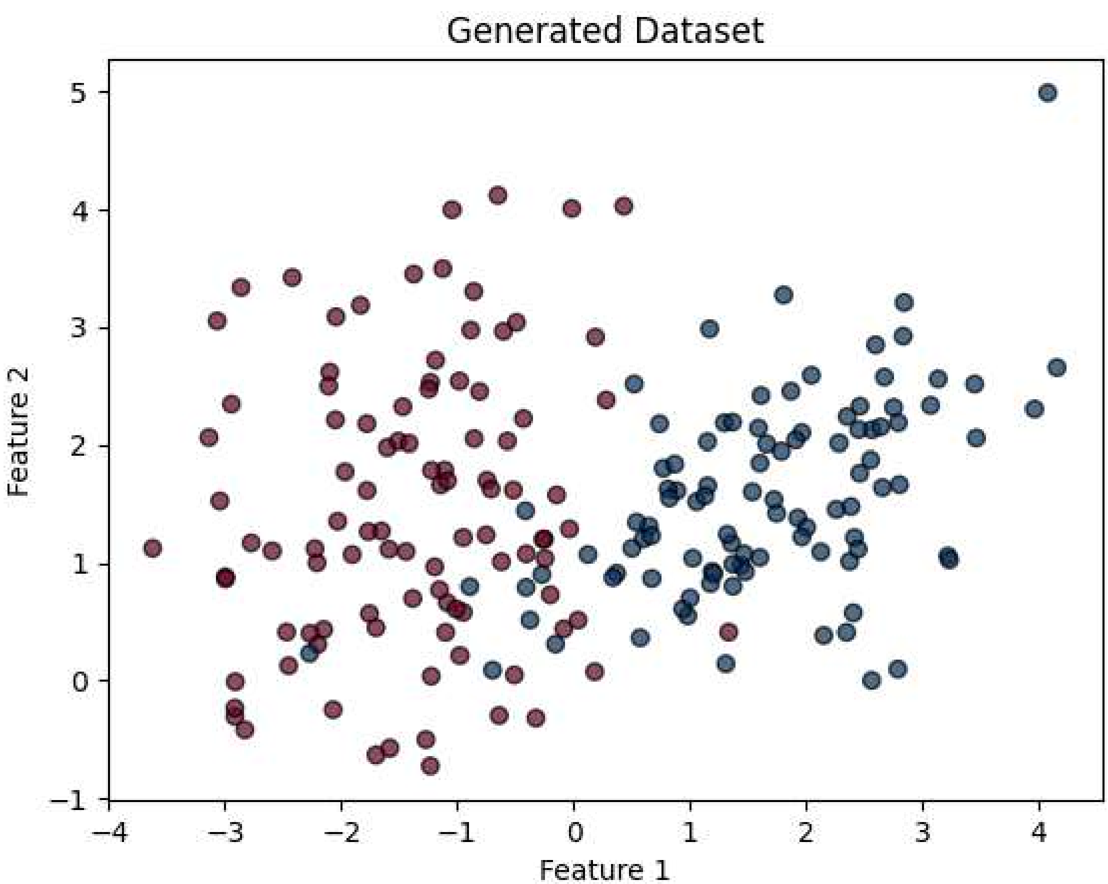
In [1]:
```
#The decision boundary separates the classes in a dataset based on probabilities. L
```

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression

# Generate synthetic dataset
np.random.seed(42)
X, y = make_classification(
    n_samples=200, n_features=2, n_informative=2,
    n_redundant=0, n_clusters_per_class=1, class_sep=1.5, random_state=42
)

# Plot the data points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='RdBu', edgecolor='k', alpha=0.7)
plt.title("Generated Dataset")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```



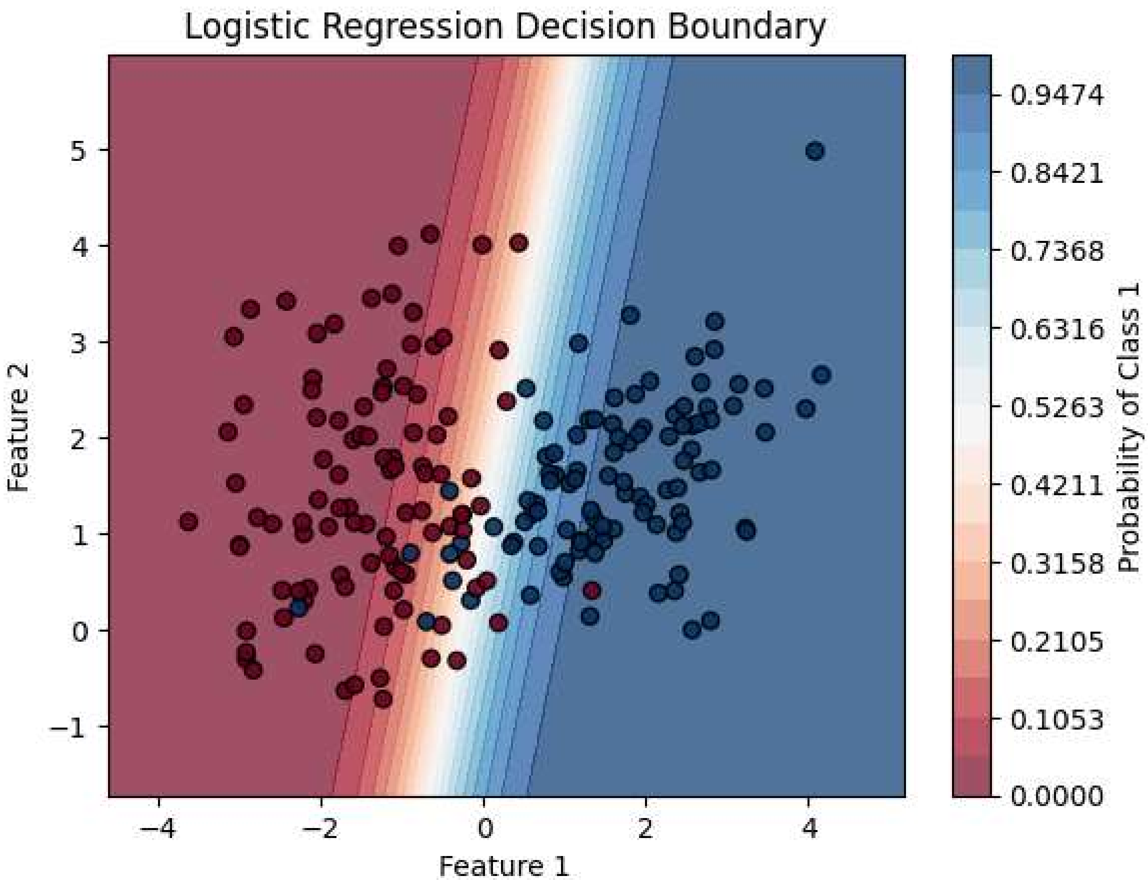
In [4]:
```
#ABOVE CONTAINS
#X contains 2D features for simplicity
#y contains the binary target (0 or 1).
#make_classification generates data that's separable enough for logistic regression
```

In [7]:
```
#We train the model to compute the decision boundary.
# Train logistic regression
model = LogisticRegression()
model.fit(X, y)

# Get model coefficients
weights = model.coef_[0]
bias = model.intercept_[0]
print(f"Weights: {weights}, Bias: {bias}")
```

```
Weights: [ 2.40955857 -0.56440271], Bias: 0.6243863390309786
```

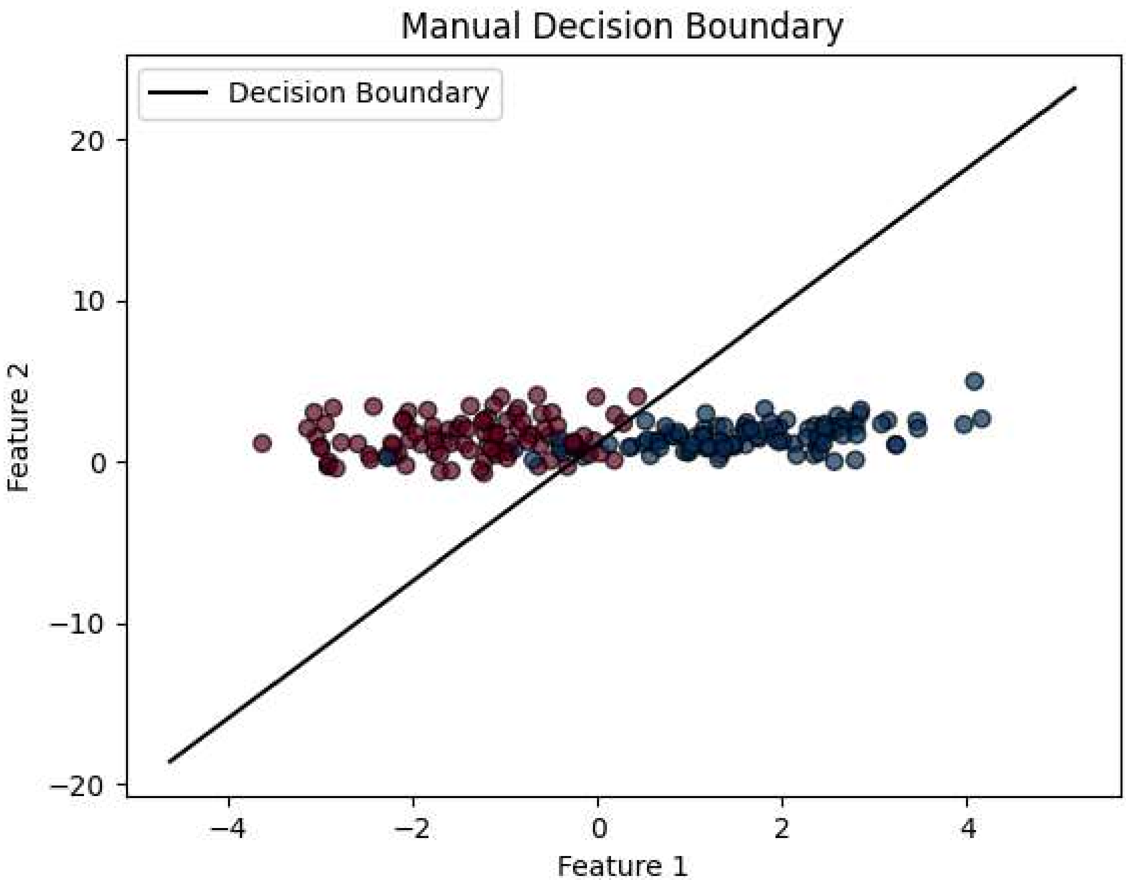
In [8]:
```
#DECISION BOUNDARY EQUATION IS W1*X1+W2*X2+B=0
```

In [10]:
```
#We use the learned parameters to draw the decision boundary
# Create a grid of points for visualization
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                     np.linspace(y_min, y_max, 100))

# Compute decision boundary
z = model.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]
z = z.reshape(xx.shape)

# Plot decision boundary
plt.contourf(xx, yy, z, levels=np.linspace(0, 1, 20), cmap='RdBu', alpha=0.7)
plt.colorbar(label="Probability of Class 1")
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', cmap='RdBu', alpha=0.9)
plt.title("Logistic Regression Decision Boundary")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

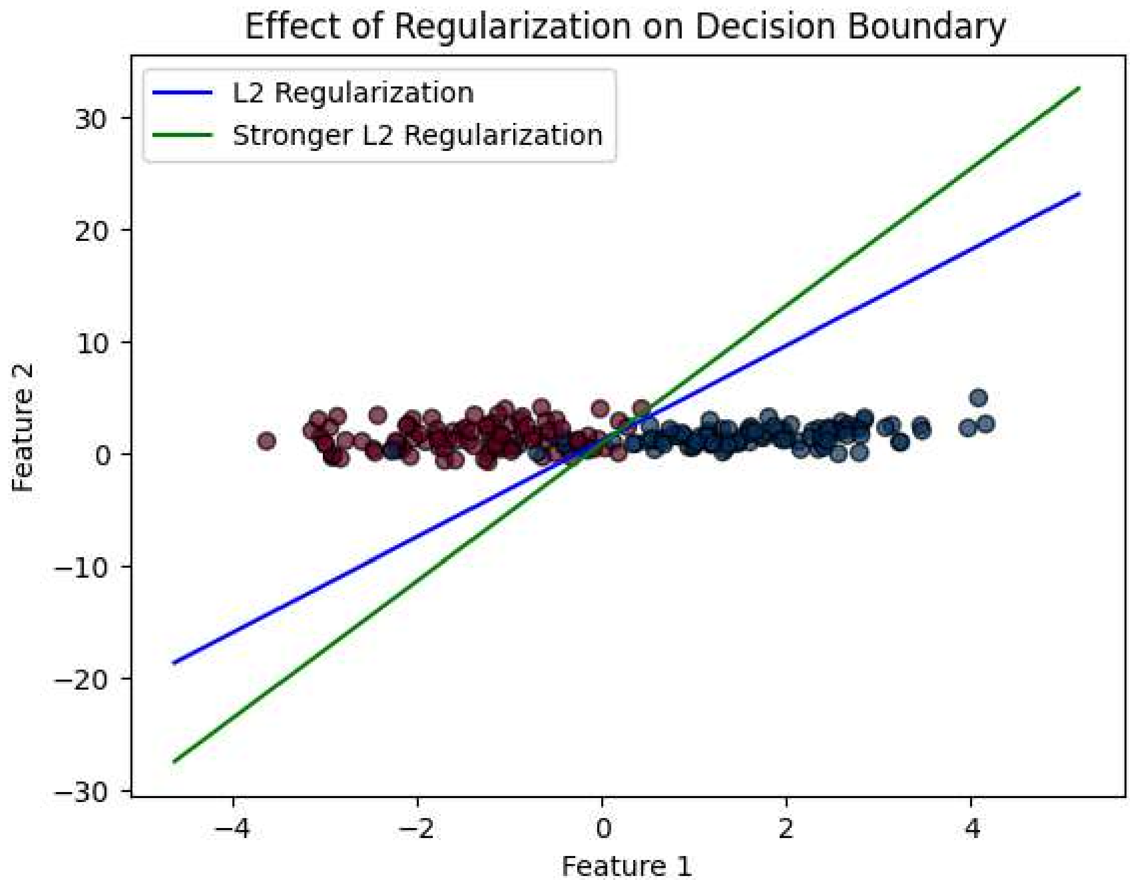


In [12]:

```
#Manually Calculating and Plotting the Decision Line
# Compute x2 for the decision boundary
x1_values = np.linspace(x_min, x_max, 100)
x2_values = -(weights[0] * x1_values + bias) / weights[1]

# Plot decision line
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='RdBu', edgecolor='k', alpha=0.7)
plt.plot(x1_values, x2_values, color='black', label="Decision Boundary")
plt.title("Manual Decision Boundary")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
```



In [14]:
```
#Regularization and Decision Boundary
#You can observe how regularization affects the decision boundary:
# Train with L2 regularization (default)
model_l2 = LogisticRegression(C=1.0)  # Higher C = less regularization
model_l2.fit(X, y)

# Train with stronger L2 regularization
model_strong_l2 = LogisticRegression(C=0.1)  # Lower C = stronger regularization
model_strong_l2.fit(X, y)

# Visualize decision boundaries
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='RdBu', edgecolor='k', alpha=0.7)
x2_l2 = -(model_l2.coef_[0][0] * x1_values + model_l2.intercept_[0]) / model_l2.coe
x2_strong_l2 = -(model_strong_l2.coef_[0][0] * x1_values + model_strong_l2.intercep

plt.plot(x1_values, x2_l2, color='blue', label="L2 Regularization")
plt.plot(x1_values, x2_strong_l2, color='green', label="Stronger L2 Regularization"
plt.title("Effect of Regularization on Decision Boundary")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
```



In [ ]:
```
#This code demonstrates how logistic regression creates a linear decision boundary.
#You see how the model separates the feature space into two regions.
#Adding regularization (L1/L2) modifies the decision boundary by penalizing the mag
```