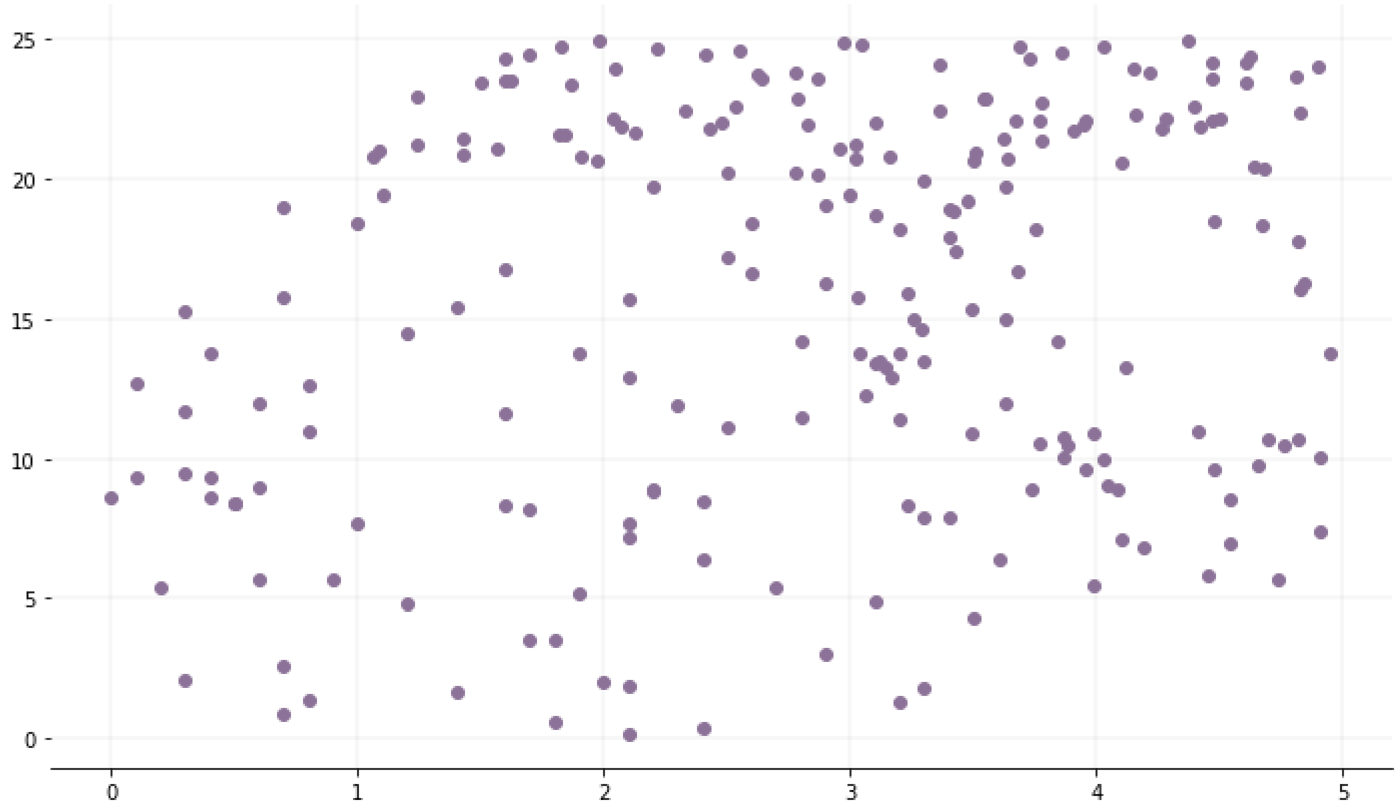
For the below programming, change the inputs and generate the output.

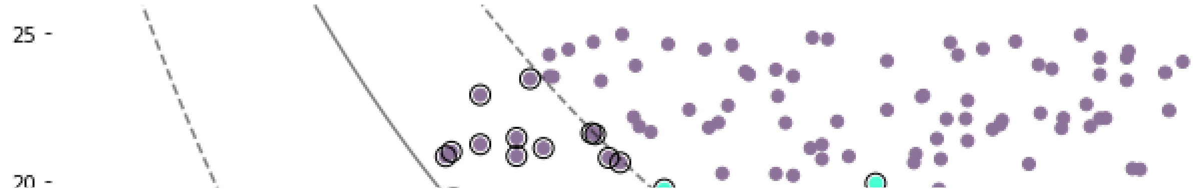
In [53]:

```
import random
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


def generate_random_dataset(size):
    x = []
    y = []
    target = []

    for i in range(size):

        # class zero
        x.append(np.round(random.uniform(0, 3.5), 1))
        y.append(np.round(random.uniform(0, 20), 1))
        target.append(0)

        # class one
        x.append(np.round(random.uniform(1, 5), 2))
        y.append(np.round(random.uniform(20, 25), 2))
        target.append(1)
        x.append(np.round(random.uniform(3, 5), 2))
        y.append(np.round(random.uniform(5, 25), 2))
        target.append(1)

        df_x = pd.DataFrame(data=x)
        df_y = pd.DataFrame(data=y)
        df_target = pd.DataFrame(data=target)

        data_frame = pd.concat([df_x, df_y], ignore_index=True, axis=1)
        data_frame = pd.concat([data_frame, df_target], ignore_index=True, axis=1)
        data_frame.columns = ['x', 'y', 'target']


    return data_frame
```

In [54]:

```
nerate  dataset
 = 100
set = generate_random_dataset(size)
ures  = dataset[['x', 'y']]
l = dataset['target']

ld out[Training] 80% of the dataset for training
_size  = int(np.round(size  * 0.8, 0))

lit dataset into training and testing sets
ain = features[:-test_size].values
ain = label[:-test_size].values
st = features[-test_size:].values
st = label[-test_size:].values

otting the training set
 ax = plt.subplots(figsize=(12, 7))

moving to and right border
pines['top'].set_visible(False)
pines['left'].set_visible(False)
pines['right'].set_visible(False)

ding major gridlines
rid(color='grey', linestyle='-', linewidth=0.25, alpha=0.5)
catter(features[:-test_size]['x'], features[:-test_size]['y'], color="#8C7298")

 sklearn import svm
l = svm.SVC(kernel='poly', degree=2)
l.fit(x_train, y_train)

show()

 ax = plt.subplots(figsize=(12, 7))

moving to and right border
pines['top'].set_visible(False)
pines['left'].set_visible(False)
pines['right'].set_visible(False)

eate grid to evaluate model
 np.linspace(-1, max(features['x']) + 1, len(x_train))
 np.linspace(0, max(features['y']) + 1, len(y_train))
XX = np.meshgrid(yy, xx)
 np.vstack([XX.ravel(), YY.ravel()]).T
n_size = len(features[:-test_size]['x'])

signing different colors to the classes
rs = y_train
rs = np.where(colors == 1, '#8C7298', '#47FFD1')

ot the dataset
catter(features[:-test_size]['x'], features[:-test_size]['y'], c=colors)

t the separating  hyperplane
model.decision_function(xy).reshape(XX.shape)

aw the decision boundary and margins
ontour(XX,  YY, Z, colors='k', levels=[-1,  0, 1], alpha=0.5, linestyles=['--',  '-', '--'])

ghlight support vectors  with a circle around them
catter(model.support_vectors_[:,  0], model.support_vectors_[:,  1], s=100, linewidth=1, facecolors='none
show()

 sklearn.metrics  import accuracy_score
ictions_poly  = model.predict(x_test)
racy_poly = accuracy_score(y_test,  predictions_poly)
t("2nd degree  polynomial Kernel\nAccuracy  (normalized): " + str(accuracy_poly))
```


25
20
15
10
5
0
0
1
2
3
4
5



```
2nd degree  polynomial Kernel
Accuracy  (normalized): 0.9
```

In [ ]: