```
pip install -U kaleido
```

```
Requirement already satisfied: kaleido in /usr/local/lib/python3.10/dist-packages (0.2.1)
```

Step 1: Import libraries

```
# Import libraries

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

import plotly.express as px
import plotly.graph_objs as go

from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix, accuracy_score


#import warnings
#warnings.filterwarnings('ignore')
```

Step 2: Generate synthetic data for two classes

Define bold text mean (mu) and covariance (Sigma) matrices for each class

```
mu1 = [-2, -2]
sigma1 = [[0.9, -0.0255], [-0.0255, 0.9]]
```

```
mu2 = [5, 5]
sigma2 = [[0.5, 0], [0, 0.3]]
```

Generate synthetic data by drawing samples from each distribution

```
num_samples_class1 = 250
num_samples_class2 = 250
```

For first class

```
np.random.seed(0)
points_1 = np.random.default_rng().multivariate_normal(mu1, sigma1, num_samples_class1)
```

```
points_1.shape
```

```
(250, 2)
```

```
pd.DataFrame(points_1, columns = ['Feature_1', 'Feature_2']).head()
```

	Feature_1	Feature_2
0	-0.219732	-1.024637
1	-1.184319	-0.386109
2	-4.875478	-1.230151
3	-3.723137	-2.446335
4	-1.693616	-1.447580

```
points_1.mean(axis = 0)
```

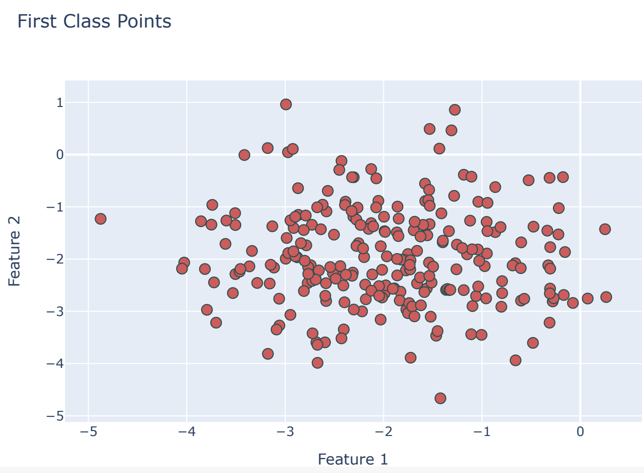
```
array([-1.96849744, -1.94048883])
```

```
np.cov(points_1.T)
```

```
array([[ 0.92553147, -0.05003727],
       [-0.05003727,  0.87691404]])
```

```
fig = px.scatter(x = points_1[:, 0], y = points_1[:, 1], width = 700, height = 500,
                        color_discrete_sequence = ['indianred'],
                        title = 'First Class Points',
                        labels = {'x' : 'Feature 1', 'y' : 'Feature 2'})

fig.update_traces(marker=dict(size = 10,
                              line = dict(width = 1,
                                          color = 'DarkSlateGrey')))
fig.show()
fig.write_image('First Class Points.png')
```




```
class1_labels = np.ones((num_samples_class1, 1), dtype = int)
print(f'Shape = {class1_labels.shape}\tType = {type(class1_labels)}\n')
pd.Series(class1_labels.flatten()).head()
```

```
Shape = (250, 1)        Type = <class 'numpy.ndarray'>

0    1
1    1
2    1
3    1
4    1
dtype: int64
```

For second class

```
np.random.seed(0)
points_2 = np.random.default_rng().multivariate_normal(mu2, sigma2, num_samples_class2)
```

```
points_2.shape
```

```
(250, 2)
```

```
pd.DataFrame(points_2, columns = ['Feature_1', 'Feature_2']).head()
```

	Feature_1	Feature_2
0	5.831511	4.966561
1	6.260290	5.010802
2	5.155045	4.889583
3	4.733564	4.658453
4	5.111328	5.290811

```
points_2.mean(axis=0)
```

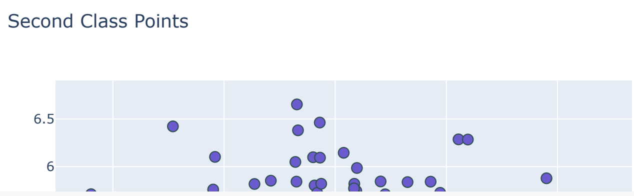
```
array([4.92393426, 4.98937958])
```

```
np.cov(points_2.T)
```

```
array([[5.70099905e-01, 8.65665736e-06],
       [8.65665736e-06, 3.02735184e-01]])
```

```
fig = px.scatter(x = points_2[:, 0], y = points_2[:, 1], width = 700, height = 500,
                        color_discrete_sequence = ['slateblue'],
                        title = 'Second Class Points',
                        labels = {'x' : 'Feature 1', 'y' : 'Feature 2'})

fig.update_traces(marker=dict(size = 10,
                              line = dict(width = 1,
                                          color = 'DarkSlateGrey')))
fig.show()
fig.write_image('Second Class Points.png')
```




```
class2_labels = np.zeros((num_samples_class2, 1), dtype = int)
print(f'Shape = {class2_labels.shape}\tType = {type(class2_labels)}\n')
pd.Series(class2_labels.flatten()).head()
```

```
Shape = (250, 1)        Type = <class 'numpy.ndarray'>

0    0
1    0
2    0
3    0
4    0
dtype: int64
```

Step 3: Combine both the distribution (classes) and their labels to form a dataset

```
class1_data = np.hstack((points_1, class1_labels))
print(f'Shape = {class1_data.shape}\n')
pd.DataFrame(class1_data, columns = ['Feature_1', 'Feature_2', 'Label']).head()
```

```
Shape = (250, 3)
```

	Feature_1	Feature_2	Label
0	-0.219732	-1.024637	1.0
1	-1.184319	-0.386109	1.0
2	-4.875478	-1.230151	1.0
3	-3.723137	-2.446335	1.0
4	-1.693616	-1.447580	1.0

```
class2_data = np.hstack((points_2, class2_labels))
print(f'Shape = {class2_data.shape}\n')
pd.DataFrame(class2_data, columns = ['Feature_1', 'Feature_2', 'Label']).head()
```

```
Shape = (250, 3)
```

	Feature_1	Feature_2	Label
0	5.831511	4.966561	0.0
1	6.260290	5.010802	0.0
2	5.155045	4.889583	0.0
3	4.733564	4.658453	0.0
4	5.111328	5.290811	0.0

```
dataset = np.vstack((class1_data, class2_data))
print(f'Shape = {dataset.shape}\n')
data_df = pd.DataFrame(dataset, columns = ['Feature_1', 'Feature_2', 'Label'])
data_df['Label'] = data_df['Label'].astype('int')
data_df['Label'] = data_df['Label'].astype('category')
data_df.head()
```

```
Shape = (500, 3)
```

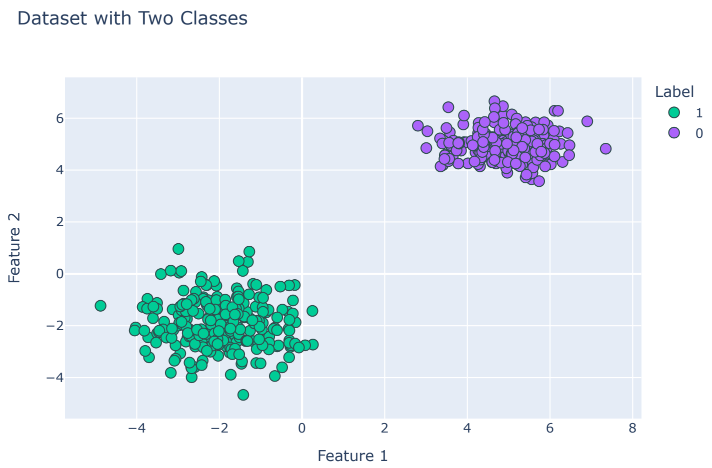
	Feature_1	Feature_2	Label
0	-0.219732	-1.024637	1
1	-1.184319	-0.386109	1
2	-4.875478	-1.230151	1
3	-3.723137	-2.446335	1
4	-1.693616	-1.447580	1

```
print('Labels = ', data_df['Label'].unique())
```

```
Labels =  [1, 0]
Categories (2, int64): [0, 1]
```

```
fig = px.scatter(data_df, x = 'Feature_1', y = 'Feature_2', width = 700, height = 500,
                        color = 'Label',
                        color_discrete_map={'1': 'indianred', '0': 'slateblue'},
                        title = 'Dataset with Two Classes',
                        labels={'Feature_1': 'Feature 1', 'Feature_2': 'Feature 2'})


fig.update_traces(marker=dict(size = 10,
                              line = dict(width = 1,
                                          color = 'DarkSlateGrey')))
fig.show()
fig.write_image('Dataset with Two Classes.png')
```



Step 4: Include bias term by adding a column of ones to input feature matrix.

```
bias = np.ones((num_samples_class1 + num_samples_class2, 1))
print(f'Shape = {bias.shape}\tType = {type(bias)}\n')
pd.Series(bias.flatten()).head()
```

```
Shape = (500, 1)        Type = <class 'numpy.ndarray'>

0    1.0
1    1.0
2    1.0
3    1.0
4    1.0
dtype: float64
```

```
input_feature_matrix = np.hstack((dataset, bias))
print(f'Shape = {input_feature_matrix.shape}\n')
pd.DataFrame(input_feature_matrix, columns = ['Feature_1', 'Feature_2', 'Label', 'Bias']).head()
```

```
Shape = (500, 4)
```

	Feature_1	Feature_2	Label	Bias
0	-0.219732	-1.024637	1.0	1.0
1	-1.184319	-0.386109	1.0	1.0
2	-4.875478	-1.230151	1.0	1.0
3	-3.723137	-2.446335	1.0	1.0
4	-1.693616	-1.447580	1.0	1.0

Step 5: Split the dataset into train and test.

```
y = input_feature_matrix[:, 2].astype('int')
print(f'Shape = {y.shape}\tType = {type(y)}\n')
pd.Series(y, name = 'Label').head()
```

```
Shape = (500,)  Type = <class 'numpy.ndarray'>

0    1
1    1
2    1
3    1
4    1
Name: Label, dtype: int64
```

```
X = np.delete(input_feature_matrix, 2, axis = 1)
print(f'Shape = {input_feature_matrix.shape}\n')
pd.DataFrame(X, columns = ['Feature_1', 'Feature_2', 'Bias']).head()
```

```
Shape = (500, 4)
```

	Feature_1	Feature_2	Bias
0	-0.219732	-1.024637	1.0
1	-1.184319	-0.386109	1.0
2	-4.875478	-1.230151	1.0
3	-3.723137	-2.446335	1.0

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 42)
```

```
print(f'Shape = {X_train.shape}\n')
pd.DataFrame(X_train, columns = ['Feature_1', 'Feature_2', 'Bias']).head()
```

```
Shape = (400, 3)
```

	Feature_1	Feature_2	Bias
0	-3.147075	-2.108397	1.0
1	4.913109	5.556402	1.0
2	-1.254478	-1.722929	1.0
3	5.944035	5.725453	1.0
4	4.833223	4.969331	1.0

```
print(f'Shape = {X_test.shape}\n')
pd.DataFrame(X_test, columns = ['Feature_1', 'Feature_2', 'Bias']).head()
```

```
Shape = (100, 3)
```

	Feature_1	Feature_2	Bias
0	4.693989	4.371286	1.0
1	-2.866492	-1.153692	1.0
2	5.572872	3.886709	1.0
3	-1.096525	-2.899893	1.0
4	-1.037154	-0.902838	1.0

```
print(f'Shape = {y_train.shape}\tType = {type(y_train)}\n')
pd.Series(y_train, name = 'Label').head()
```

```
Shape = (400,)  Type = <class 'numpy.ndarray'>

0    1
1    0
2    1
3    0
4    0
Name: Label, dtype: int64
```

```
print(f'Shape = {y_test.shape}\tType = {type(y_test)}\n')
pd.Series(y_test, name = 'Label').head()
```

```
Shape = (100,)  Type = <class 'numpy.ndarray'>

0    0
1    1
2    0
3    1
4    1
Name: Label, dtype: int64
```

Step 6: Write a function to train the perceptron that will take data, labels, learning rate and max_epochs as parameters.

```
def train_perceptron(data, labels, learning_rate, max_epochs):
    np.random.seed(0)
    weights = np.random.rand(data.shape[1] - 1, 1)
    bias = data[0, -1]

    for epoch in range(max_epochs):
        for i in range(len(data)):
            x_i = data[i, :-1]
            predicted_weighted_sum = np.dot(x_i, weights) + bias
            predicted_label = 1 if predicted_weighted_sum >= 0 else 0    # activation

            training_error = labels[i] - predicted_label
            weights += learning_rate * training_error * data[i, :-1].reshape(-1, 1)
            bias += learning_rate * training_error

    return weights, bias
```

Step 7: Define a step activation function where it will return 1 if value >= 0 else 0.

```
def step_activation(value):
    return 1 if value >= 0 else 0
```

Step 8: Train the perceptron on training set.

```
LEARNING_RATE = 0.01
MAX_EPOCHS = 100
```

```
weights, bias = train_perceptron(X_train, y_train, LEARNING_RATE, MAX_EPOCHS)
```

```
weights
```

```
array([[-0.24328191],
       [-0.07354245]])
```

```
weights.shape
```

```
(2, 1)
```

```
bias
```

```
0.8999999999999999
```

Step 9: Make predictions using trained perceptron on test set. Tune the hyperparameters like learning rate, test size and find the optimal accurate perceptron model.

```
y_pred = np.array([step_activation(np.dot(X_test[i, :-1], weights) + bias) for i in range(len(X_test))])
```

```
print(f'Shape = {y_pred.shape}\tType = {type(y_pred)}\n')
pd.Series(y_pred, name = 'Label').head()
```

```
Shape = (100,)  Type = <class 'numpy.ndarray'>

0    0
1    1
2    0
3    1
4    1
Name: Label, dtype: int64
```

```
def predict_perceptron(weights, bias, data):
    predictions = np.dot(data[:, :-1], weights) + bias
    activations = np.vectorize(step_activation)(predictions)
    return activations

y_pred = predict_perceptron(weights, bias, X_test)
y_pred = y_pred.flatten()
```

```
print(f'Shape = {y_pred.shape}\tType = {type(y_pred)}\n')
pd.Series(y_pred, name = 'Label').head()
```

```
Shape = (100,)  Type = <class 'numpy.ndarray'>

0    0
1    1
2    0
3    1
4    1
Name: Label, dtype: int64
```

```
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

```
1.0
```

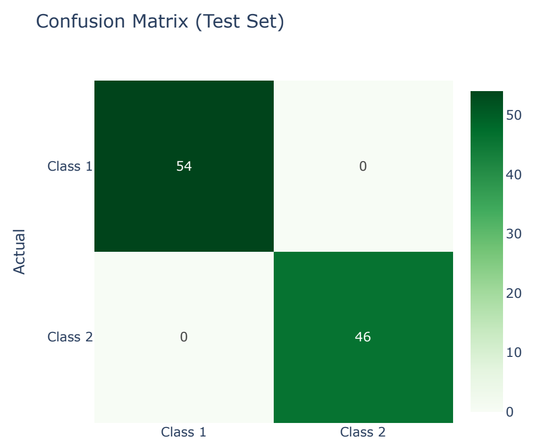
```
cf_matrix = confusion_matrix(y_test, y_pred)      # Index = Actual;  Column = Predicted
cf_matrix
```

```
array([[54,  0],
       [ 0, 46]])
```

```
class_labels = ['Class 1', 'Class 2']
```

```
fig = px.imshow(cf_matrix, text_auto = True, aspect = 'auto', color_continuous_scale = 'greens', width = 500, height = 500,
                title = 'Confusion Matrix (Test Set)',
                labels = dict(x = 'Predicted', y = 'Actual'),
                x = class_labels, y = class_labels)


fig.show()
fig.write_image('Confusion Matrix (Test Set).png')
```



Hyperparameter tuning

```
best_accuracy = 0
best_model = None
```

```
for learning_rate in [0.001, 0.01, 0.1]:
    for test_size in [0.2, 0.3, 0.4]:
        X_train_new, X_test_new, y_train_new, y_test_new = train_test_split(X, y, test_size = test_size, random_state = 42)

        weights, bias = train_perceptron(X_train_new, y_train_new, learning_rate, MAX_EPOCHS)
        y_pred_new = predict_perceptron(weights, bias, X_test_new)
        accuracy = accuracy_score(y_test_new, y_pred_new)

        if accuracy > best_accuracy:
            best_accuracy = accuracy
            best_model = {'weights': weights, 'bias': bias, 'learning_rate': learning_rate, 'test_size': test_size}
```

```
best_model
```

```
{'weights': array([[-0.18967397],
        [-0.07446646]]),
 'bias': 0.9229999999999998,
 'learning_rate': 0.001,
 'test_size': 0.3}
```

```
X_train_best, X_test_best, y_train_best, y_test_best = train_test_split(X, y, test_size = best_model['test_size'], random_state = 42)
```

```
print(f'Shape of X_train_best= {X_train_best.shape}\n')
print(f'Shape of y_train_best= {y_train_best.shape}\tType = {type(y_train_best)}\n')
print(f'Shape of X_test_best= {X_test_best.shape}\n')
print(f'Shape of y_test_best = {y_test_best.shape}\tType = {type(y_test_best)}\n')
```

```
Shape of X_train_best= (350, 3)

Shape of y_train_best= (350,)   Type = <class 'numpy.ndarray'>

Shape of X_test_best= (150, 3)

Shape of y_test_best = (150,)   Type = <class 'numpy.ndarray'>
```

```
best_weights, best_bias = train_perceptron(X_train_best, y_train_best, best_model['learning_rate'], MAX_EPOCHS)
```

```
best_weights
```

```
array([[-0.18967397],
       [-0.07446646]])
```

```
best_bias
```

```
0.9229999999999998
```

```
y_pred_best = predict_perceptron(weights, bias, X_test_best)
y_pred_best = y_pred_best.flatten()

print(f'Shape = {y_pred_best.shape}\tType = {type(y_pred_best)}\n')
pd.Series(y_pred_best, name = 'Label').head()
```

```
Shape = (150,)  Type = <class 'numpy.ndarray'>

0    0
1    1
2    0
3    1
4    1
Name: Label, dtype: int64
```

Step 10: Plot the decision boundary between two classified class.

```
a, b = best_weights.flatten()
c = best_bias
```

```
a
```

```
-0.1896739675237998
```

```
b
```

```
-0.07446646277414819
```

```
# Find the minimum and maximum values in Feature 1 (X-axis)

x_axis_vals = dataset[:, 0]
min_x_val = min(x_axis_vals)
max_x_val = max(x_axis_vals)

print(f'Range of X-axis is: ({min_x_val}, {max_x_val})')
```

```
Range of X-axis is: (-4.875477861541521, 7.3485254894826575)
```

```
# Generate x values

x_values = np.linspace(min_x_val, max_x_val, 100)
x_values
```

```
array([-4.87547786, -4.75200308, -4.6285283 , -4.50505352, -4.38157874,
       -4.25810395, -4.13462917, -4.01115439, -3.88767961, -3.76420483,
       -3.64073005, -3.51725527, -3.39378049, -3.2703057 , -3.14683092,
       -3.02335614, -2.89988136, -2.77640658, -2.6529318 , -2.52945702,
       -2.40598224, -2.28250745, -2.15903267, -2.03555789, -1.91208311,
       -1.78860833, -1.66513355, -1.54165877, -1.41818398, -1.2947092 ,
       -1.17123442, -1.04775964, -0.92428486, -0.80081008, -0.6773353 ,
       -0.55386052, -0.43038573, -0.30691095, -0.18343617, -0.05996139,
        0.06351339,  0.18698817,  0.31046295,  0.43393774,  0.55741252,
        0.6808873 ,  0.80436208,  0.92783686,  1.05131164,  1.17478642,
        1.2982612 ,  1.42173599,  1.54521077,  1.66868555,  1.79216033,
        1.91563511,  2.03910989,  2.16258467,  2.28605946,  2.40953424,
        2.53300902,  2.6564838 ,  2.77995858,  2.90343336,  3.02690814,
        3.15038292,  3.27385771,  3.39733249,  3.52080727,  3.64428205,
        3.76775683,  3.89123161,  4.01470639,  4.13818118,  4.26165596,
        4.38513074,  4.50860552,  4.6320803 ,  4.75555508,  4.87902986,
        5.00250464,  5.12597943,  5.24945421,  5.37292899,  5.49640377,
        5.61987855,  5.74335333,  5.86682811,  5.99030289,  6.11377768,
        6.23725246,  6.36072724,  6.48420202,  6.6076768 ,  6.73115158,
        6.85462636,  6.97810115,  7.10157593,  7.22505071,  7.34852549])
```

```
# Calculate y values based on the line equation

y_values = (-a * x_values - c) / b
y_values
```

```
array([24.81319994, 24.49869659, 24.18419325, 23.8696899 , 23.55518656,
       23.24068321, 22.92617987, 22.61167652, 22.29717318, 21.98266983,
       21.66816649, 21.35366314, 21.03915979, 20.72465645, 20.4101531 ,
       20.09564976, 19.78114641, 19.46664307, 19.15213972, 18.83763638,
       18.52313303, 18.20862969, 17.89412634, 17.57962299, 17.26511965,
       16.9506163 , 16.63611296, 16.32160961, 16.00710627, 15.69260292,
       15.37809958, 15.06359623, 14.74909289, 14.43458954, 14.1200862 ,
       13.80558285, 13.4910795 , 13.17657616, 12.86207281, 12.54756947,
       12.23306612, 11.91856278, 11.60405943, 11.28955609, 10.97505274,
       10.6605494 , 10.34604605, 10.03154271,  9.71703936,  9.40253601,
        9.08803267,  8.77352932,  8.45902598,  8.14452263,  7.83001929,
        7.51551594,  7.2010126 ,  6.88650925,  6.57200591,  6.25750256,
        5.94299921,  5.62849587,  5.31399252,  4.99948918,  4.68498583,
        4.37048249,  4.05597914,  3.7414758 ,  3.42697245,  3.11246911,
        2.79796576,  2.48346242,  2.16895907,  1.85445572,  1.53995238,
        1.22544903,  0.91094569,  0.59644234,  0.281939  , -0.03256435,
       -0.34706769, -0.66157104, -0.97607438, -1.29057773, -1.60508107,
       -1.91958442, -2.23408777, -2.54859111, -2.86309446, -3.1775978 ,
       -3.49210115, -3.80660449, -4.12110784, -4.43561118, -4.75011453,
       -5.06461787, -5.37912122, -5.69362456, -6.00812791, -6.32263126])
```

```
# Create a DataFrame for plotting

points_on_line = {'x': x_values, 'y': y_values}
line_df = pd.DataFrame(points_on_line)
line_df.head()
```

	x	y
0	-4.875478	24.813200
1	-4.752003	24.498697
2	-4.628528	24.184193
3	-4.505054	23.869690
4	-4.381579	23.555187

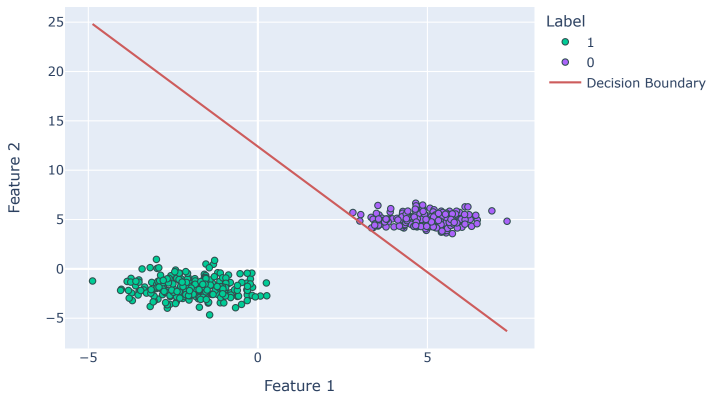
```
fig = px.scatter(data_df, x = 'Feature_1', y = 'Feature_2', width = 700, height = 500,
                        color = 'Label',
                        color_discrete_map={'1': 'indianred', '0': 'slateblue'},
                        title = 'Perceptron Decision Boundary for Two Classes',
                        labels={'Feature_1': 'Feature 1', 'Feature_2': 'Feature 2'})
```

```
                    labels={'feature_1': 'Feature 1', 'feature_2': 'Feature 2'})


fig.update_traces(marker=dict(size = 6,
                              line = dict(width = 1,
                                        color = 'DarkSlateGrey')))

fig.add_scatter(x=line_df['x'], y=line_df['y'], mode='lines', name = 'Decision Boundary', line=dict(color='indianred'))
fig.show()
fig.write_image('Perceptron Decision Boundary for Two Classes.png')
```

Perceptron Decision Boundary for Two Classes



Step 11: Plot the confusion matrix.

```
cf_matrix_best = confusion_matrix(y_test_best, y_pred_best)      # Index = Actual;  Column = Predicted
cf_matrix_best
```

```
    array([[74,  0],
           [ 0, 76]])
```

```
fig = px.imshow(cf_matrix_best, text_auto = True, aspect = 'auto', color_continuous_scale = 'blues', width = 500, height = 500,
                title = 'Confusion Matrix (Best Model)',
                labels = dict(x = 'Predicted', y = 'Actual'),
                x = class_labels, y = class_labels)


fig.show()
fig.write_image('Confusion Matrix (Best Model).png')
```

Confusion Matrix (Best Model)