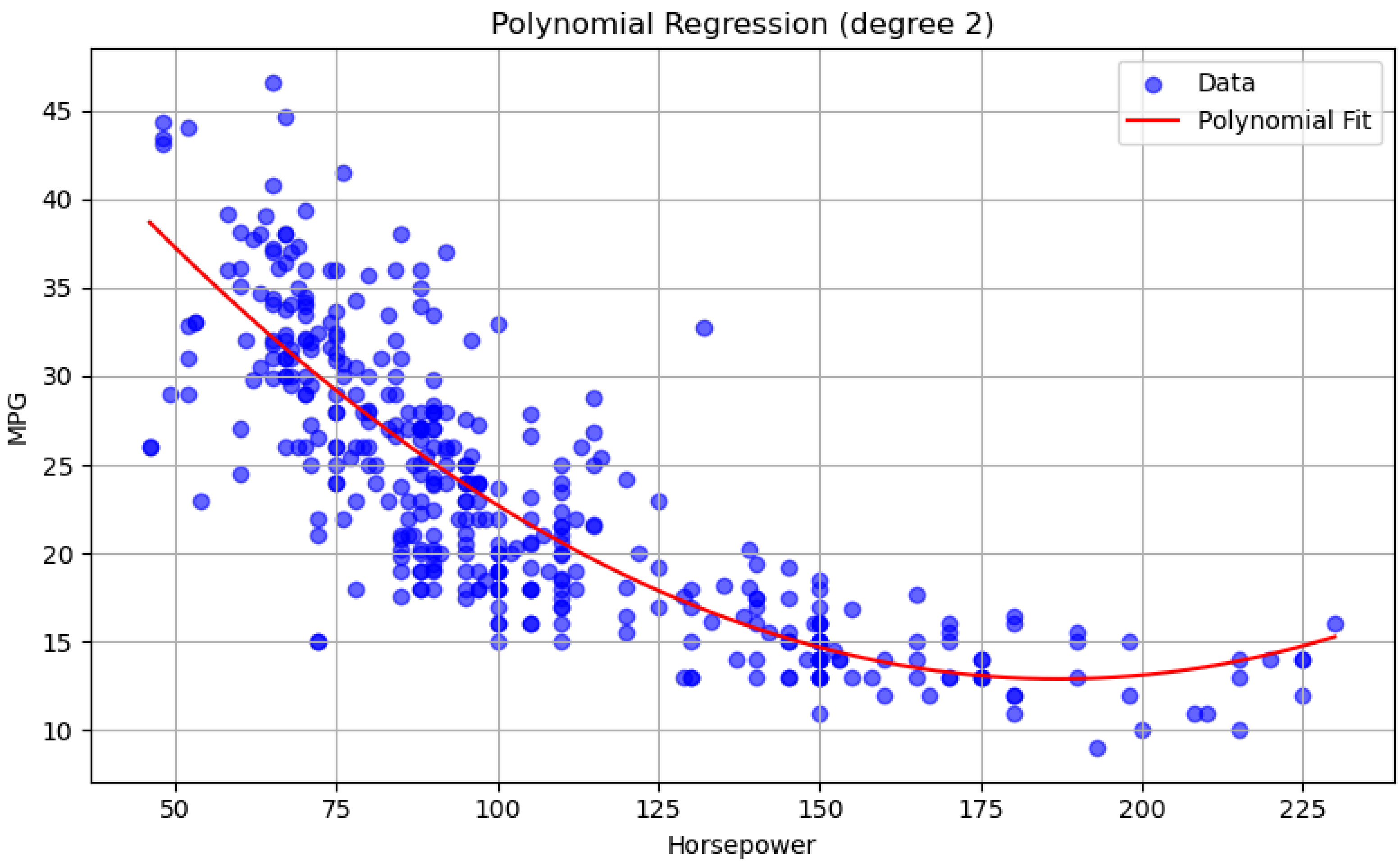
```
--- Locally Weighted Regression ---
```



7

In [20]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Load and clean data
df = sns.load_dataset("mpg").dropna(subset=['horsepower', 'mpg'])
X_train, X_test, y_train, y_test = train_test_split(df[['horsepower']], df['mpg'], test_size=0.2,
                                                    random_state=42)
```

```python
# Fit polynomial regression model
degree = 2
poly = PolynomialFeatures(degree).fit(X_train)
model = LinearRegression().fit(poly.transform(X_train), y_train)

# Predict and plot
X_range = np.linspace(df['horsepower'].min(), df['horsepower'].max(), 200).reshape(-1, 1)
y_range_pred = model.predict(poly.transform(X_range))

plt.figure(figsize=(8, 5))
plt.scatter(df['horsepower'], df['mpg'], color='blue', alpha=0.6, label='Data')
plt.plot(X_range, y_range_pred, color='red', label='Polynomial Fit')
plt.xlabel('Horsepower')
plt.ylabel('MPG')
plt.title(f'Polynomial Regression (degree {degree})')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [24]:

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

# Load dataset
data = pd.read_csv(r"C:\Users\PAVITHRA H R\Downloads\Boston housing dataset.csv")

# Check for NaN values and handle them
```

```python
if data.isnull().values.any():
    print("Data contains NaN values. Filling NaNs with column means.")
    data.fillna(data.mean(), inplace=True)  # Fill NaNs with column means

# Define features and target
X, y = data.drop(columns='MEDV'), data['MEDV']

# Standardize features and split data
X_scaled = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
                                                    random_state=42)

# Train model and predict
model = LinearRegression().fit(X_train, y_train)
y_pred = model.predict(X_test)

# Evaluate and display metrics
print(f"MSE: {mean_squared_error(y_test, y_pred):.2f}, RMSE: {np.sqrt(mean_squared_error
(y_test, y_pred)):.2f}, R²: {r2_score(y_test, y_pred):.2f}")

# Plot actual vs predicted
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.6, edgecolors="k", s=80)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red',
         linestyle='--')
plt.title('Linear Regression: Actual vs Predicted')
plt.xlabel('Actual MEDV')
plt.ylabel('Predicted MEDV')
plt.grid(True)
plt.tight_layout()
plt.show()
```

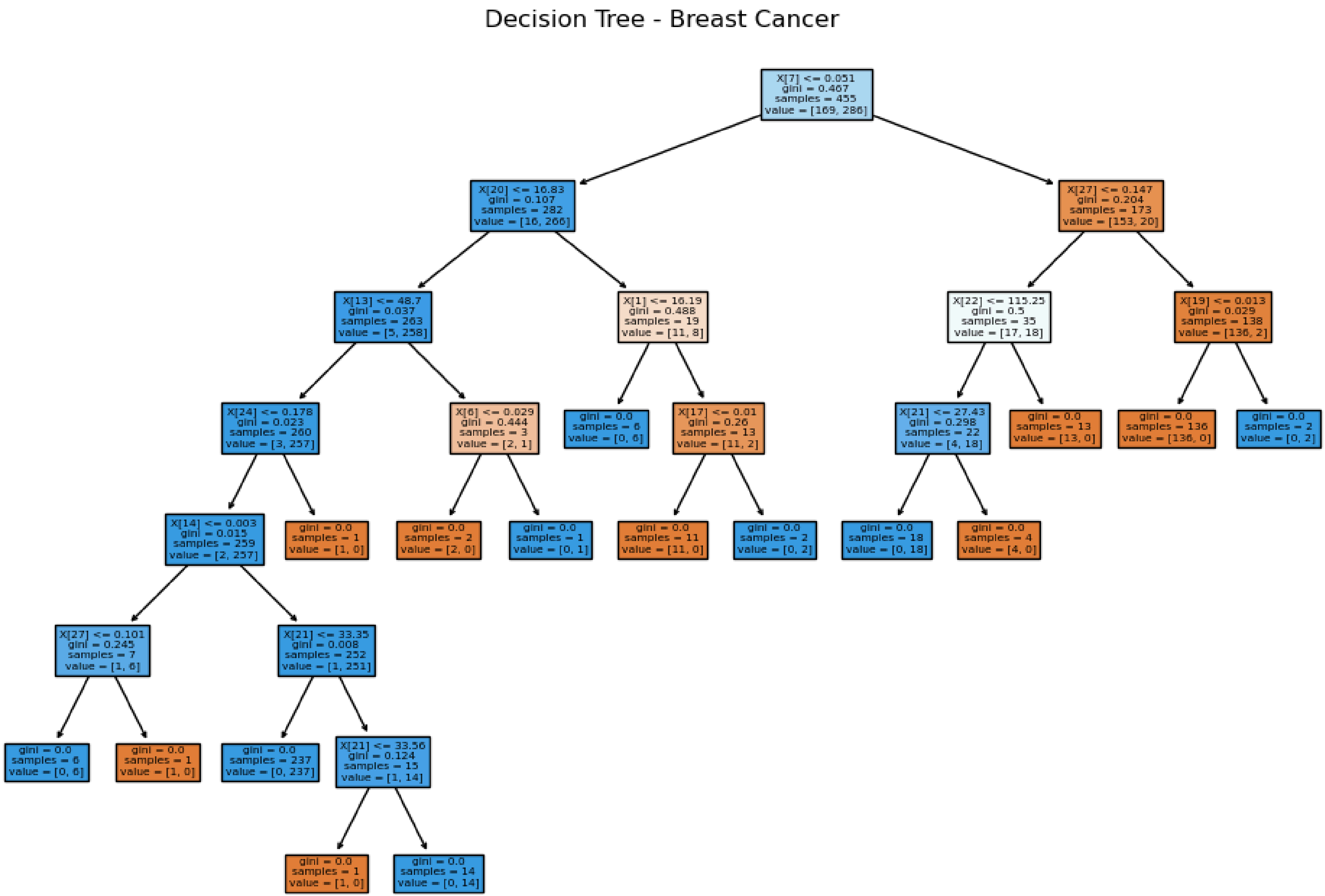
```
Data contains NaN values. Filling NaNs with column means.
MSE: 25.02, RMSE: 5.00, R²: 0.66
```

8

In [19]:
```
import numpy as np
import pandas as pd
```

```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score

print("\n--- Decision Tree ---")
data = load_breast_cancer()
X, y = data.data, data.target

# Split the data 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)

# Fit the Decision Tree model
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

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

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Plot the decision tree
plt.figure(figsize=(12, 8))
plot_tree(model, filled=True)
plt.title('Decision Tree - Breast Cancer')
plt.show()
```

```
--- Decision Tree ---
Accuracy: 0.94
```


Decision Tree - Breast Cancer
X[7] <= 0.051
gini = 0.467
samples = 455
value = [169, 286]
X[20] <= 16.83
gini = 0.107
samples = 282
value = [16, 266]
X[27] <= 0.147
gini = 0.204
samples = 173
value = [153, 20]
X[13] <= 48.7
gini = 0.037
samples = 263
value = [5, 258]
X[1] <= 16.19
gini = 0.488
samples = 19
value = [11, 8]
X[22] <= 115.25
gini = 0.5
samples = 35
value = [17, 18]
X[19] <= 0.013
gini = 0.029
samples = 138
value = [136, 2]
X[24] <= 0.178
gini = 0.023
samples = 260
value = [3, 257]
X[6] <= 0.029
gini = 0.444
samples = 3
value = [2, 1]
gini = 0.0
samples = 6
value = [0, 6]
X[17] <= 0.01
gini = 0.26
samples = 13
value = [11, 2]
X[21] <= 27.43
gini = 0.298
samples = 22
value = [4, 18]
gini = 0.0
samples = 13
value = [13, 0]
gini = 0.0
samples = 136
value = [136, 0]
gini = 0.0
samples = 2
value = [0, 2]
X[14] <= 0.003
gini = 0.015
samples = 259
value = [2, 257]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 2
value = [2, 0]
gini = 0.0
samples = 1
value = [0, 1]
gini = 0.0
samples = 11
value = [11, 0]
gini = 0.0
samples = 2
value = [0, 2]
gini = 0.0
samples = 18
value = [0, 18]
gini = 0.0
samples = 4
value = [4, 0]
X[27] <= 0.101
gini = 0.245
samples = 7
value = [1, 6]
X[21] <= 33.35
gini = 0.008
samples = 252
value = [1, 251]
gini = 0.0
samples = 6
value = [0, 6]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 237
value = [0, 237]
X[21] <= 33.56
gini = 0.124
samples = 15
value = [1, 14]
gini = 0.0
samples = 1
value = [1, 0]
gini = 0.0
samples = 14
value = [0, 14]

In [15]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_olivetti_faces
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

data = fetch_olivetti_faces()

data.keys()


def print_faces(images, target, top_n):
    # Ensure the number of images does not exceed available data
    top_n = min(top_n, len(images))

    # Set up figure size based on the number of images
    grid_size = int(np.ceil(np.sqrt(top_n)))
    fig, axes = plt.subplots(grid_size, grid_size, figsize=(15, 15))
    fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.2, wspace=0.2)

    for i, ax in enumerate(axes.ravel()):
        if i < top_n:
            ax.imshow(images[i], cmap='bone')
            ax.axis('off')
            ax.text(2, 12, str(target[i]), fontsize=9, color='red')
            ax.text(2, 55, f"face: {i}", fontsize=9, color='blue')
        else:
            ax.axis('off')

    plt.show()
print_faces(data.images,data.target,10)
```


0
face: 0
0
face: 1
0
face: 2
0
face: 3
0
face: 4
0
face: 5
0
face: 6
0
face: 7
0
face: 8
0
face: 9

In [13]:
```
print("\n--- Naive Bayes - Olivetti Faces ---")
faces = fetch_olivetti_faces()
X, y = faces.data, faces.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

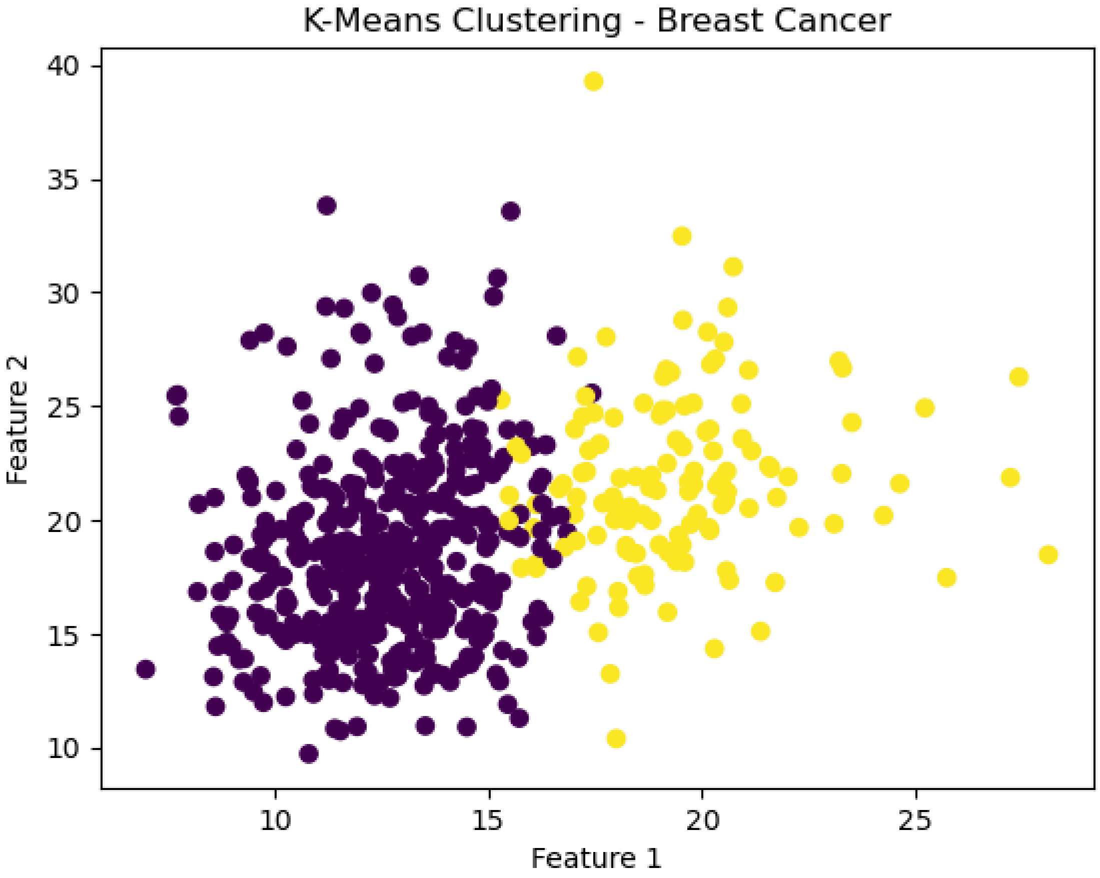
```
--- Naive Bayes - Olivetti Faces ---
Accuracy: 0.7875
```

10

In [10]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split

from sklearn.cluster import KMeans
print("\n--- K-Means Clustering ---")
data = load_breast_cancer()
X = data.data
model = KMeans(n_clusters=2, n_init=10)
model.fit(X)
labels = model.labels_
plt.scatter(X[:, 0], X[:, 1], c=labels)
```

```python
plt.title('K-Means Clustering - Breast Cancer')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

```
--- K-Means Clustering ---
```



In [ ]: