In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, Normalizer
import matplotlib.pyplot as plt
from sklearn.svm import SVC
import warnings
warnings.filterwarnings("ignore")
```

```
Matplotlib is building the font cache; this may take a moment.
```

In [2]:
```python
def draw_line(coef,intercept, mi, ma):
    # for the separating hyper plane ax+by+c=0, the weights are [a, b] and the intercept is c
    # to draw the hyper plane we are creating two points
    # 1. ((b*min-c)/a, min) i.e ax+by+c=0 ==> ax = (-by-c) ==> x = (-by-c)/a here in place of y we are keeping the minimum value of y
    # 2. ((b*max-c)/a, max) i.e ax+by+c=0 ==> ax = (-by-c) ==> x = (-by-c)/a here in place of y we are keeping the maximum value of y
    points=np.array([[((-coef[1]*mi - intercept)/coef[0]), mi],[((-coef[1]*ma - intercept)/coef[0]), ma]])
    plt.plot(points[:,0], points[:,1])
```

What if Data is imabalanced

```
1. As a part of this task you will observe how linear models work in case of data imbalanced
2. observe how hyper plane is changs according to change in your learning rate.
3. below we have created 4 random datasets which are linearly separable and having class imbalance
4. in the first dataset the ratio between positive and negative is 100 : 2, in the 2nd data its 100:20,
in the 3rd data its 100:40 and in 4th one its 100:80
```

In [3]:
```python
np.random.seed(0)
```

In [4]:
```python
np.random.randint(12)
```

Out[4]:
```
5
```

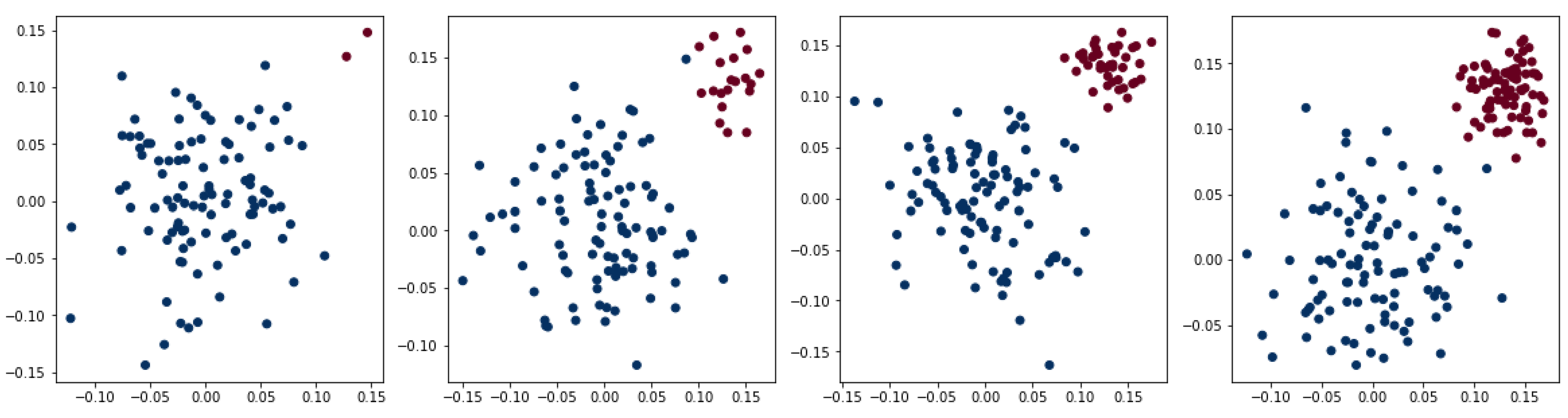
In [5]:
```python
from collections import namedtuple

Data = namedtuple('Data', ['X', 'Y'])
```

In [6]:
```python
def generate_data(ratio):
    X_p=np.random.normal(0,0.05,size=(ratio[0],2))   # Positive Points X
    X_n=np.random.normal(0.13,0.02,size=(ratio[1],2)) # Negative Points X
    y_p=np.array([1]*ratio[0]).reshape(-1,1)
    y_n=np.array([0]*ratio[1]).reshape(-1,1)
    X=np.vstack((X_p,X_n))
    y=np.vstack((y_p,y_n))

    print(X.shape)

    return Data(X, y)
```

In [7]:
```python
# here we are creating 2d imbalanced data points
ratios = [(100,2), (100, 20), (100, 40), (100, 80)]
Datasets = list(map(generate_data, ratios))
```

```
(102, 2)
(120, 2)
(140, 2)
(180, 2)
```

In [8]:
```python
plt.figure(figsize=(20,5))
for i, data in enumerate(Datasets): # data -> (x,y)
    plt.subplot(1, 4, i+1)
    #plt.scatter(X_p[:,0],X_p[:,1])
    plt.scatter(data.X[:,0],data.X[:,1], c=data.Y, cmap='RdBu')
plt.show()
```



> your task is to apply SVM (sklearn.svm.SVC) and LR (sklearn.linear_model.LogisticRegression) with different regularization strength [0.001, 1, 100]

Task 1: Applying SVM

```
1. you need to create a grid of plots like this
```

```
in each of the cell[i][j] you will be drawing the hyper plane that you get after applying SVM on ith dataset and
        jth learnig rate

i.e
```

Plane(SVM().fit(D1, C=0.001))	Plane(SVM().fit(D1, C=1))	Plane(SVM().fit(D1, C=100))
Plane(SVM().fit(D2, C=0.001))	Plane(SVM().fit(D2, C=1))	Plane(SVM().fit(D2, C=100))
Plane(SVM().fit(D3, C=0.001))	Plane(SVM().fit(D3, C=1))	Plane(SVM().fit(D3, C=100))
Plane(SVM().fit(D4, C=0.001))	Plane(SVM().fit(D4, C=1))	Plane(SVM().fit(D4, C=100))

```
if you can do, you can represent the support vectors in different colors,
which will help us understand the position of hyper plane
```

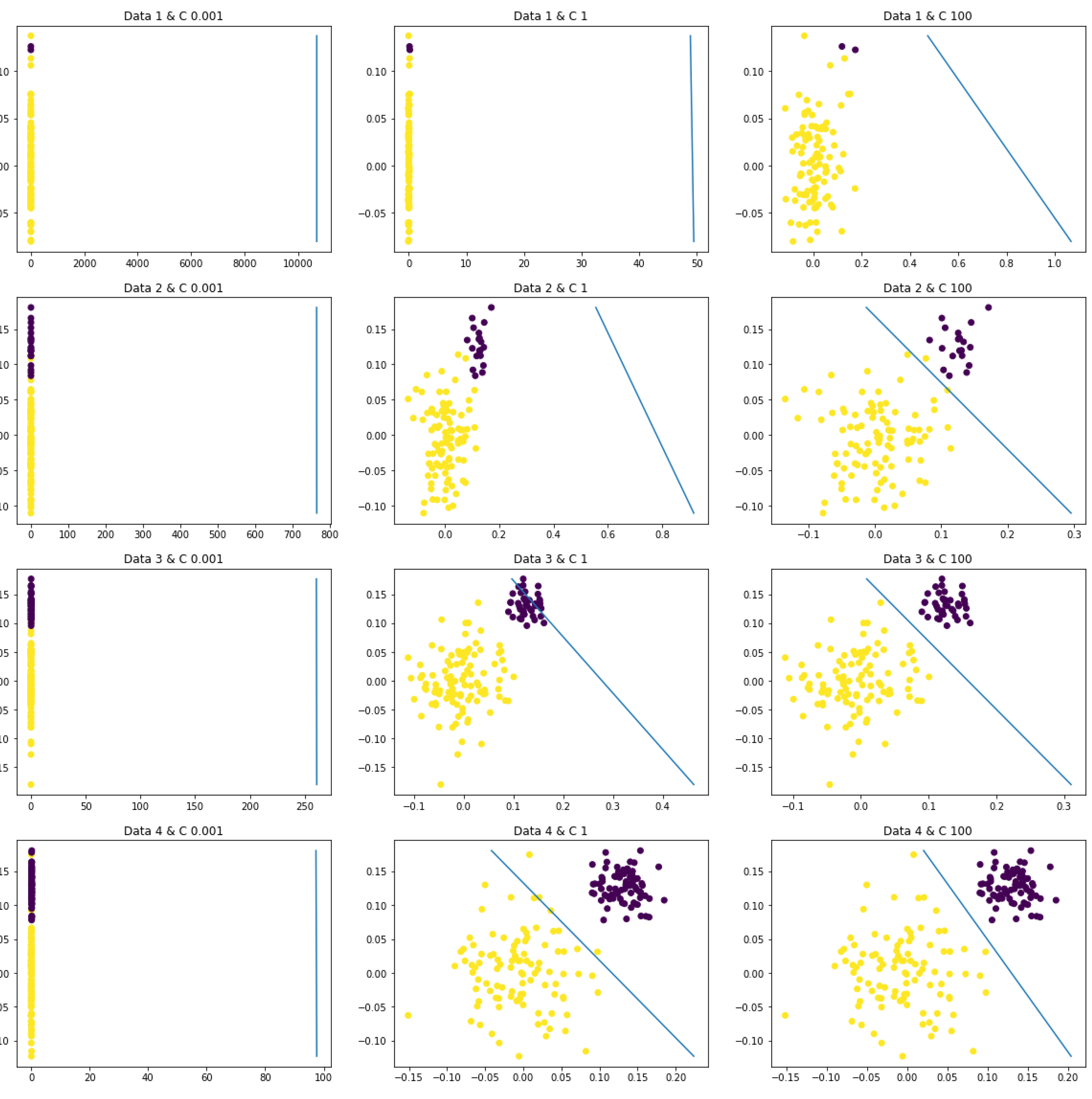
Write in your own words, the observations from the above plots, and what do you think about the position of the hyper plane

```
check the optimization problem here https://scikit-learn.org/stable/modules/svm.html#mathematical-formulation

if you can describe your understanding by writing it on a paper
and attach the picture, or record a video upload it in assignment.
```

NOTE :- Plane is define by Weight Vectors

In [9]:
```python
def svm_linear(X, y, regParam):
    ''' Apply SVM Linear Kernel(ie Hinge Loss) & return Weights & bias'''
    classifier = SVC(kernel='linear', C=regParam)
    classifier.fit(X, y)

    # As Coef is nd Array & intercept is also Array
    w = classifier.coef_[0]
    b = classifier.intercept_[0]
    return w, b
```

In [150…
```python
plt.figure(figsize=(20, 20))
for i, d in enumerate(Datasets, start=1):
    x1, x2 = d.X[:,0], d.X[:,1]
    mi, ma = np.min(x2), np.max(x2)
    row = 3*(i-1)
    for j, C in enumerate([0.001, 1, 100], start=0):
        plt.subplot(4, 3, row + j + 1)
        plt.scatter(x1,x2, c=d.Y)
        w, b = svm_linear(d.X, d.Y, C)
        draw_line(w, b, mi, ma)
        plt.title(f'Data {i} & C {C}')
```



In [61]:
```python
np.max(d.X[:,1])
```

Out[61]:
```
0.14559193255359343
```

Task 2: Applying LR

```
you will do the same thing what you have done in task 1.1, except instead of SVM you apply logistic regression
```

```
these are results we got when we are experimenting with one of the model
```

In [146…
```python
#you can start writing code here.

def logistic(X, y, regParam):
    ''' Apply Logistic Regression & return Weights & bias'''
    # Here LogisticRegression is similar to SGDClassifier with loss=log, & you can consider it as a probabilistic way of solving problem
    classifier = LogisticRegression(C=regParam)
    classifier.fit(X, y)

    # As Coef is nd Array & intercept is also Array
    w = classifier.coef_[0]
    b = classifier.intercept_[0]
    return w,b
```

In [152…
```python
plt.figure(figsize=(20, 20))
for i, d in enumerate(Datasets, start=1):
    x1, x2 = d.X[:,0], d.X[:,1]
    mi, ma = np.min(x2), np.max(x2)
    row = 3*(i-1)
    for j, C in enumerate([0.001, 1, 100], start=0):
        plt.subplot(4, 3, row + j + 1)
        plt.scatter(x1,x2, c=d.Y)
        w, b = logistic(d.X, d.Y, C)
        draw_line(w, b, mi, ma)
        plt.title(f'Data {i} & C {C}')
```

Observations

- For Imbalannced Dataset Logistic Regressionn performs quite well compare to SVC
- Imbalanced Dataset requires some more proclivity towards overfitting as Plane appear some what reasonable for larger value of C (ie We need to do more work for Generalization)

In [ ]: