Experiment 1

Part 1: To solve the system of equations for

2x+3y=6
-4x+y=-8

using matrix method

In [3]:
```
import numpy as np

# Coefficient matrix A
A = np.array([
    [2, 3],
    [-4, 1]
])

# Constant matrix B
B = np.array([6, -8])

# Solve the system AX = B
X = np.linalg.solve(A, B)

print("Solution:")
print(f"x = {X[0]}")
print(f"y = {X[1]}")
```

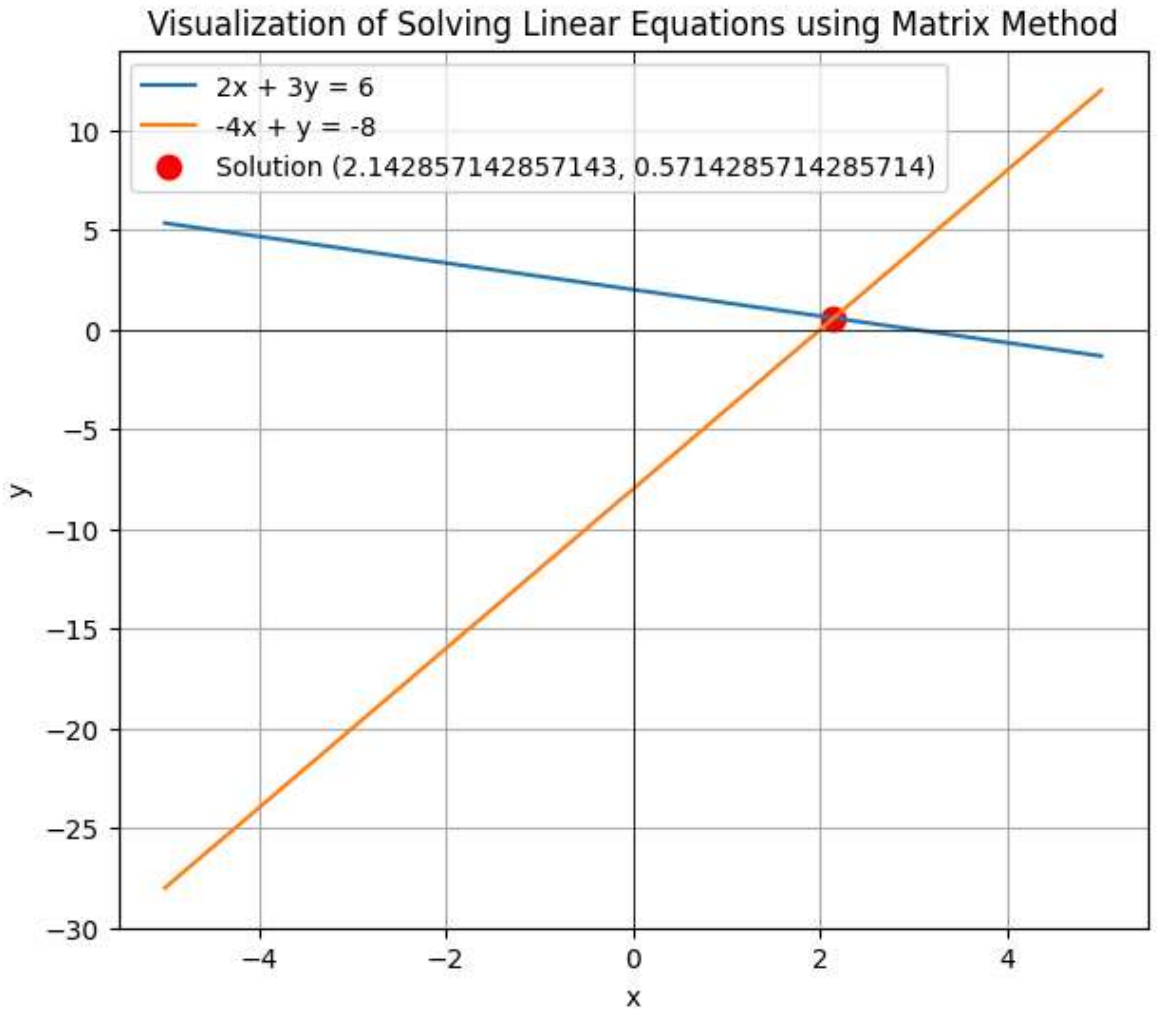
```
Solution:
x = 2.142857142857143
y = 0.5714285714285714
```

Part 2 : Write python code and visualize the method

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt

# System of equations:
# 2x + 3y = 6
# -4x + y = -8

# Coefficient matrix A
A = np.array([[2, 3],
              [-4, 1]])

# Constant matrix B
B = np.array([6, -8])

# Solve the matrix equation AX = B
solution = np.linalg.solve(A, B)
x_sol, y_sol = solution

print("Solution:")
```

```
print("x =", x_sol)
print("y =", y_sol)

# -------- Visualization -------- #

# Create x values for plotting
x = np.linspace(-5, 5, 400)

# Convert each equation to y = mx + c form
y1 = (6 - 2*x) / 3        # From 2x + 3y = 6
y2 = -8 + 4*x             # From -4x + y = -8

plt.figure(figsize=(7, 6))

# Plot the two lines
plt.plot(x, y1, label="2x + 3y = 6")
plt.plot(x, y2, label="-4x + y = -8")

# Mark the solution point
plt.scatter(x_sol, y_sol, color='red', s=80, label=f"Solution ({x_sol}, {y_sol})")

# Add labels and grid
plt.xlabel("x")
plt.ylabel("y")
plt.title("Visualization of Solving Linear Equations using Matrix Method")
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.grid(True)
plt.legend()

plt.show()
```

```
Solution:
x = 2.142857142857143
y = 0.5714285714285714
```



Part 3 :

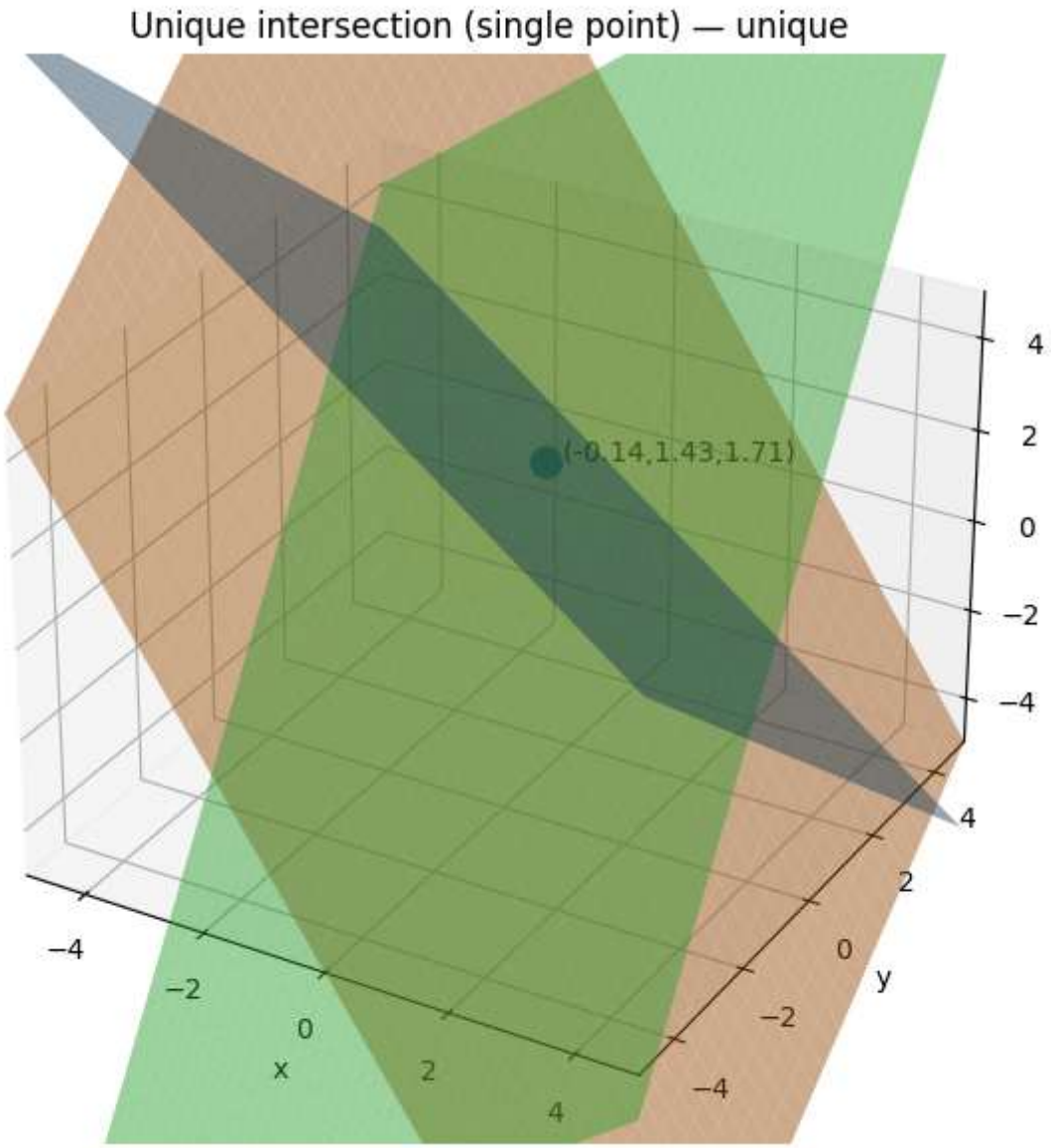
In [5]:
```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # noqa: F401

def plane_from_coeffs(coeffs):
    """Return (a,b,c,d) from [a,b,c] and d (so plane: a x + b y + c z = d)."""
    a, b, c = coeffs[0]
    d = coeffs[1]
    return a, b, c, d

def rank_and_solution(A, b, tol=1e-10):
    """Return ranks and classification and solution or paramization."""
    # ranks
    rank_A = np.linalg.matrix_rank(A, tol)
    aug = np.hstack([A, b.reshape(-1,1)])
    rank_aug = np.linalg.matrix_rank(aug, tol)

    result = {"rank_A": rank_A, "rank_aug": rank_aug}

    if rank_A == rank_aug == 3:
        # unique
        x = np.linalg.solve(A, b)
        result.update({"type": "unique", "solution": x})
    elif rank_A != rank_aug:
        result.update({"type": "inconsistent", "solution": None})
```

```
    else:
        # rank_A == rank_aug < 3 => infinitely many solutions
        # find particular solution (least squares) and nullspace basis
        x0, *_ = np.linalg.lstsq(A, b, rcond=None)
        # nullspace using SVD
        U, S, Vt = np.linalg.svd(A)
        # nullspace dimension
        null_mask = (S <= tol)
        # but S contains only min(m,n) singular values; better compute nullspace
        # Determine nullspace vectors as rows of Vt corresponding to zero singul
        nullspace = Vt.T[:, np.where(S <= tol)[0]] if np.any(S <= tol) else Vt.T
        # if above fails, compute nullspace via SVD rank
        if nullspace.size == 0:
            # compute via SVD more robustly
            nullspace = Vt.T[:, rank_A:]
        result.update({"type": "infinite", "particular": x0, "nullspace": nullsp
    return result

def plot_planes_and_solution(case_name, planes, analysis):
    """
    planes: list of tuples (a,b,c,d) representing a x + b y + c z = d
    analysis: output from rank_and_solution
    """
    # Create grid for plotting planes
    xx, yy = np.meshgrid(np.linspace(-5,5,40), np.linspace(-5,5,40))
    fig = plt.figure(figsize=(8,6))
    ax = fig.add_subplot(111, projection='3d')
    ax.set_title(case_name + " — " + analysis["type"])
    ax.set_xlabel('x'); ax.set_ylabel('y'); ax.set_zlabel('z')
    ax.set_xlim(-5,5); ax.set_ylim(-5,5); ax.set_zlim(-5,5)

    # Plot planes
    for i, (a,b,c,d) in enumerate(planes):
        # avoid division by zero for c; if c is zero, solve for y instead
        if abs(c) > 1e-8:
            z = (d - a*xx - b*yy) / c
            ax.plot_surface(xx, yy, z, alpha=0.45, linewidth=0, rstride=1, cstri
        elif abs(b) > 1e-8:
            # solve for y: y = (d - a x - c z)/b -> treat z values from mesh as
            zz, xx2 = np.meshgrid(np.linspace(-5,5,40), np.linspace(-5,5,40))
            yy2 = (d - a*xx2 - c*zz) / b
            ax.plot_surface(xx2, yy2, zz, alpha=0.45, linewidth=0)
        else:
            # solve for x: x = d / a (vertical plane)
            x_const = d / a
            yy2, zz2 = np.meshgrid(np.linspace(-5,5,40), np.linspace(-5,5,40))
            xx2 = np.full_like(yy2, x_const)
            ax.plot_surface(xx2, yy2, zz2, alpha=0.45, linewidth=0)

    # Plot according to type
    if analysis["type"] == "unique":
        x, y, z = analysis["solution"]
        ax.scatter([x], [y], [z], s=120)
        ax.text(x, y, z, f"  ({x:.2f},{y:.2f},{z:.2f})", size=10)
    elif analysis["type"] == "inconsistent":
        # No common intersection; nothing extra to plot
        pass
    elif analysis["type"] == "infinite":
        # Determine nullspace dimension
        nullspace = analysis["nullspace"]
```

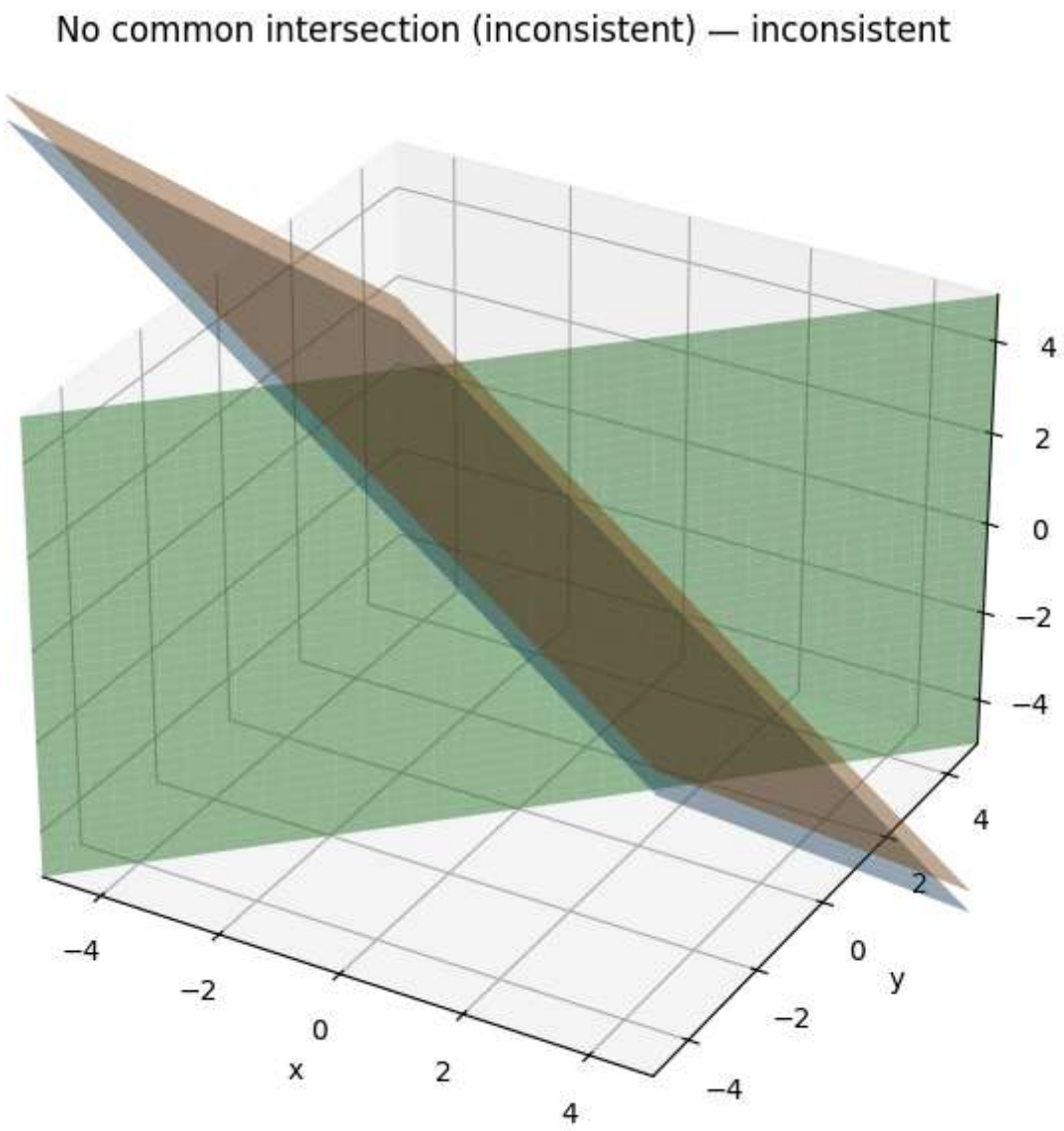
```
        x0 = analysis["particular"].flatten()
        ns_dim = nullspace.shape[1]
        if ns_dim == 1:
            # intersection is a line: param t along nullspace vector
            v = nullspace[:,0].flatten()
            t = np.linspace(-10,10,200)
            line = x0.reshape(3,1) + np.outer(v, t)
            ax.plot(line[0,:], line[1,:], line[2,:], linewidth=3)
            ax.scatter([x0[0]], [x0[1]], [x0[2]], s=60)
            ax.text(x0[0], x0[1], x0[2], "  particular", size=9)
        elif ns_dim == 2:
            # infinite plane of solutions: plot the plane of solutions spanning
            v1 = nullspace[:,0].flatten()
            v2 = nullspace[:,1].flatten()
            s = np.linspace(-5,5,30)
            t = np.linspace(-5,5,30)
            S, T = np.meshgrid(s,t)
            P = x0.reshape(3,1,1) + v1.reshape(3,1,1)*S + v2.reshape(3,1,1)*T
            ax.plot_surface(P[0,:,:], P[1,:,:], P[2,:,:], alpha=0.6)
            ax.scatter([x0[0]], [x0[1]], [x0[2]], s=60)
            ax.text(x0[0], x0[1], x0[2], "  particular", size=9)

    plt.tight_layout()
    plt.show()

# --- Example cases ---
cases = {
    "Unique intersection (single point)": {
        # planes defined as [(a,b,c), d]
        "planes": [
            (np.array([1, 1, 1]), 3),
            (np.array([2, -1, 1]), 0),
            (np.array([1, 2, -1]), 1)
        ]
    },
    "No common intersection (inconsistent)": {
        # Make two planes parallel but inconsistent, and a third plane that inte
        "planes": [
            (np.array([1, 1, 1]), 1),
            (np.array([2, 2, 2]), 3),  # inconsistent multiple of first but diff
            (np.array([1, -1, 0]), 0)
        ]
    },
    "Infinitely many solutions - line (two same planes)": {
        # First two are multiples (same geometric plane), third intersects them
        "planes": [
            (np.array([1, 1, 1]), 4),
            (np.array([2, 2, 2]), 8),  # same plane as first
            (np.array([1, -1, 0]), 0)
        ]
    },
    "Infinitely many solutions - plane (all same plane)": {
        "planes": [
            (np.array([1, 2, -1]), 3),
            (np.array([2, 4, -2]), 6),  # same as first
            (np.array([3, 6, -3]), 9)   # same plane
        ]
    }
}
```

```python
# Run through examples, analyze and plot
for name, data in cases.items():
    planes = [plane_from_coeffs(p) for p in data["planes"]]
    A = np.vstack([p[0] for p in data["planes"]]).astype(float)
    b = np.array([p[1] for p in data["planes"]], dtype=float)
    analysis = rank_and_solution(A, b)
    print("Case:", name)
    print("Coefficient matrix A:\n", A)
    print("Right-hand side b:", b)
    print("Rank(A):", analysis["rank_A"], "Rank([A|b]):", analysis["rank_aug"])
    if analysis["type"] == "unique":
        sol = analysis["solution"]
        print("Unique solution:", sol)
    elif analysis["type"] == "inconsistent":
        print("No solution (inconsistent system).")
    else:
        x0 = analysis["particular"].flatten()
        ns = analysis["nullspace"]
        print("Infinite solutions. Particular solution (one):", x0)
        print("Nullspace basis vectors (columns):\n", ns)
        print("Nullspace dimension:", ns.shape[1])
    print("\n--- Visualizing ---\n")
    plot_planes_and_solution(name, planes, analysis)

print("Done. Replace the plane coefficients in the 'cases' dict to test your own
```

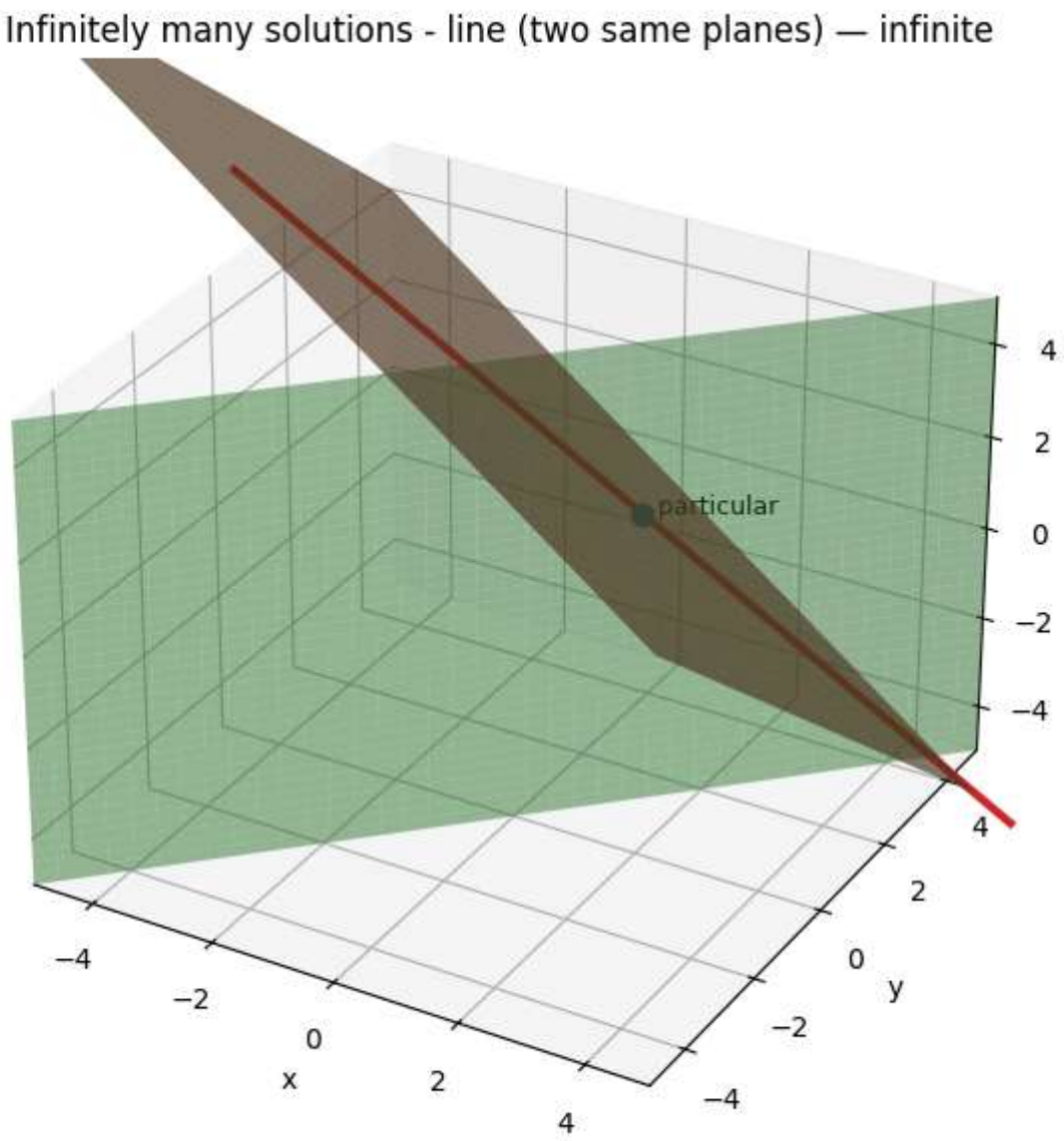
```
Case: Unique intersection (single point)
Coefficient matrix A:
 [[ 1.  1.  1.]
 [ 2. -1.  1.]
 [ 1.  2. -1.]]
Right-hand side b: [3. 0. 1.]
Rank(A): 3 Rank([A|b]): 3
Unique solution: [-0.14285714  1.42857143  1.71428571]

--- Visualizing ---
```



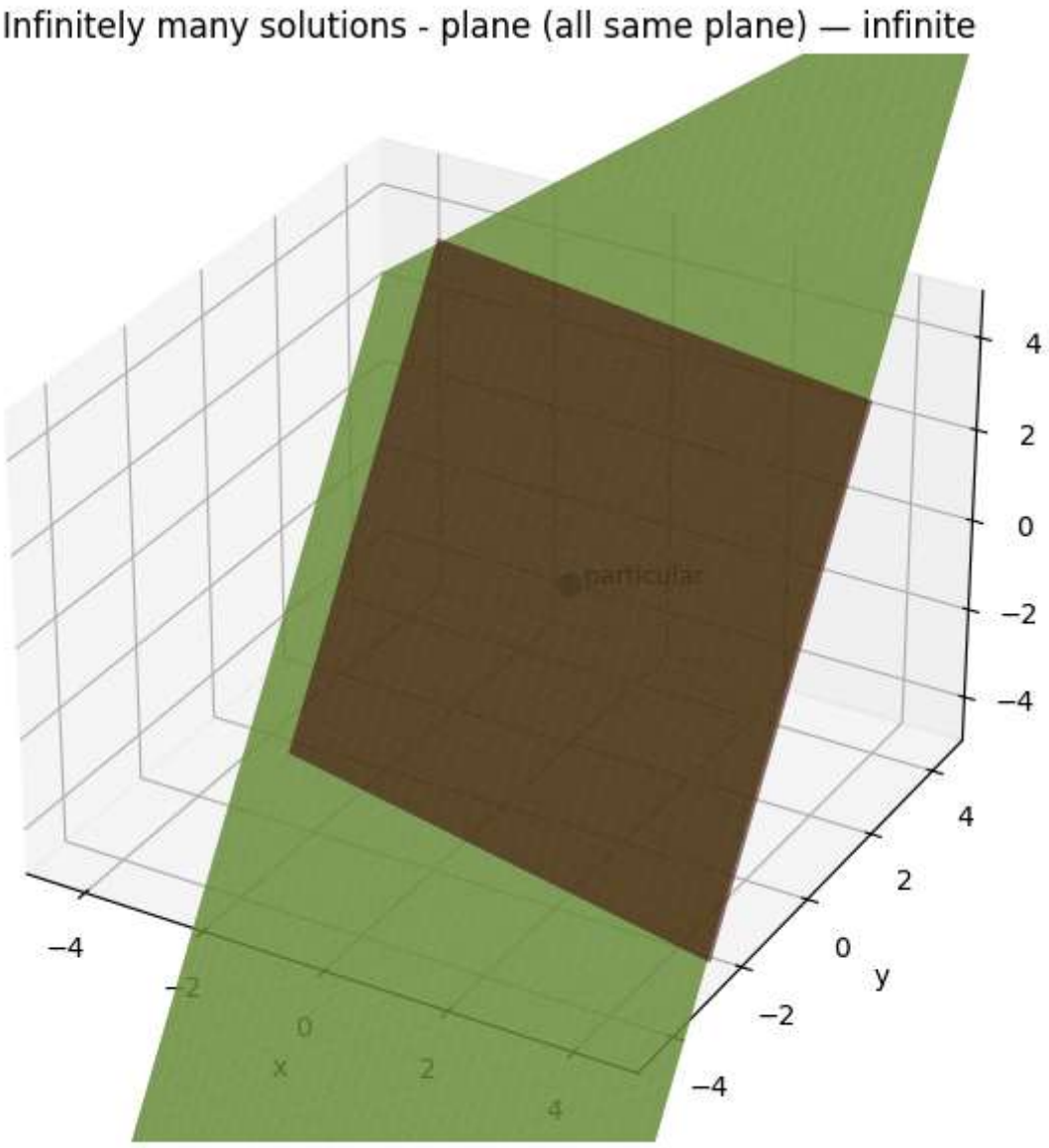

```
Case: No common intersection (inconsistent)
Coefficient matrix A:
 [[ 1.  1.  1.]
 [ 2.  2.  2.]
 [ 1. -1.  0.]]
Right-hand side b: [1. 3. 0.]
Rank(A): 2 Rank([A|b]): 3
No solution (inconsistent system).

--- Visualizing ---
```




```
Case: Infinitely many solutions - line (two same planes)
Coefficient matrix A:
 [[ 1.  1.  1.]
 [ 2.  2.  2.]
 [ 1. -1.  0.]]
Right-hand side b: [4. 8. 0.]
Rank(A): 2 Rank([A|b]): 2
Infinite solutions. Particular solution (one): [1.33333333 1.33333333 1.33333333]
Nullspace basis vectors (columns):
 [[ 0.40824829]
 [ 0.40824829]
 [-0.81649658]]
Nullspace dimension: 1

--- Visualizing ---
```




```
Case: Infinitely many solutions - plane (all same plane)
Coefficient matrix A:
 [[ 1.  2. -1.]
 [ 2.  4. -2.]
 [ 3.  6. -3.]]
Right-hand side b: [3. 6. 9.]
Rank(A): 1 Rank([A|b]): 1
Infinite solutions. Particular solution (one): [ 0.5  1.  -0.5]
Nullspace basis vectors (columns):
 [[-0.91287093  0.        ]
 [ 0.36514837  0.4472136 ]
 [-0.18257419  0.89442719]]
Nullspace dimension: 2

--- Visualizing ---
```



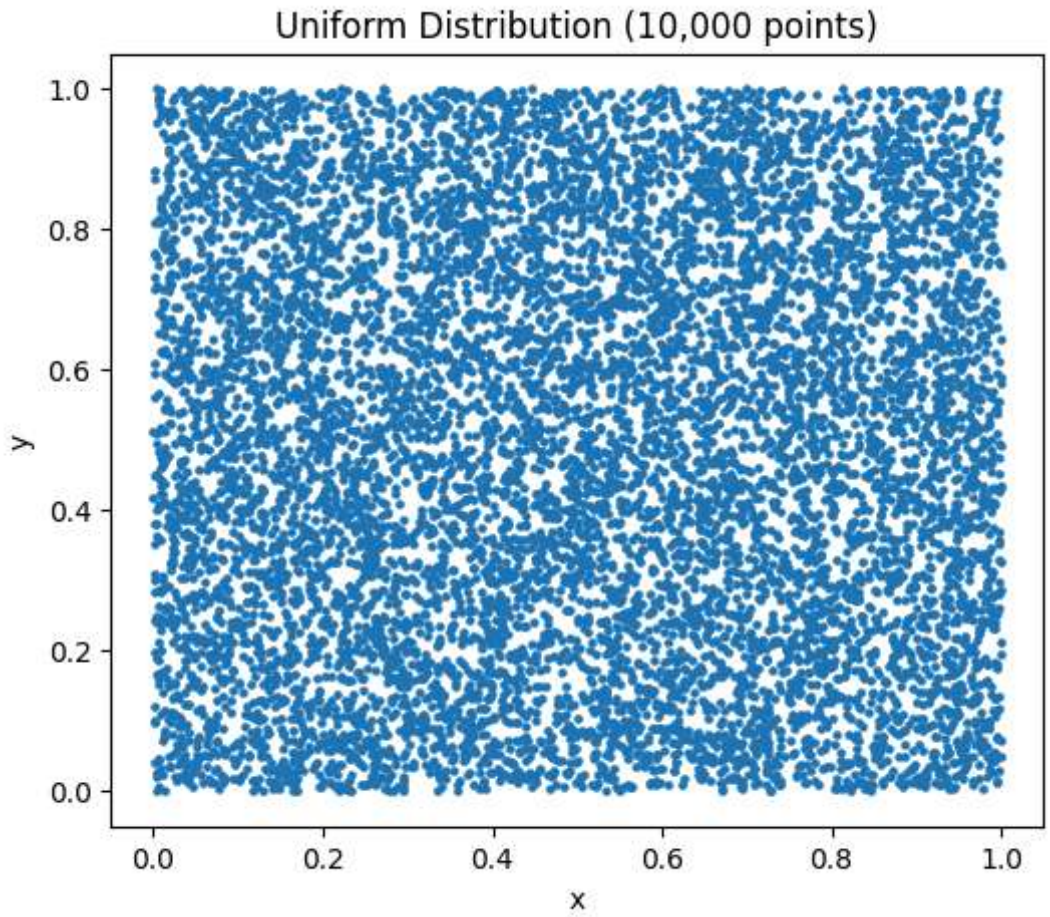

```
Done. Replace the plane coefficients in the 'cases' dict to test your own example
s.
```

Experiment 1.2

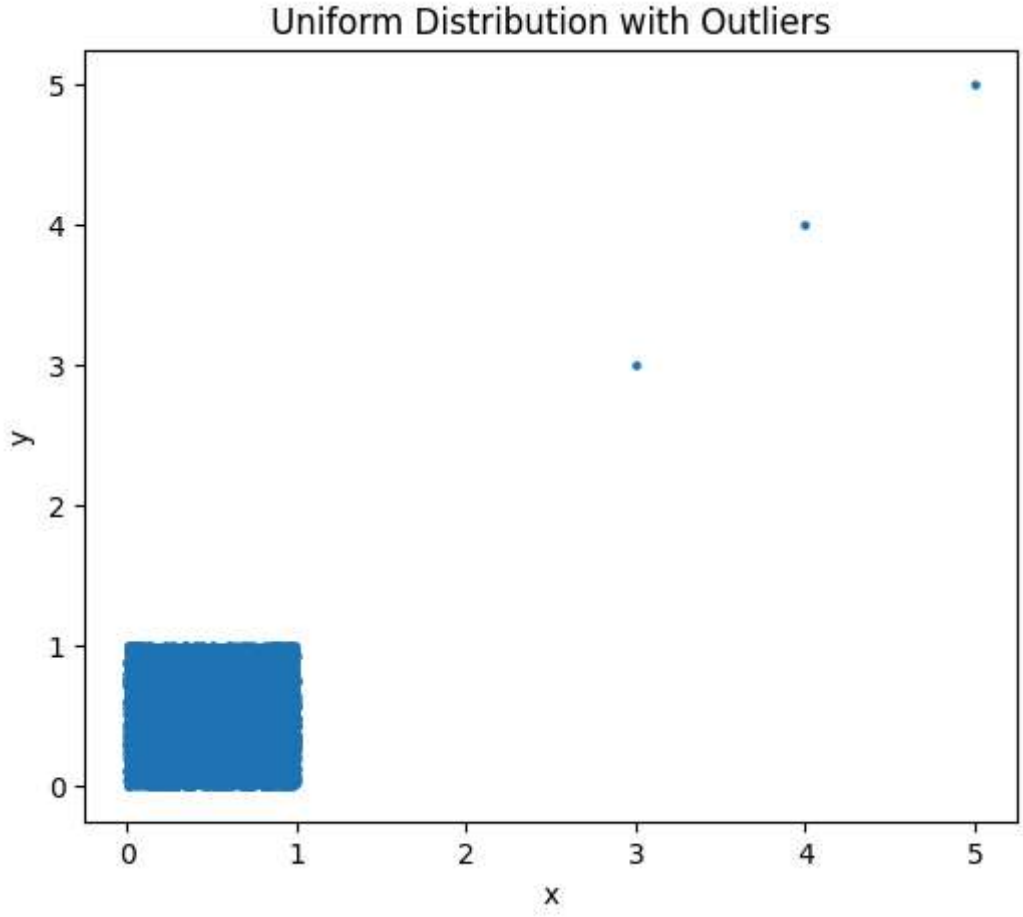
In [6]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Helper function to compute and print stats
def stats(name, data):
    mean = np.mean(data, axis=0)
    median = np.median(data, axis=0)
    print(f"{name} Mean: {mean}, Median: {median}")
    return mean, median

# ==== A) 2D Uniform Distribution ====
np.random.seed(42)
uniform_points = np.random.uniform(low=0, high=1, size=(10000, 2))

mean_u, median_u = stats("Uniform (original)", uniform_points)

plt.figure(figsize=(6,5))
plt.scatter(uniform_points[:,0], uniform_points[:,1], s=5)
```

```
plt.title("Uniform Distribution (10,000 points)")
plt.xlabel("x")
plt.ylabel("y")
plt.show()

# B) Adding outliers
outliers = np.array([[3,3], [4,4], [5,5]])
uniform_with_outliers = np.vstack([uniform_points, outliers])

mean_u_o, median_u_o = stats("Uniform (with outliers)", uniform_with_outliers)

plt.figure(figsize=(6,5))
plt.scatter(uniform_with_outliers[:,0], uniform_with_outliers[:,1], s=5)
plt.title("Uniform Distribution with Outliers")
plt.xlabel("x")
plt.ylabel("y")
plt.show()

# ==== Gaussian Distribution ====
gaussian_points = np.random.normal(loc=0, scale=1, size=(10000, 2))

mean_g, median_g = stats("Gaussian (original)", gaussian_points)

plt.figure(figsize=(6,5))
plt.scatter(gaussian_points[:,0], gaussian_points[:,1], s=5)
plt.title("Gaussian Distribution (10,000 points)")
plt.xlabel("x")
plt.ylabel("y")
plt.show()

# Adding outliers for Gaussian
gaussian_with_outliers = np.vstack([gaussian_points, outliers])

mean_g_o, median_g_o = stats("Gaussian (with outliers)", gaussian_with_outliers)

plt.figure(figsize=(6,5))
plt.scatter(gaussian_with_outliers[:,0], gaussian_with_outliers[:,1], s=5)
plt.title("Gaussian Distribution with Outliers")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

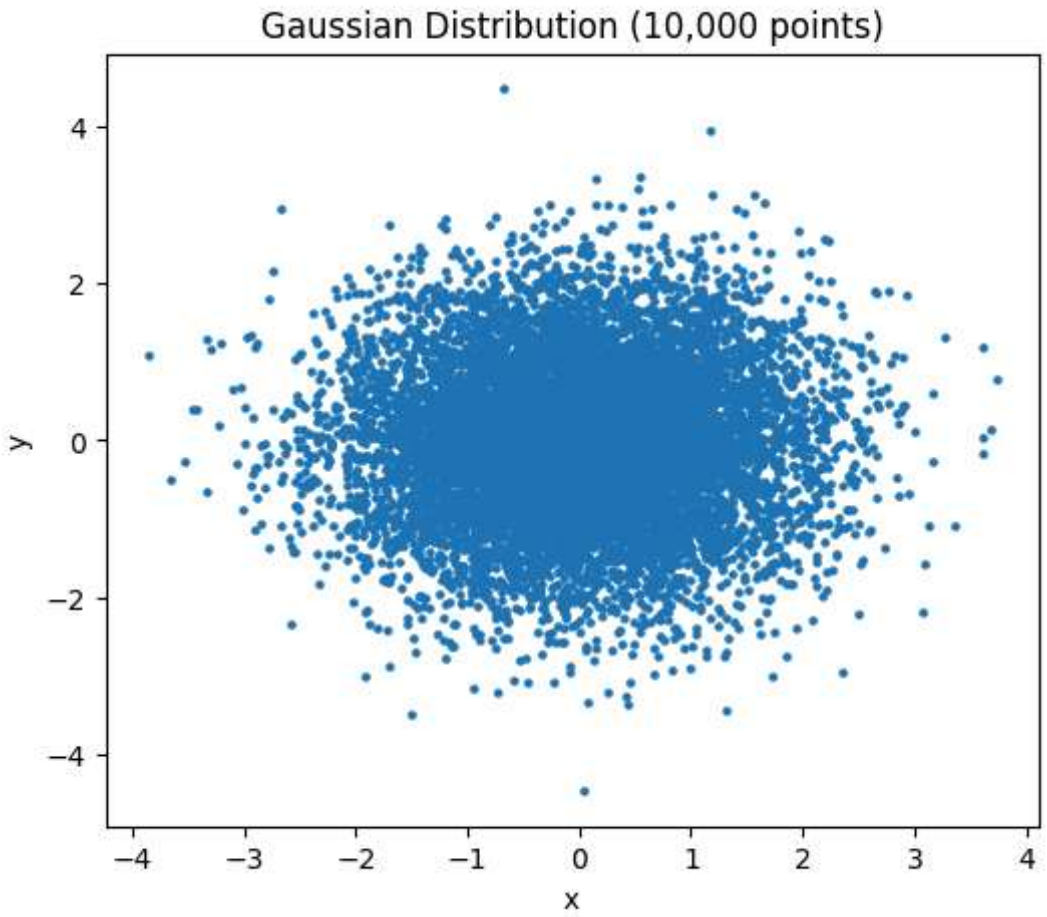
```
Uniform (original) Mean: [0.49847358 0.50021585], Median: [0.49835983 0.50027152]
```



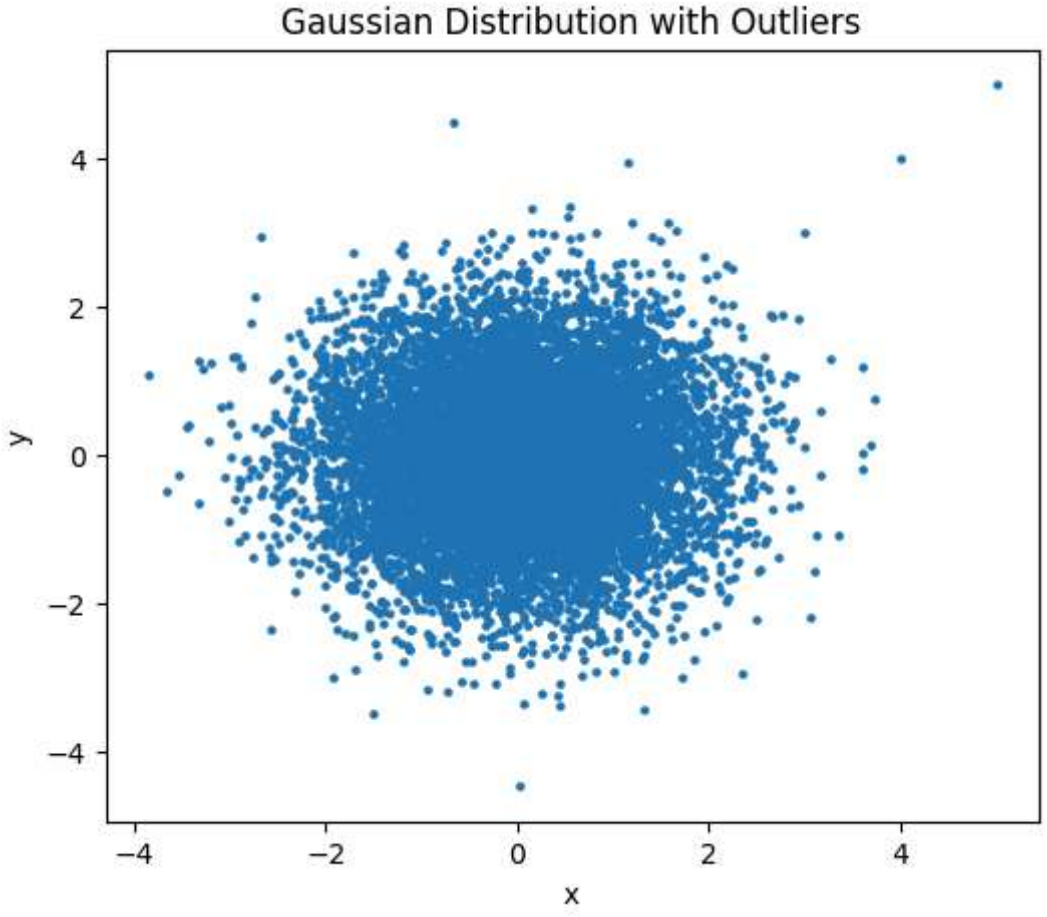

```
Uniform (with outliers) Mean: [0.49952372 0.50126547], Median: [0.49840754 0.5003
5836]
```




```
Gaussian (original) Mean: [-0.00449359 -0.00775959], Median: [ 0.00150458 -0.0059
2736]
```




```
Gaussian (with outliers) Mean: [-0.00329261 -0.00655763], Median: [ 0.00173494 -
0.00572284]
```



Experiment 2

Greenfield town is seeing rapid real estate development. A real estate company wants to predict ouse prices based solely on the size of the house (in sq ft) they have collected data from 2 recently sold houses.

Size (sqft)	Price ($)
1200	300000
1500	320000
2000	450000
1800	400000
2500	550000
3000	600000
2200	480000
2700	700000

In [48]:
```
import numpy as np
import matplotlib.pyplot as plt
# Data
sizes = np.array([1200,1500,2000,1800,2500,3000,2200,2700])
prices = np.array([300000,320000,450000,400000,550000,600000,480000,700000])

# Means
x_mean = sizes.mean()
y_mean = prices.mean()

# Compute slope m manually
numerator = np.sum((sizes - x_mean) * (prices - y_mean))
denominator = np.sum((sizes - x_mean)**2)
m = numerator / denominator

# Compute intercept b manually
b = y_mean - m * x_mean

# Predictions
y_pred = m * sizes + b

print(f"Slope (Coefficient): {m:.2f}")
print(f"Intercept: {b:.2f}")
```

```
Slope (Coefficient): 212.17
Intercept: 26789.65
```

In [49]:
```
import numpy as np

# Data
sizes = np.array([1200,1500,2000,1800,2500,3000,2200,2700])
prices = np.array([300000,320000,450000,400000,550000,600000,480000,700000])
```

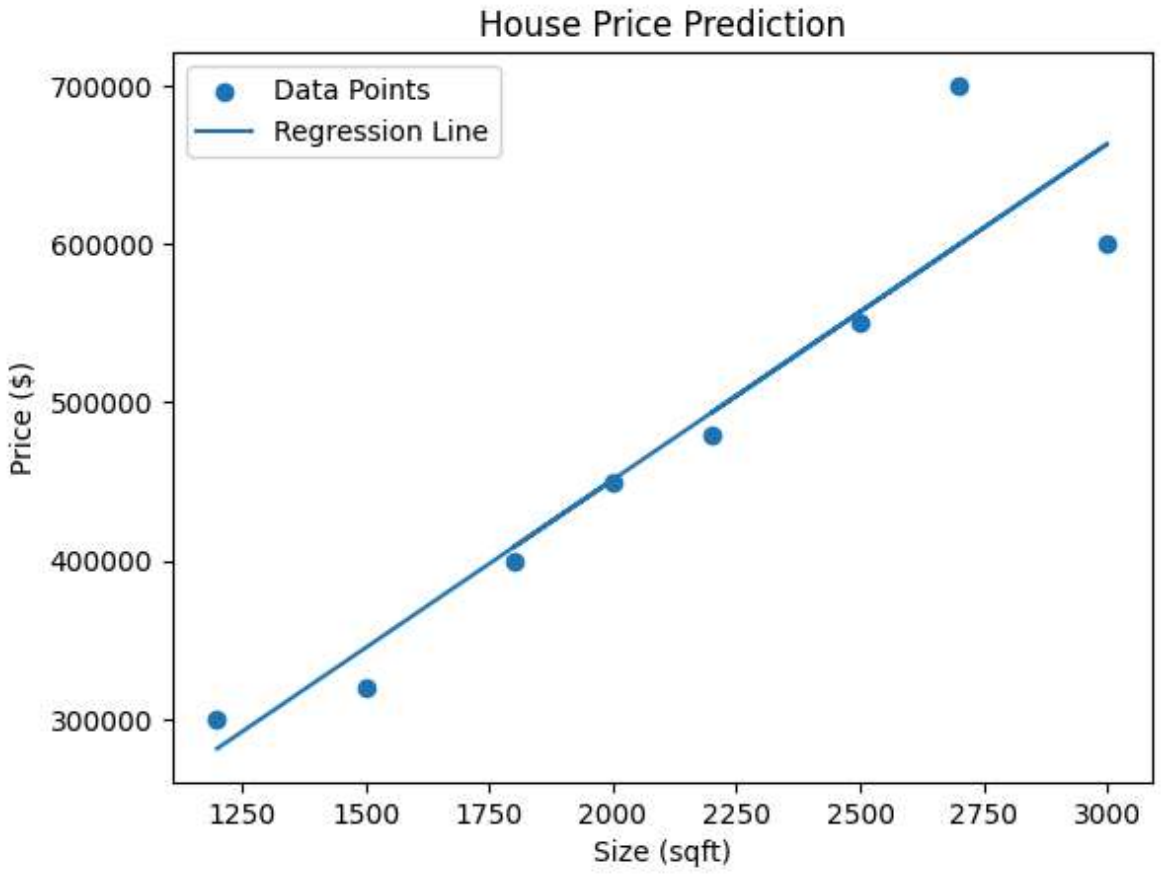
```
# Construct X matrix with a column of 1s
X = np.column_stack((np.ones(len(sizes)), sizes))

# y vector
y = prices.reshape(-1, 1)

# Closed-form solution θ = (X^T X)^(-1) X^T y
theta = np.linalg.inv(X.T @ X) @ X.T @ y

theta0 = theta[0, 0]   # intercept
theta1 = theta[1, 0]   # slope

print(f"Theta0 (Intercept): {theta0:.2f}")
print(f"Theta1 (Slope): {theta1:.2f}")
```

```
Theta0 (Intercept): 26789.65
Theta1 (Slope): 212.17
```

In [50]:
```
xpred = 1940
y_pred = m * xpred + b
print(f"Predicted price for size {xpred} sqft: {y_pred:.2f}")
```

```
Predicted price for size 1940 sqft: 438400.57
```

In [51]:
```
y_line = m * sizes + b

plt.scatter(sizes, prices, label='Data Points')
plt.plot(sizes, y_line, label='Regression Line')
plt.xlabel('Size (sqft)')
plt.ylabel('Price ($)')
plt.title('House Price Prediction')
plt.legend()
plt.show()
```



Experiment 2.2

In [52]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression

data=pd.read_csv('../assets/exp2_data.csv')
```

In [53]:
```
data.head(5)
```

Out[53]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

In [54]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Prepare X and y
X = data['YearsExperience'].values.reshape(-1, 1)
y = data['Salary'].values

# Split into train/test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

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

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

# Get slope and intercept
m = model.coef_[0]
b = model.intercept_

print(f"Slope (Coefficient): {m:.2f}")
print(f"Intercept: {b:.2f}")
```

```
Slope (Coefficient): 9423.82
Intercept: 25321.58
```

In [55]:
```
y_line = model.predict(X)  # prediction for the full dataset

plt.scatter(data['YearsExperience'], data['Salary'], color='blue', label='Data P
plt.plot(data['YearsExperience'], y_line, color='red', label='Regression Line')
```

```
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.title('Salary vs Experience')
plt.legend()
plt.show()
```

In [56]:
```
# Predict test values
y_pred = model.predict(X_test)
print(y_pred)
```

```
[115790.21011287  71498.27809463 102596.86866063  75267.80422384
  55477.79204548  60189.69970699]
```

In [57]:
```
# Calcualte R^2 score
r2_score = model.score(X_test, y_test)
print(f"R^2 Score: {r2_score:.4f}")
```

```
R^2 Score: 0.9024
```

In [58]:
```
import pickle
import os

# Ensure the folder exists (optional but recommended)
os.makedirs("../models", exist_ok=True)

filename = '../models/experiment2-2.sav'

with open(filename, 'wb') as f:
    pickle.dump(model, f)
```

In [59]:
```
loaded_model = pickle.load(open(filename, 'rb'))
loaded_model
```

Out[59]:

LinearRegression

▼ Parameters

fit_intercept	True
copy_X	True
tol	1e-06
n_jobs	None
positive	False

In [60]:
```
result=loaded_model.predict([[15]])
print(result)
```

```
[166678.81285724]
```

In [61]:
```
prediction_input=int(input("Enter years of experience: "))
print(loaded_model.predict([[prediction_input]]))
```

```
[336307.4886718]
```

Experiment 3

For a dataset given

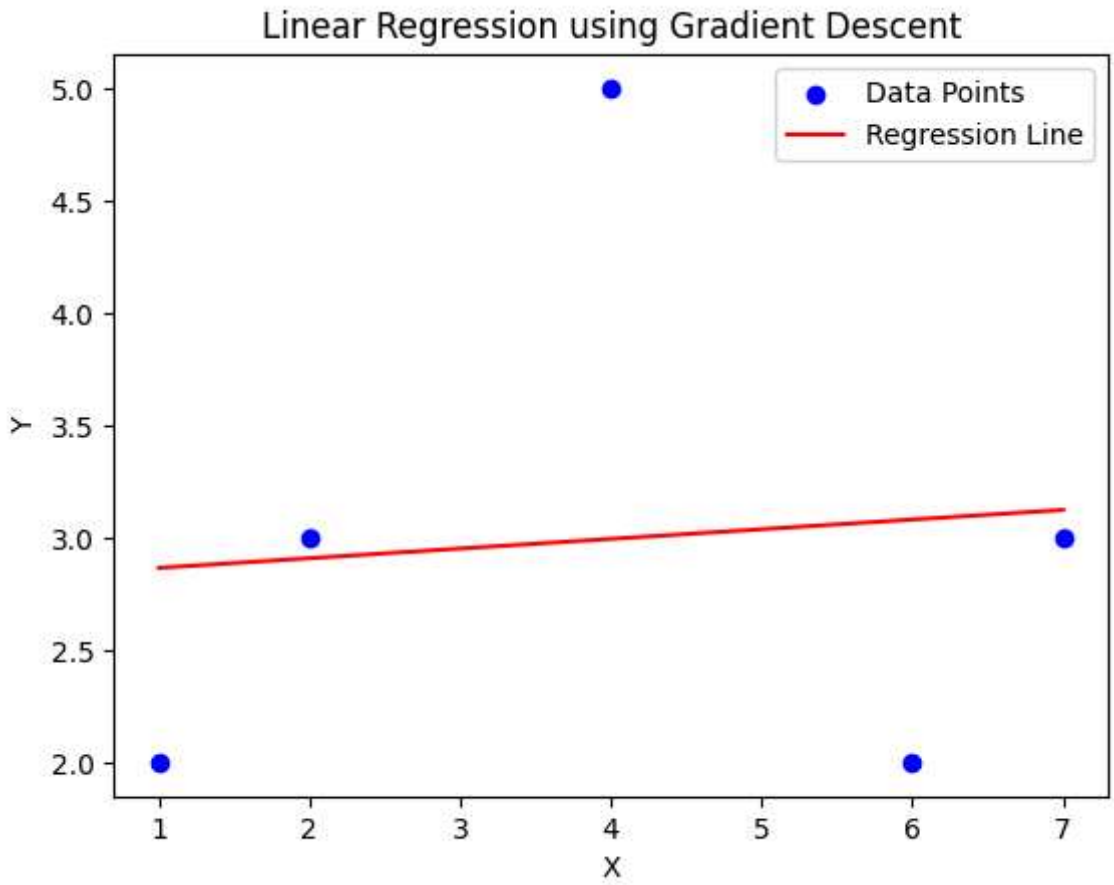
- X=[1,2,4,6,7]
- Y=[2,3,5,2,3]

write the code for linear regression using gradient descent withouth using fit function and visualtize the result

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

# initialize data
X = np.array([1,2,4,6,7])
Y = np.array([2,3,5,2,3])

# initialize parameters
m=0
c=0
alpha=0.01
epochs=1000
n=len(X)

for _ in range(epochs):
    Y_pred = m*X + c
    D_m = (-2/n) * sum(X * (Y - Y_pred))
    D_c = (-2/n) * sum(Y - Y_pred)
    m = m - alpha * D_m
    c = c - alpha * D_c

print(f"Optimized parameters: m = {m:.4f}, c = {c:.4f}")
```

```
Optimized parameters: m = 0.0430, c = 2.8224
```

In [2]:
```
# Predict Y values using the model
Y_pred = m*X + c

# Plotting the results
plt.scatter(X, Y, color='blue', label='Data Points')
plt.plot(X, Y_pred, color='red', label='Regression Line')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Linear Regression using Gradient Descent')
plt.legend()
plt.show()
```


Linear Regression using Gradient Descent
Data Points
Regression Line
5.0
4.5
4.0
3.5
3.0
2.5
2.0
Y
1
2
3
4
5
6
7
X

Objective:

To apply Multiple Linear Regression and Polynomial Regression on the California Housing dataset, handle missing values and categorical variables, and compare the performance of both models using appropriate evaluation metrics.

Tasks:

Task 1: Dataset Loading and Preprocessing

- Load the California Housing dataset using Pandas.
- Inspect the dataset to understand feature types and structure.
- Handle missing (NaN) values by replacing numerical missing values with column means.
- Convert categorical features into numerical form using One-Hot Encoding.

Task 2: Feature and Target Preparation

- Identify the target variable (`median_house_value`).
- Separate input features and the target variable.
- Split the dataset into training and testing sets (80% training, 20% testing).

Task 3: Multiple Linear Regression

- Train a Multiple Linear Regression model using the training data.
- Predict housing prices on the test dataset.
- Evaluate the model using the following metrics:
  - Mean Squared Error (MSE)
  - R^2 Score

Task 4: Polynomial Regression

- Transform input features using Polynomial Features of degree 2.
- Train a Polynomial Regression model using the transformed features.
- Predict housing prices on the test dataset.
- Evaluate the model using the following metrics:
  - Mean Squared Error (MSE)
  - R^2 Score

Task 5: Model Comparison

- Compare the performance of Multiple Linear Regression and Polynomial Regression models.
- Analyze the impact of polynomial feature expansion on model accuracy.

In [3]:
```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
import kagglehub
import os
```

```
c:\Users\adity\OneDrive\Documents\GitHub\MLLab\.venv\Lib\site-packages\tqdm\auto.
py:21: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. Se
e https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

In [4]:
```
# Download dataset from Kaggle
dataset_path = kagglehub.dataset_download(
    "camnugent/california-housing-prices"
)

# Load CSV file
csv_path = os.path.join(dataset_path, "housing.csv")
df = pd.read_csv(csv_path)

print("First 5 rows of dataset:")
df.head()
```

```
Downloading to C:\Users\adity\.cache\kagglehub\datasets\camnugent\california-hous
ing-prices\1.archive...
100%|██████████| 400k/400k [00:01<00:00, 378kB/s]
Extracting files...
First 5 rows of dataset:
```

Out[4]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-122.23	37.88	41.0	880.0	129.0	322.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0

In [5]:
```
# Fill NaN values with column mean
df.fillna(df.mean(numeric_only=True), inplace=True)

print("Missing values after handling:")
df.isnull().sum()
```

```
Missing values after handling:
```

Out[5]:
```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
median_house_value    0
ocean_proximity       0
dtype: int64
```

In [6]:
```
# Convert categorical column to numerical
df_encoded = pd.get_dummies(df, drop_first=True)

print("Columns after encoding:")
df_encoded.columns
```

```
Columns after encoding:
```

Out[6]:
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value', 'ocean_proximity_INLAND',
       'ocean_proximity_ISLAND', 'ocean_proximity_NEAR BAY',
       'ocean_proximity_NEAR OCEAN'],
      dtype='str')
```

In [7]:
```
# Target variable
y = df_encoded['median_house_value']

# Feature variables
X = df_encoded.drop('median_house_value', axis=1)
```

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)

print("Training and testing data split completed.")
```

```
Training and testing data split completed.
```

In [9]:
```
linear_model = LinearRegression()

# Train the model
linear_model.fit(X_train, y_train)

# Predict on test data
y_pred_linear = linear_model.predict(X_test)

# Evaluate performance
mse_linear = mean_squared_error(y_test, y_pred_linear)
r2_linear = r2_score(y_test, y_pred_linear)

print("--- Multiple Linear Regression Results ---")
print("Mean Squared Error (MSE):", mse_linear)
print("R² Score:", r2_linear)
```

```
--- Multiple Linear Regression Results ---
Mean Squared Error (MSE): 4904399775.949299
R² Score: 0.6257351821159687
```

In [10]:
```python
poly = PolynomialFeatures(degree=2, include_bias=False)

# Transform features
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)

# Train polynomial regression model
poly_model = LinearRegression()
poly_model.fit(X_train_poly, y_train)

# Predict on test data
y_pred_poly = poly_model.predict(X_test_poly)

# Evaluate polynomial model
mse_poly = mean_squared_error(y_test, y_pred_poly)
r2_poly = r2_score(y_test, y_pred_poly)

print("--- Polynomial Regression (Degree 2) Results ---")
print("Mean Squared Error (MSE):", mse_poly)
print("R² Score:", r2_poly)
```

```
--- Polynomial Regression (Degree 2) Results ---
Mean Squared Error (MSE): 4426103431.798885
R² Score: 0.6622349582997737
```

In [11]:
```python
results = pd.DataFrame({
    "Model": [
        "Multiple Linear Regression",
        "Polynomial Regression (Degree 2)"
    ],
    "MSE": [
        mse_linear,
        mse_poly
    ],
    "R² Score": [
        r2_linear,
        r2_poly
    ]
})

results
```

Out[11]:

	Model	MSE	R² Score
0	Multiple Linear Regression	4.904400e+09	0.625735
1	Polynomial Regression (Degree 2)	4.426103e+09	0.662235

In [12]:
```python
print("Program executed successfully.")
```

```
Program executed successfully.
```

Objective:

to generate data of degree 3 polynomial and fit models of degree 1, degree 3 and degree 13 polynomial and compare their performances.

Tasks:

Task 1: Data generation

- generate 500 samples of the target variable y using the equaton , y=0.5x^3-x^2+2 and add noise
- Visualize the generated data points

Task 2: Polynomial regression (500 samples)

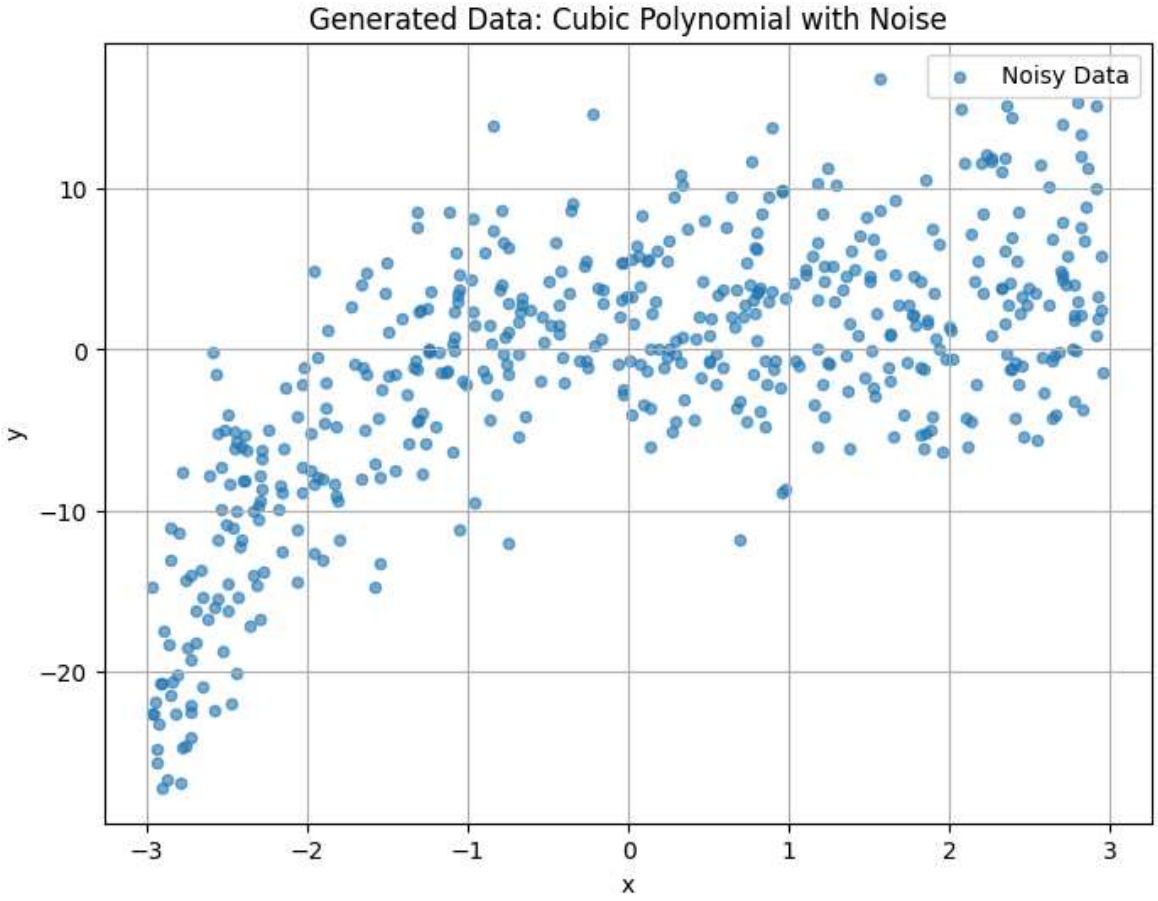
Fit polynomial regression models of the following degree to the generated data:(use polynomial features library from Scikit-learn)

- Linnear Regression mode(Degree - 1)
- Polynomial Regression model (Degree - 3)
- Polynomial Regression model (Degree - 13)
- Compare the estimated model co-efficients to the actual ground truth equations
- Evaluate the model performance using metrics: MSE , MAE, R^2

Task 1 Code:

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(42)

# Number of samples
n_samples = 500

# Generate input data
x = np.random.uniform(-3, 3, n_samples)

# True underlying function
y_true = 0.5 * x**3 - x**2 + 2

# Add Gaussian noise
noise = np.random.normal(0, 5, n_samples)
y = y_true + noise
```

```
# Visualize the generated data
plt.figure(figsize=(8, 6))
plt.scatter(x, y, s=20, alpha=0.6, label="Noisy Data")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Generated Data: Cubic Polynomial with Noise")
plt.legend()
plt.grid(True)
plt.show()
```



Task 2 Code:

In [4]:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Reshape X for sklearn
X = x.reshape(-1, 1)
y_target = y

# Degrees to test
degrees = [1, 3, 13]

results = []

for degree in degrees:
    # Polynomial feature transformation
    poly = PolynomialFeatures(degree=degree, include_bias=True)
```

```
    X_poly = poly.fit_transform(X)

    # Linear regression on transformed features
    model = LinearRegression()
    model.fit(X_poly, y_target)

    # Predictions
    y_pred = model.predict(X_poly)

    # Evaluation metrics
    mse = mean_squared_error(y_target, y_pred)
    mae = mean_absolute_error(y_target, y_pred)
    r2 = r2_score(y_target, y_pred)

    results.append({
        "Degree": degree,
        "MSE": mse,
        "MAE": mae,
        "R2": r2,
        "Coefficients": model.coef_,
        "Intercept": model.intercept_
    })

    # Display coefficients
    print(f"\nPolynomial Degree {degree}")
    print("Intercept:", model.intercept_)
    for i, coef in enumerate(model.coef_):
        print(f"x^{i}: {coef:.4f}")

# Convert results to dataframe
results_df = pd.DataFrame(results)
print("\nModel Performance Summary")
print(results_df[["Degree", "MSE", "MAE", "R2"]])
```

```
Polynomial Degree 1
Intercept: -1.2555889536901006
x^0: 0.0000
x^1: 3.0521

Polynomial Degree 3
Intercept: 2.5318007492252086
x^0: 0.0000
x^1: 0.1944
x^2: -1.1537
x^3: 0.4978

Polynomial Degree 13
Intercept: 2.034565317696061
x^0: 0.0000
x^1: -3.5659
x^2: 0.8056
x^3: 11.4449
x^4: -1.5138
x^5: -9.8695
x^6: 0.3932
x^7: 3.9528
x^8: -0.0272
x^9: -0.7787
x^10: -0.0021
x^11: 0.0738
x^12: 0.0002
x^13: -0.0027

Model Performance Summary
   Degree        MSE       MAE        R2
0       1  39.476836  5.097990  0.430638
1       3  25.005302  3.992331  0.639356
2      13  24.379374  3.912345  0.648384
```

In [1]:
```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("yasserh/breast-cancer-dataset")

print("Path to dataset files:", path)
```

```
c:\Users\adity\OneDrive\Documents\GitHub\MLLab\.venv\Lib\site-packages\tqdm\auto.
py:21: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. Se
e https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
Downloading to C:\Users\adity\.cache\kagglehub\datasets\yasserh\breast-cancer-dat
aset\1.archive...
100%|██████████| 48.6k/48.6k [00:00<00:00, 216kB/s]
Extracting files...
Path to dataset files: C:\Users\adity\.cache\kagglehub\datasets\yasserh\breast-ca
ncer-dataset\versions\1
```

Perform logistic regression on breast cancer data

- Find the Co-relation matrix
- Accuracy
- Precision
- Recall/sensitivity
- F1 Score
- False positive Rate
- True Positive rate
- ROC curve

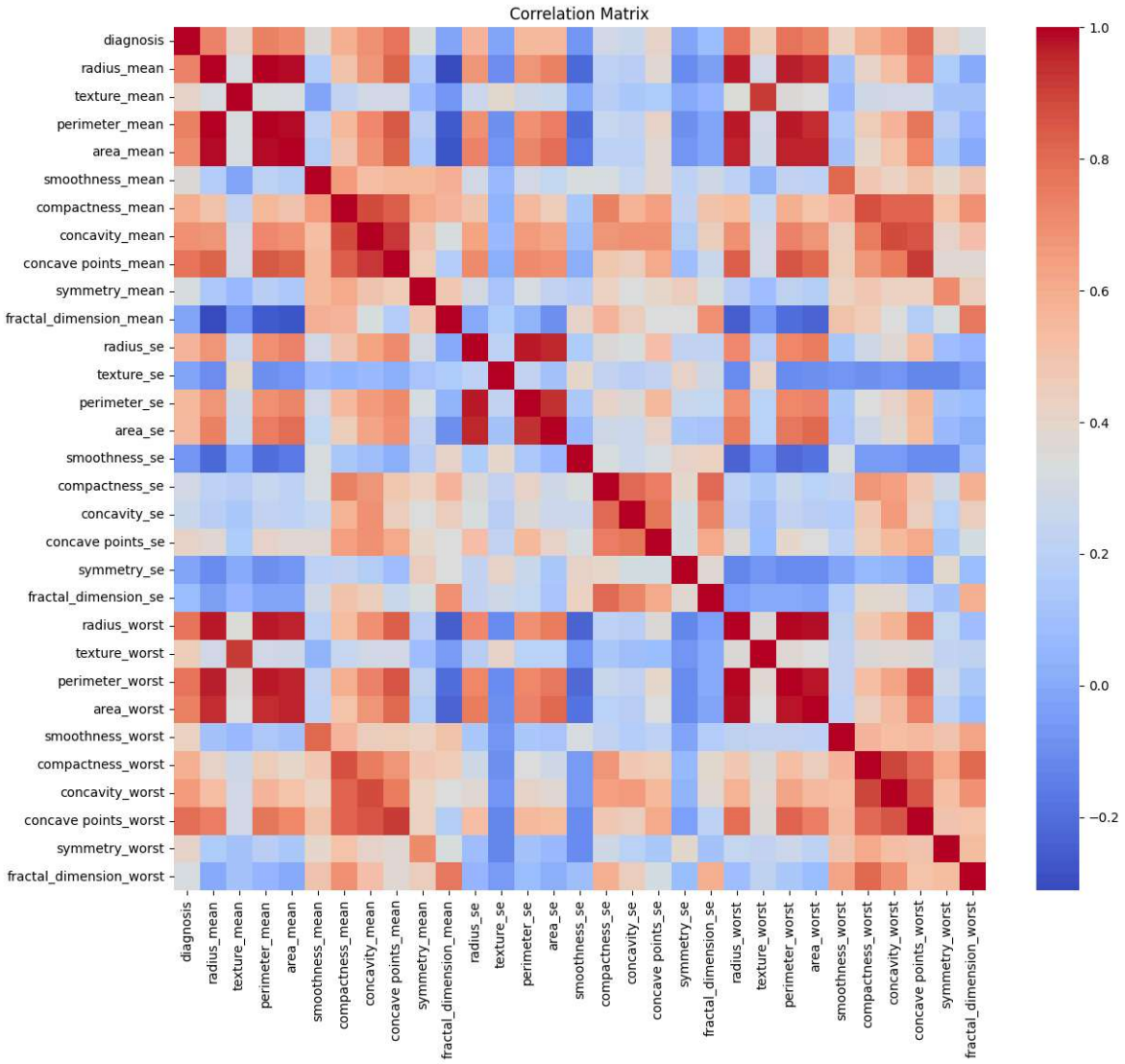
In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix,
    roc_curve,
    auc
)

# Load dataset
df = pd.read_csv(path + "/breast-cancer.csv")
```

In [3]:
```
df.columns
```

Out[3]:
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='str')
```

In [4]:
```
# Drop ID column (not useful for ML)
df.drop("id", axis=1, inplace=True)

# Encode target variable
df["diagnosis"] = df["diagnosis"].map({"M": 1, "B": 0})

# Split features and target
X = df.drop("diagnosis", axis=1)
y = df["diagnosis"]
```

In [5]:
```
# Correlation Matrix

plt.figure(figsize=(14, 12))
sns.heatmap(df.corr(), cmap="coolwarm")
plt.title("Correlation Matrix")
plt.show()
```



In [6]:
```python
# Train/Test Split

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

# Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [7]:
```python
# Logistic Regression

model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:, 1]
```

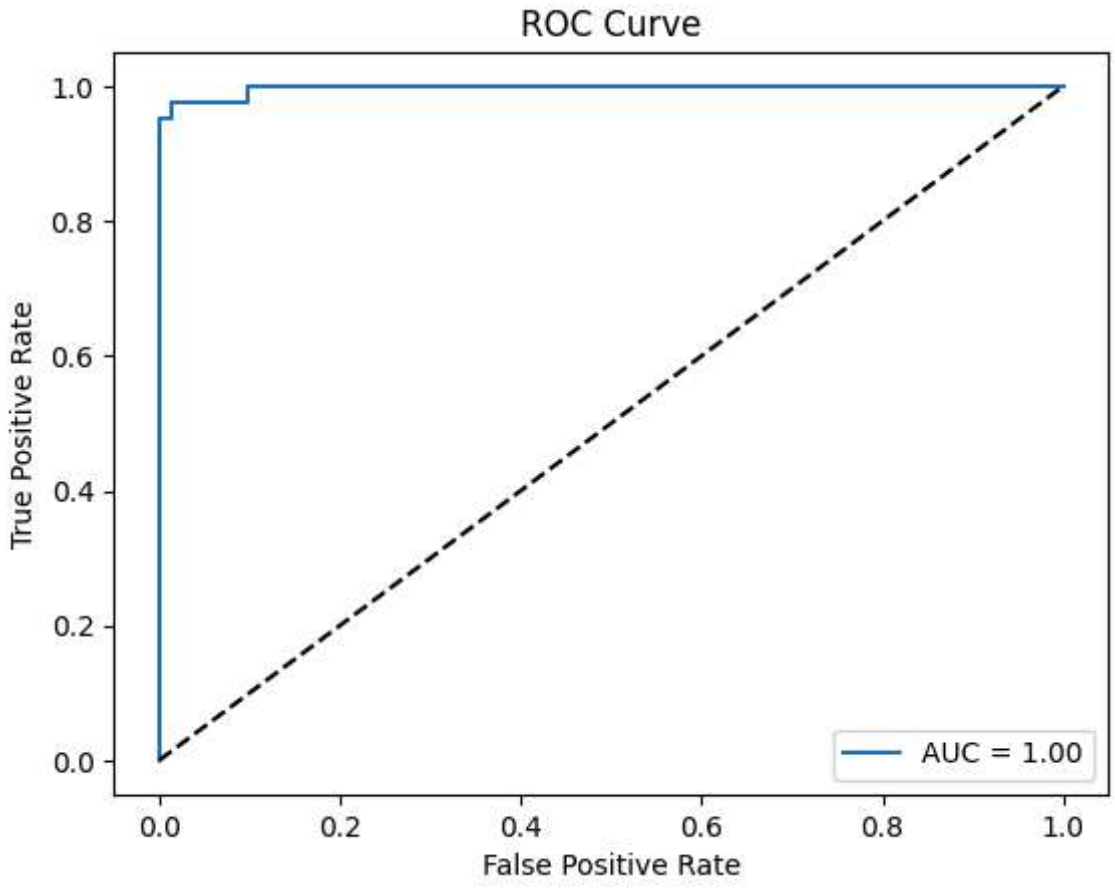
In [8]:
```python
# Metrics

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)       # Sensitivity
```

```
f1 = f1_score(y_test, y_pred)

cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()

false_positive_rate = fp / (fp + tn)
true_positive_rate = tp / (tp + fn)
```

In [9]:
```
# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
```

In [10]:
```
# Output

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall or Sensitivity:", recall)
print("F1 Score:", f1)
print("False Positive Rate:", false_positive_rate)
print("True Positive Rate:", true_positive_rate)
print("AUC:", roc_auc)

# ROC Plot
plt.figure()
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.2f}")
plt.plot([0,1], [0,1], "k--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```

```
Accuracy: 0.9736842105263158
Precision: 0.9761904761904762
Recall or Sensitivity: 0.9534883720930233
F1 Score: 0.9647058823529412
False Positive Rate: 0.014084507042253521
True Positive Rate: 0.9534883720930233
AUC: 0.99737962659679
```


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 1.00

Perform logistic regression on Heart disease data

- Find the Co-relation matrix
- Accuracy
- Precision
- Recall/sensitivity
- F1 Score
- False positive Rate
- True Positive rate
- ROC curve

In [1]:
```
import kagglehub
from kagglehub import KaggleDatasetAdapter

file_path = "heart.csv"

# Load the latest version
df = kagglehub.load_dataset(
    KaggleDatasetAdapter.PANDAS,
    "johnsmith88/heart-disease-dataset",
    file_path,
)

print(f"Dataset shape: {df.shape}")
print("First 5 records:")
print(df.head())
```

```
c:\Users\adity\OneDrive\Documents\GitHub\MLLab\.venv\Lib\site-packages\tqdm\auto.
py:21: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. Se
e https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
C:\Users\adity\AppData\Local\Temp\ipykernel_11452\2879929043.py:7: DeprecationWar
ning: Use dataset_load() instead of load_dataset(). load_dataset() will be remove
d in a future version.
  df = kagglehub.load_dataset(
Downloading to C:\Users\adity\.cache\kagglehub\datasets\johnsmith88\heart-disease
-dataset\versions\2\heart.csv...
100%|██████████| 37.2k/37.2k [00:00<00:00, 137kB/s]
```

```
Dataset shape: (1025, 14)
First 5 records:
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   52    1   0       125   212    0        1      168      0      1.0      2
1   53    1   0       140   203    1        0      155      1      3.1      0
2   70    1   0       145   174    0        1      125      1      2.6      0
3   61    1   0       148   203    0        1      161      0      0.0      2
4   62    0   0       138   294    1        1      106      0      1.9      1

   ca  thal  target
0   2     3       0
1   0     3       0
2   0     3       0
3   1     3       0
4   3     2       0
```

In [2]:
```python
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
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix,
    roc_curve,
    auc
)
```

Helper functions

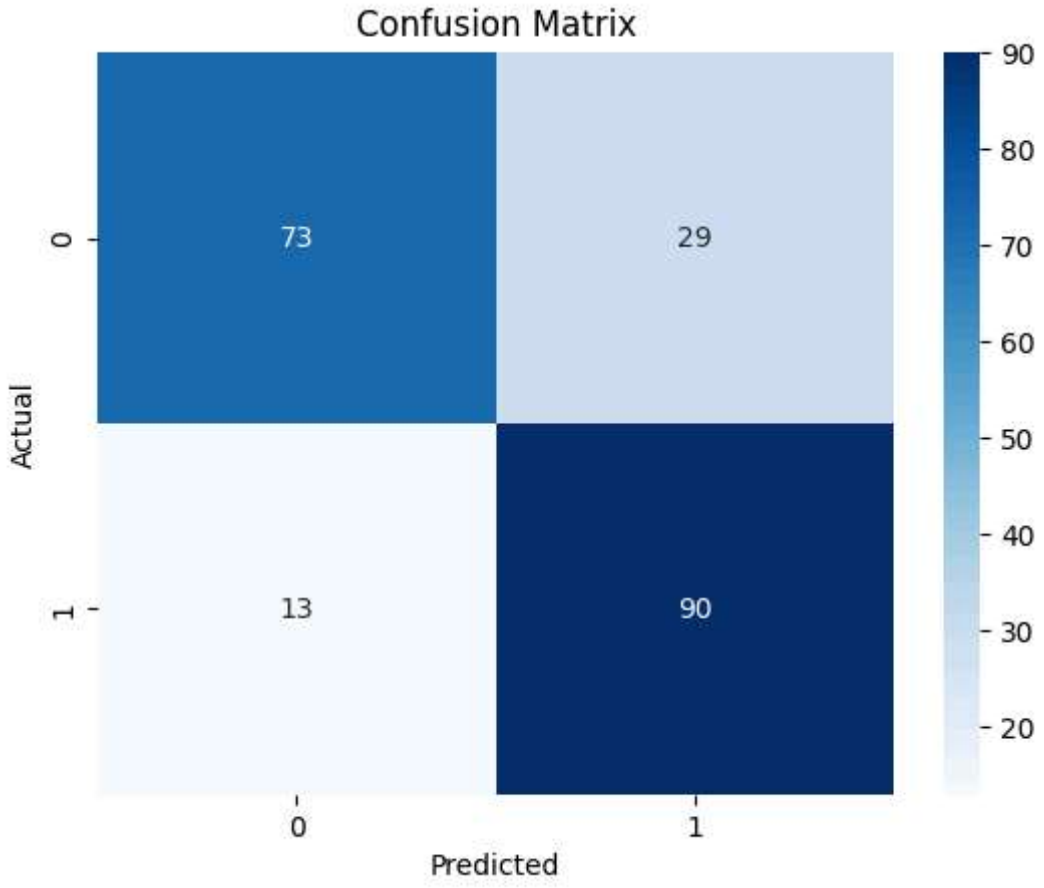
In [3]:
```python
# Find accuracy, precision, recall, f1-score
def evaluate_model(y_true, y_pred):
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    return accuracy, precision, recall, f1
# Plot confusion matrix
def plot_confusion_matrix(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()
# Plot ROC curve
def plot_roc_curve(y_true, y_scores):
    fpr, tpr, _ = roc_curve(y_true, y_scores)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], 'k--')
```

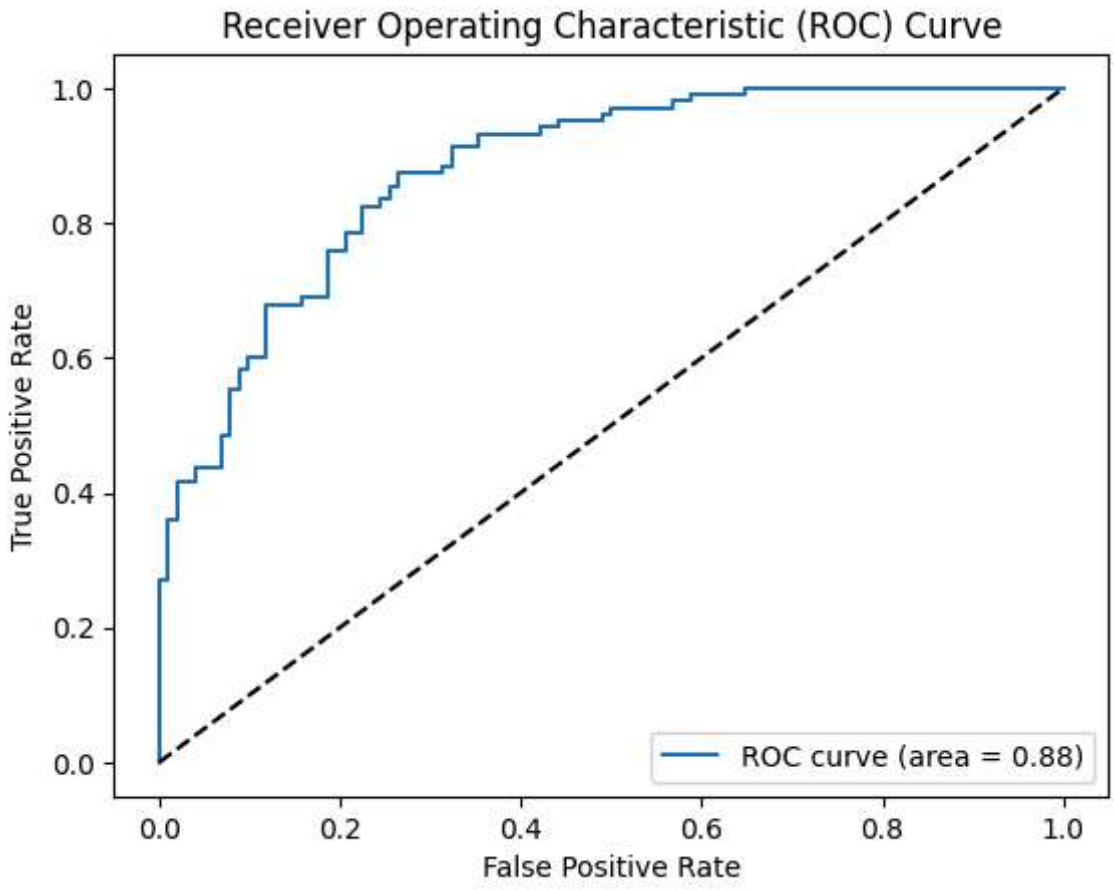
```
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.show()

# Preprocess data
X = df.drop('target', axis=1)
y = df['target']
```

In [4]:

```
# Perform train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Perform XG Boost
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_scores = model.predict_proba(X_test)[:, 1]

# Evaluate model
accuracy, precision, recall, f1 = evaluate_model(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
# Plot confusion matrix
plot_confusion_matrix(y_test, y_pred)
# Plot ROC curve
plot_roc_curve(y_test, y_scores)
```

```
Accuracy: 0.80
Precision: 0.76
Recall: 0.87
F1-Score: 0.81
```


Confusion Matrix
73
29
13
90
0
1
Actual
0
1
Predicted
90
80
70
60
50
40
30
20


Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.88)

Experiment 8

Compare the models SVM , decision tree , random forest and xgboost by tuning their hyperparameters using gridsearchCV

Steps:

1. Load Dataset
2. Preprocess the data
3. Split Data
4. Define Hyperparameter grid

- f1_estimators : [50,100,200]
- max_depth : [None, 10, 20, 30]
- min_samples_split: [2,5,10]
- min_samples_leaf: [1,2,4]

5. Perform grid search: Use GridSearchCV with cv=5
6. Train and evaluate: Asess performance on the test set
7. Report results: Print the best parameneters and test accuracy

In [1]:
```
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, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_mat
import warnings
warnings.filterwarnings('ignore')
```

Step 1: Load Dataset

In [2]:
```
import kagglehub
from kagglehub import KaggleDatasetAdapter

file_path = "heart.csv"

# Load the latest version
df = kagglehub.load_dataset(
    KaggleDatasetAdapter.PANDAS,
    "johnsmith88/heart-disease-dataset",
    file_path,
)
```

```
print(f"Dataset shape: {df.shape}")
print("\nFirst 5 records:")
print(df.head())
print("\nDataset info:")
print(df.info())
print("\nTarget distribution:")
print(df['target'].value_counts())
```

```
Dataset shape: (1025, 14)

First 5 records:
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   52    1   0       125   212    0        1      168      0      1.0      2
1   53    1   0       140   203    1        0      155      1      3.1      0
2   70    1   0       145   174    0        1      125      1      2.6      0
3   61    1   0       148   203    0        1      161      0      0.0      2
4   62    0   0       138   294    1        1      106      0      1.9      1

   ca  thal  target
0   2     3       0
1   0     3       0
2   0     3       0
3   1     3       0
4   3     2       0

Dataset info:
<class 'pandas.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1025 non-null   int64
 1   sex       1025 non-null   int64
 2   cp        1025 non-null   int64
 3   trestbps  1025 non-null   int64
 4   chol      1025 non-null   int64
 5   fbs       1025 non-null   int64
 6   restecg   1025 non-null   int64
 7   thalach   1025 non-null   int64
 8   exang     1025 non-null   int64
 9   oldpeak   1025 non-null   float64
 10  slope     1025 non-null   int64
 11  ca        1025 non-null   int64
 12  thal      1025 non-null   int64
 13  target    1025 non-null   int64
dtypes: float64(1), int64(13)
memory usage: 112.2 KB
None

Target distribution:
target
1    526
0    499
Name: count, dtype: int64
```

Step 2: Preprocess the data

In [3]:
```
# Check for missing values
print("Missing values:")
```

```
print(df.isnull().sum())

# Separate features and target
X = df.drop('target', axis=1)
y = df['target']

print(f"\nFeatures shape: {X.shape}")
print(f"Target shape: {y.shape}")
```

```
Missing values:
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64

Features shape: (1025, 13)
Target shape: (1025,)
```

Step 3: Split Data

In [4]:

```
# 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, stratify=y
)

# Standardize the features (important for SVM)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print(f"Training set shape: {X_train.shape}")
print(f"Test set shape: {X_test.shape}")
print(f"\nTraining target distribution:")
print(y_train.value_counts())
print(f"\nTest target distribution:")
print(y_test.value_counts())
```

```
Training set shape: (820, 13)
Test set shape: (205, 13)

Training target distribution:
target
1    421
0    399
Name: count, dtype: int64

Test target distribution:
target
1    105
0    100
Name: count, dtype: int64
```

Step 4: Define Hyperparameter Grids

In [5]:
```python
# Define hyperparameter grids for each model

# SVM hyperparameters
svm_param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['rbf', 'linear'],
    'gamma': ['scale', 'auto', 0.001, 0.01]
}

# Decision Tree hyperparameters
dt_param_grid = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}

# Random Forest hyperparameters
rf_param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

print("Hyperparameter grids defined successfully!")
print("\nSVM parameters:", list(svm_param_grid.keys()))
print("Decision Tree parameters:", list(dt_param_grid.keys()))
print("Random Forest parameters:", list(rf_param_grid.keys()))
```

```
Hyperparameter grids defined successfully!

SVM parameters: ['C', 'kernel', 'gamma']
Decision Tree parameters: ['max_depth', 'min_samples_split', 'min_samples_leaf',
'criterion']
Random Forest parameters: ['n_estimators', 'max_depth', 'min_samples_split', 'min
_samples_leaf']
```

Step 5 & 6: Perform Grid Search and Train Models

In [6]:
```
# Dictionary to store results
results = {}

print("=" * 60)
print("MODEL COMPARISON WITH GRIDSEARCHCV")
print("=" * 60)
```

```
============================================================
MODEL COMPARISON WITH GRIDSEARCHCV
============================================================
```

SVM with GridSearchCV

In [7]:
```
print("\n" + "=" * 60)
print("1. SUPPORT VECTOR MACHINE (SVM)")
print("=" * 60)

# Initialize SVM
svm = SVC(random_state=42)

# Perform Grid Search
svm_grid = GridSearchCV(
    svm,
    svm_param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)

# Fit on scaled data (SVM requires scaling)
svm_grid.fit(X_train_scaled, y_train)

# Make predictions
svm_pred = svm_grid.predict(X_test_scaled)
svm_accuracy = accuracy_score(y_test, svm_pred)

# Store results
results['SVM'] = {
    'best_params': svm_grid.best_params_,
    'best_cv_score': svm_grid.best_score_,
    'test_accuracy': svm_accuracy
}

print(f"\nBest Parameters: {svm_grid.best_params_}")
print(f"Best CV Accuracy: {svm_grid.best_score_:.4f}")
print(f"Test Accuracy: {svm_accuracy:.4f}")
```

```
============================================================
1. SUPPORT VECTOR MACHINE (SVM)
============================================================
Fitting 5 folds for each of 32 candidates, totalling 160 fits

Best Parameters: {'C': 100, 'gamma': 'scale', 'kernel': 'rbf'}
Best CV Accuracy: 0.9793
Test Accuracy: 1.0000
```

Decision Tree with GridSearchCV

In [8]:
```
print("\n" + "=" * 60)
print("2. DECISION TREE")
print("=" * 60)

# Initialize Decision Tree
dt = DecisionTreeClassifier(random_state=42)

# Perform Grid Search
dt_grid = GridSearchCV(
    dt,
    dt_param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)

# Fit on original data (Decision trees don't require scaling)
dt_grid.fit(X_train, y_train)

# Make predictions
dt_pred = dt_grid.predict(X_test)
dt_accuracy = accuracy_score(y_test, dt_pred)

# Store results
results['Decision Tree'] = {
    'best_params': dt_grid.best_params_,
    'best_cv_score': dt_grid.best_score_,
    'test_accuracy': dt_accuracy
}

print(f"\nBest Parameters: {dt_grid.best_params_}")
print(f"Best CV Accuracy: {dt_grid.best_score_:.4f}")
print(f"Test Accuracy: {dt_accuracy:.4f}")
```

```
============================================================
2. DECISION TREE
============================================================
Fitting 5 folds for each of 72 candidates, totalling 360 fits

Best Parameters: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf':
1, 'min_samples_split': 2}
Best CV Accuracy: 0.9890
Test Accuracy: 1.0000
```

Random Forest with GridSearchCV

In [9]:
```
print("\n" + "=" * 60)
print("3. RANDOM FOREST")
print("=" * 60)

# Initialize Random Forest
rf = RandomForestClassifier(random_state=42)

# Perform Grid Search
rf_grid = GridSearchCV(
    rf,
    rf_param_grid,
    cv=5,
```

```
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)

# Fit on original data
rf_grid.fit(X_train, y_train)

# Make predictions
rf_pred = rf_grid.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_pred)

# Store results
results['Random Forest'] = {
    'best_params': rf_grid.best_params_,
    'best_cv_score': rf_grid.best_score_,
    'test_accuracy': rf_accuracy
}

print(f"\nBest Parameters: {rf_grid.best_params_}")
print(f"Best CV Accuracy: {rf_grid.best_score_:.4f}")
print(f"Test Accuracy: {rf_accuracy:.4f}")
```

```
============================================================
3. RANDOM FOREST
============================================================
Fitting 5 folds for each of 108 candidates, totalling 540 fits

Best Parameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 200}
Best CV Accuracy: 0.9854
Test Accuracy: 1.0000
```

XGBoost with GridSearchCV

In [10]:

```
print("\n" + "=" * 60)
print("4. XGBOOST")
print("=" * 60)

try:
    import xgboost as xgb

    # XGBoost hyperparameters
    xgb_param_grid = {
        'n_estimators': [50, 100, 200],
        'max_depth': [3, 6, 10],
        'learning_rate': [0.01, 0.1, 0.3],
        'subsample': [0.8, 1.0]
    }

    # Initialize XGBoost
    xgb_model = xgb.XGBClassifier(random_state=42, eval_metric='logloss')

    # Perform Grid Search
    xgb_grid = GridSearchCV(
        xgb_model,
        xgb_param_grid,
        cv=5,
        scoring='accuracy',
```

```
        n_jobs=-1,
        verbose=1
    )

    # Fit on original data
    xgb_grid.fit(X_train, y_train)

    # Make predictions
    xgb_pred = xgb_grid.predict(X_test)
    xgb_accuracy = accuracy_score(y_test, xgb_pred)

    # Store results
    results['XGBoost'] = {
        'best_params': xgb_grid.best_params_,
        'best_cv_score': xgb_grid.best_score_,
        'test_accuracy': xgb_accuracy
    }

    print(f"\nBest Parameters: {xgb_grid.best_params_}")
    print(f"Best CV Accuracy: {xgb_grid.best_score_:.4f}")
    print(f"Test Accuracy: {xgb_accuracy:.4f}")

except ImportError:
    print("\nXGBoost not installed. Installing...")
    import subprocess
    import sys
    subprocess.check_call([sys.executable, "-m", "pip", "install", "xgboost"])
    print("Please restart the kernel and run this cell again.")
```

```
============================================================
4. XGBOOST
============================================================

XGBoost not installed. Installing...
Please restart the kernel and run this cell again.
```

Step 7: Report Results

In [11]:

```
print("\n" + "=" * 80)
print("FINAL RESULTS SUMMARY")
print("=" * 80)

# Create a summary DataFrame
summary_data = []
for model_name, result in results.items():
    summary_data.append({
        'Model': model_name,
        'Best CV Score': f"{result['best_cv_score']:.4f}",
        'Test Accuracy': f"{result['test_accuracy']:.4f}",
        'Best Parameters': str(result['best_params'])
    })

summary_df = pd.DataFrame(summary_data)
print("\n")
print(summary_df.to_string(index=False))

# Find best model
best_model = max(results.items(), key=lambda x: x[1]['test_accuracy'])
print("\n" + "=" * 80)
```

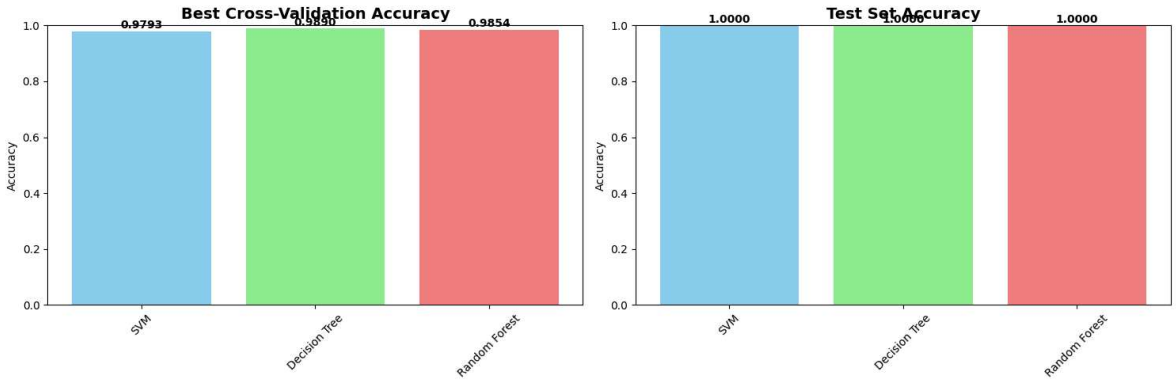
```
print(f"BEST MODEL: {best_model[0]}")
print(f"Test Accuracy: {best_model[1]['test_accuracy']:.4f}")
print("=" * 80)
```

```
================================================================================
FINAL RESULTS SUMMARY
================================================================================


        Model Best CV Score Test Accuracy
Best Parameters
          SVM        0.9793        1.0000
{'C': 100, 'gamma': 'scale', 'kernel': 'rbf'}
Decision Tree        0.9890        1.0000 {'criterion': 'entropy', 'max_depth': N
one, 'min_samples_leaf': 1, 'min_samples_split': 2}
Random Forest        0.9854        1.0000      {'max_depth': 10, 'min_samples_lea
f': 1, 'min_samples_split': 2, 'n_estimators': 200}

================================================================================
BEST MODEL: SVM
Test Accuracy: 1.0000
================================================================================
```

Visualization of Results

In [12]:

```
# Create visualization
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

models = list(results.keys())
cv_scores = [results[model]['best_cv_score'] for model in models]
test_scores = [results[model]['test_accuracy'] for model in models]

# Plot 1: Cross-Validation Scores
axes[0].bar(models, cv_scores, color=['skyblue', 'lightgreen', 'lightcoral', 'go
axes[0].set_title('Best Cross-Validation Accuracy', fontsize=14, fontweight='bol
axes[0].set_ylabel('Accuracy')
axes[0].set_ylim(0, 1)
for i, v in enumerate(cv_scores):
    axes[0].text(i, v + 0.01, f'{v:.4f}', ha='center', fontweight='bold')
axes[0].tick_params(axis='x', rotation=45)

# Plot 2: Test Accuracy
axes[1].bar(models, test_scores, color=['skyblue', 'lightgreen', 'lightcoral', '
axes[1].set_title('Test Set Accuracy', fontsize=14, fontweight='bold')
axes[1].set_ylabel('Accuracy')
axes[1].set_ylim(0, 1)
for i, v in enumerate(test_scores):
    axes[1].text(i, v + 0.01, f'{v:.4f}', ha='center', fontweight='bold')
axes[1].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()
```



Detailed Classification Reports

In [13]:
```
# Print detailed classification reports
print("\n" + "=" * 80)
print("DETAILED CLASSIFICATION REPORTS")
print("=" * 80)

predictions = {
    'SVM': svm_pred,
    'Decision Tree': dt_pred,
    'Random Forest': rf_pred
}

if 'XGBoost' in results:
    predictions['XGBoost'] = xgb_pred

for model_name, y_pred in predictions.items():
    print(f"\n{model_name}:")
    print("-" * 40)
    print(classification_report(y_test, y_pred))
```

```
================================================================================
DETAILED CLASSIFICATION REPORTS
================================================================================

SVM:
----------------------------------------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       100
           1       1.00      1.00      1.00       105

    accuracy                           1.00       205
   macro avg       1.00      1.00      1.00       205
weighted avg       1.00      1.00      1.00       205


Decision Tree:
----------------------------------------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       100
           1       1.00      1.00      1.00       105

    accuracy                           1.00       205
   macro avg       1.00      1.00      1.00       205
weighted avg       1.00      1.00      1.00       205


Random Forest:
----------------------------------------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       100
           1       1.00      1.00      1.00       105

    accuracy                           1.00       205
   macro avg       1.00      1.00      1.00       205
weighted avg       1.00      1.00      1.00       205
```

Experiment 9: K-Means Clustering on Heart Disease Dataset

This notebook demonstrates K-Means clustering analysis on the Heart Disease dataset from UCI Machine Learning Repository.

Objectives:

- Load and pre-process the Heart Disease dataset
- Determine optimal number of clusters using Silhouette score
- Reduce dimensionality to 2D using PCA
- Visualize clustered data and compare with actual labels

Step 0: Import Required Libraries

Import necessary Python libraries for data manipulation, clustering, and visualization.

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
import urllib.request
import warnings

warnings.filterwarnings("ignore")

# Set random seed for reproducibility
np.random.seed(42)
```

Step 1: Load and Pre-process the Dataset

Load the Heart Disease dataset from UCI Repository. The dataset contains 14 attributes including:

- **Demographic**: age, sex
- **Symptoms**: cp (chest pain type), trestbps (resting blood pressure), chol (cholesterol)
- **Test results**: restecg, thalach, exang, oldpeak, slope, ca, thal
- **Target**: presence of heart disease (0-4, converted to binary)

In [3]:
```
print("=" * 60)
print("STEP 1: Loading and Pre-processing Heart Disease Dataset")
print("=" * 60)

# Download the Heart Disease dataset from UCI
```

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/p

try:
    # Download and load the dataset
    column_names = [
        "age",
        "sex",
        "cp",
        "trestbps",
        "chol",
        "fbs",
        "restecg",
        "thalach",
        "exang",
        "oldpeak",
        "slope",
        "ca",
        "thal",
        "target",
    ]

    df = pd.read_csv(url, names=column_names, na_values="?")
    print("Dataset downloaded successfully from UCI Repository!")
except Exception as e:
    print(f"Error downloading dataset: {e}")
    print("Creating sample dataset for demonstration...")
    # Create synthetic data if download fails
    np.random.seed(42)
    n_samples = 303
    df = pd.DataFrame(
        {
            "age": np.random.randint(29, 77, n_samples),
            "sex": np.random.randint(0, 2, n_samples),
            "cp": np.random.randint(0, 4, n_samples),
            "trestbps": np.random.randint(94, 200, n_samples),
            "chol": np.random.randint(126, 564, n_samples),
            "fbs": np.random.randint(0, 2, n_samples),
            "restecg": np.random.randint(0, 3, n_samples),
            "thalach": np.random.randint(71, 202, n_samples),
            "exang": np.random.randint(0, 2, n_samples),
            "oldpeak": np.random.uniform(0, 6.2, n_samples),
            "slope": np.random.randint(0, 3, n_samples),
            "ca": np.random.randint(0, 4, n_samples),
            "thal": np.random.randint(0, 4, n_samples),
            "target": np.random.randint(0, 2, n_samples),
        }
    )

# Handle missing values
df = df.dropna()

print(f"Dataset shape: {df.shape}")
print("\nFirst 5 records:")
print(df.head())
print("\nDataset info:")
print(df.info())
print("\nTarget distribution:")
print(df["target"].value_counts())
print(f"\nMissing values: {df.isnull().sum().sum()}")
```

```
============================================================
STEP 1: Loading and Pre-processing Heart Disease Dataset
============================================================
Dataset downloaded successfully from UCI Repository!
Dataset shape: (297, 14)

First 5 records:
    age  sex   cp  trestbps   chol  fbs  restecg  thalach  exang  oldpeak  \
0  63.0  1.0  1.0     145.0  233.0  1.0      2.0    150.0    0.0      2.3
1  67.0  1.0  4.0     160.0  286.0  0.0      2.0    108.0    1.0      1.5
2  67.0  1.0  4.0     120.0  229.0  0.0      2.0    129.0    1.0      2.6
3  37.0  1.0  3.0     130.0  250.0  0.0      0.0    187.0    0.0      3.5
4  41.0  0.0  2.0     130.0  204.0  0.0      2.0    172.0    0.0      1.4

   slope   ca  thal  target
0    3.0  0.0   6.0       0
1    2.0  3.0   3.0       2
2    2.0  2.0   7.0       1
3    3.0  0.0   3.0       0
4    1.0  0.0   3.0       0

Dataset info:
<class 'pandas.DataFrame'>
Index: 297 entries, 0 to 301
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       297 non-null    float64
 1   sex       297 non-null    float64
 2   cp        297 non-null    float64
 3   trestbps  297 non-null    float64
 4   chol      297 non-null    float64
 5   fbs       297 non-null    float64
 6   restecg   297 non-null    float64
 7   thalach   297 non-null    float64
 8   exang     297 non-null    float64
 9   oldpeak   297 non-null    float64
 10  slope     297 non-null    float64
 11  ca        297 non-null    float64
 12  thal      297 non-null    float64
 13  target    297 non-null    int64
dtypes: float64(13), int64(1)
memory usage: 34.8 KB
None

Target distribution:
target
0    160
1     54
2     35
3     35
4     13
Name: count, dtype: int64

Missing values: 0
```

Data Preparation

Separate features from target variable and standardize the features for clustering.

In [4]:
```
# Separate features and target (convert target to binary: 0 = no disease, 1 = di
X = df.drop("target", axis=1)
y = (df["target"] > 0).astype(int)  # Convert to binary classification

print(f"\nFeatures shape: {X.shape}")
print(f"Target distribution (binary):\n{y.value_counts()}")

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
print(f"\nFeatures standardized successfully!")
```

```
Features shape: (297, 13)
Target distribution (binary):
target
0    160
1    137
Name: count, dtype: int64

Features standardized successfully!
```

Step 2: Apply K-Means and Determine Optimal k

Use the **Silhouette Score** to find the optimal number of clusters. The silhouette score ranges from -1 to 1, where:

- **1**: Samples are well-clustered and far from neighboring clusters
- **0**: Samples are on or very close to the decision boundary
- **-1**: Samples may have been assigned to the wrong cluster

In [5]:
```
print("\n" + "=" * 60)
print("STEP 2: Finding Optimal Number of Clusters")
print("=" * 60)

# Test different values of k
k_range = range(2, 11)
silhouette_scores = []
inertias = []

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    cluster_labels = kmeans.fit_predict(X_scaled)

    # Calculate silhouette score
    silhouette_avg = silhouette_score(X_scaled, cluster_labels)
    silhouette_scores.append(silhouette_avg)
    inertias.append(kmeans.inertia_)

    print(
        f"k={k}: Silhouette Score = {silhouette_avg:.4f}, Inertia = {kmeans.iner
    )
```

```
# Find optimal k
optimal_k = k_range[np.argmax(silhouette_scores)]
print(f"\nOptimal number of clusters (k) = {optimal_k}")
print(f"Best Silhouette Score = {max(silhouette_scores):.4f}")

# Apply K-Means with optimal k
kmeans_optimal = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
cluster_labels = kmeans_optimal.fit_predict(X_scaled)

print(f"\nCluster distribution:")
unique, counts = np.unique(cluster_labels, return_counts=True)
for cluster, count in zip(unique, counts):
    print(f"  Cluster {cluster}: {count} samples")
```

```
============================================================
STEP 2: Finding Optimal Number of Clusters
============================================================
k=2: Silhouette Score = 0.1751, Inertia = 3193.45
k=3: Silhouette Score = 0.1298, Inertia = 2931.15
k=4: Silhouette Score = 0.1304, Inertia = 2743.77
k=5: Silhouette Score = 0.1127, Inertia = 2609.80
k=6: Silhouette Score = 0.1137, Inertia = 2507.28
k=7: Silhouette Score = 0.1013, Inertia = 2404.55
k=8: Silhouette Score = 0.1060, Inertia = 2318.45
k=9: Silhouette Score = 0.1000, Inertia = 2252.42
k=10: Silhouette Score = 0.1120, Inertia = 2206.82

Optimal number of clusters (k) = 2
Best Silhouette Score = 0.1751

Cluster distribution:
  Cluster 0: 184 samples
  Cluster 1: 113 samples
```

Step 3: Dimensionality Reduction using PCA

Apply **Principal Component Analysis (PCA)** to reduce the 13-dimensional feature space to 2 dimensions for visualization. PCA transforms the data to a new coordinate system where the greatest variance lies on the first coordinate (PC1), the second greatest variance on the second coordinate (PC2), and so on.

In [6]:
```
print("\n" + "=" * 60)
print("STEP 3: Dimensionality Reduction using PCA")
print("=" * 60)

# Apply PCA to reduce to 2 dimensions
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

print(f"Original dimensions: {X_scaled.shape[1]}")
print(f"Reduced dimensions: {X_pca.shape[1]}")
print(f"\nExplained Variance Ratio:")
print(
    f"  PC1: {pca.explained_variance_ratio_[0]:.4f} ({pca.explained_variance_rat
)
print(
    f"  PC2: {pca.explained_variance_ratio_[1]:.4f} ({pca.explained_variance_rat
)
```

```
print(
    f"  Total: {sum(pca.explained_variance_ratio_):.4f} ({sum(pca.explained_vari
)
```

```
============================================================
STEP 3: Dimensionality Reduction using PCA
============================================================
Original dimensions: 13
Reduced dimensions: 2

Explained Variance Ratio:
  PC1: 0.2370 (23.70%)
  PC2: 0.1235 (12.35%)
  Total: 0.3604 (36.04%)
```

Step 4: Visualize the Clusters in 2D

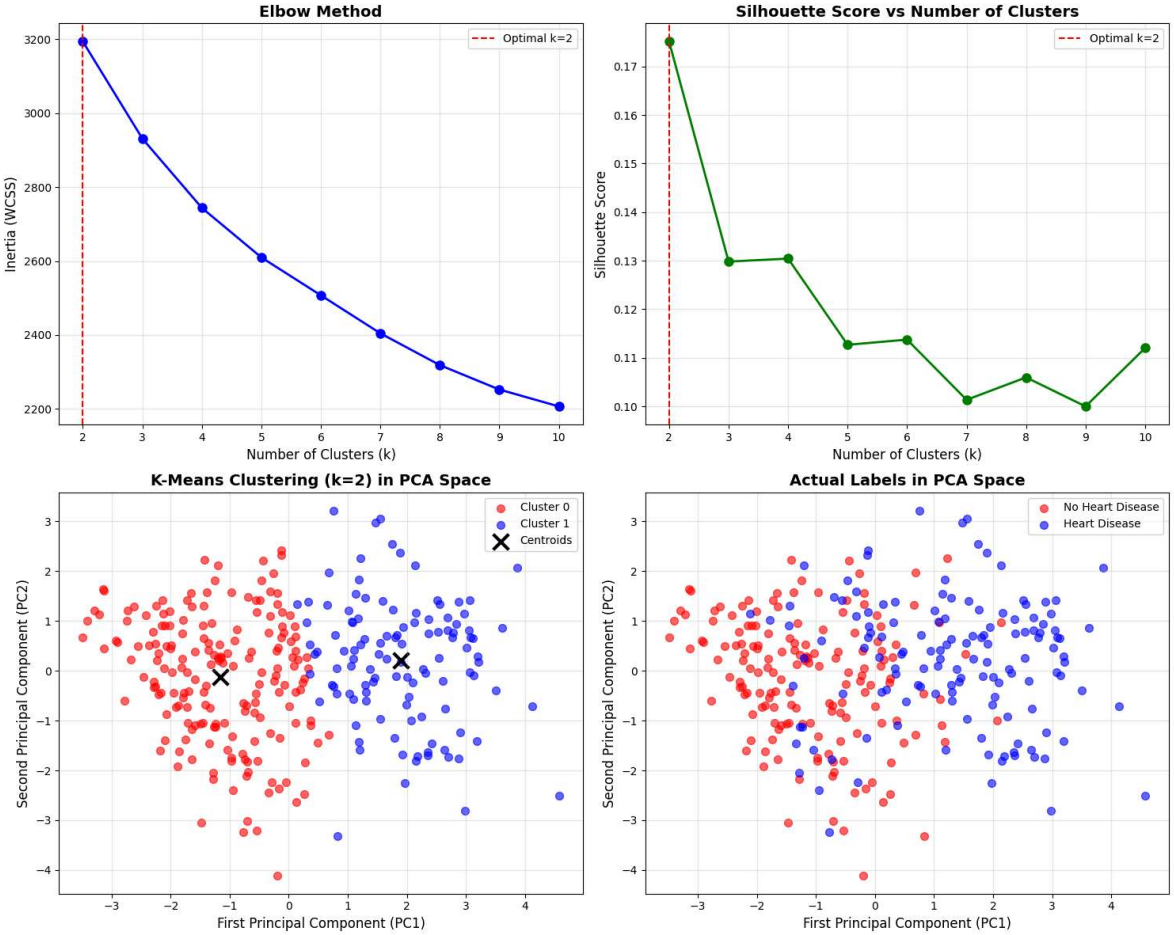
Create comprehensive visualizations including:

1. **Elbow Method**: Shows the optimal k using Within-Cluster Sum of Squares (WCSS)
2. **Silhouette Scores**: Plot of silhouette scores for different k values
3. **K-Means Clusters**: Clusters visualized in PCA space with centroids
4. **Actual Labels**: Ground truth labels for comparison

In [7]:
```
print("\n" + "=" * 60)
print("STEP 4: Visualizing Clusters")
print("=" * 60)

# Create a figure with multiple subplots
fig, axes = plt.subplots(2, 2, figsize=(15, 12))

# Plot 1: Elbow Method
axes[0, 0].plot(k_range, inertias, "bo-", linewidth=2, markersize=8)
axes[0, 0].set_xlabel("Number of Clusters (k)", fontsize=12)
axes[0, 0].set_ylabel("Inertia (WCSS)", fontsize=12)
axes[0, 0].set_title("Elbow Method", fontsize=14, fontweight="bold")
axes[0, 0].grid(True, alpha=0.3)
axes[0, 0].axvline(
    x=optimal_k, color="r", linestyle="--", label=f"Optimal k={optimal_k}"
)
axes[0, 0].legend()

# Plot 2: Silhouette Scores
axes[0, 1].plot(k_range, silhouette_scores, "go-", linewidth=2, markersize=8)
axes[0, 1].set_xlabel("Number of Clusters (k)", fontsize=12)
axes[0, 1].set_ylabel("Silhouette Score", fontsize=12)
axes[0, 1].set_title(
    "Silhouette Score vs Number of Clusters", fontsize=14, fontweight="bold"
)
axes[0, 1].grid(True, alpha=0.3)
axes[0, 1].axvline(
    x=optimal_k, color="r", linestyle="--", label=f"Optimal k={optimal_k}"
)
axes[0, 1].legend()

# Plot 3: K-Means Clusters in PCA Space
colors = [
    "red",
```

```
    "blue",
    "green",
    "orange",
    "purple",
    "brown",
    "pink",
    "gray",
    "olive",
    "cyan",
]
for i in range(optimal_k):
    mask = cluster_labels == i
    axes[1, 0].scatter(
        X_pca[mask, 0],
        X_pca[mask, 1],
        c=colors[i],
        label=f"Cluster {i}",
        alpha=0.6,
        s=50,
    )

# Plot cluster centers in PCA space
centers_pca = pca.transform(kmeans_optimal.cluster_centers_)
axes[1, 0].scatter(
    centers_pca[:, 0],
    centers_pca[:, 1],
    c="black",
    marker="x",
    s=200,
    linewidths=3,
    label="Centroids",
)
axes[1, 0].set_xlabel("First Principal Component (PC1)", fontsize=12)
axes[1, 0].set_ylabel("Second Principal Component (PC2)", fontsize=12)
axes[1, 0].set_title(
    f"K-Means Clustering (k={optimal_k}) in PCA Space", fontsize=14, fontweight=
)
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)

# Plot 4: Actual Target Labels in PCA Space
for i in [0, 1]:
    mask = y == i
    label = "No Heart Disease" if i == 0 else "Heart Disease"
    axes[1, 1].scatter(
        X_pca[mask, 0], X_pca[mask, 1], c=colors[i], label=label, alpha=0.6, s=5
    )
axes[1, 1].set_xlabel("First Principal Component (PC1)", fontsize=12)
axes[1, 1].set_ylabel("Second Principal Component (PC2)", fontsize=12)
axes[1, 1].set_title("Actual Labels in PCA Space", fontsize=14, fontweight="bold
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig("../assets/heart_disease_kmeans_clustering.png", dpi=150, bbox_inche
print("Visualization saved as '../assets/heart_disease_kmeans_clustering.png'")
plt.show()
```

```
============================================================
STEP 4: Visualizing Clusters
============================================================
Visualization saved as '../assets/heart_disease_kmeans_clustering.png'
```



Additional Analysis

Compare the discovered clusters with actual labels and analyze PCA components.

In [8]:
```
print("\n" + "=" * 60)
print("Additional Analysis")
print("=" * 60)

# Compare clusters with actual labels
print("\nCluster vs Actual Label Cross-tabulation:")
crosstab = pd.crosstab(cluster_labels, y, rownames=["Cluster"], colnames=["Actua
print(crosstab)

# Feature importance in PCA
print("\nPCA Component Analysis:")
feature_names = X.columns
components_df = pd.DataFrame(
    pca.components_.T, columns=["PC1", "PC2"], index=feature_names
)
print(components_df.round(4))

print("\n" + "=" * 60)
print("Experiment Complete!")
print("=" * 60)
```

```
============================================================
Additional Analysis
============================================================

Cluster vs Actual Label Cross-tabulation:
Actual     0    1
Cluster
0        147   37
1         13  100

PCA Component Analysis:
             PC1     PC2
age       0.2859 -0.4187
sex       0.1168  0.4316
cp        0.2862  0.1525
trestbps  0.1678 -0.3915
chol      0.0835 -0.4282
fbs       0.0761 -0.2399
restecg   0.1459 -0.2665
thalach  -0.3927 -0.0541
exang     0.3331  0.2083
oldpeak   0.3970  0.0617
slope     0.3520  0.0745
ca        0.3064 -0.1578
thal      0.3462  0.2634

============================================================
Experiment Complete!
============================================================
```