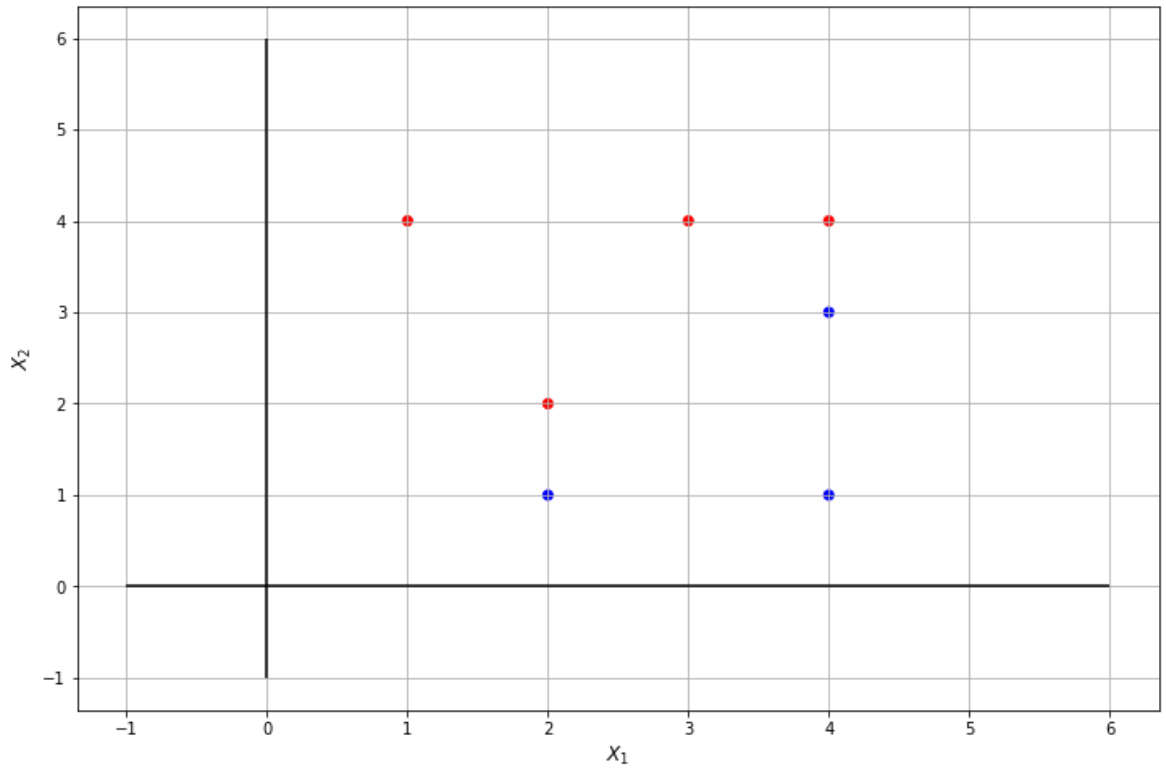
In [6]:
```
"""
Landon Buell
Marek Petrik
CS 750.01
3 April 2020
"""

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

Problem 1 [33%]

Here we explore the maximal margin classifier on a toy data set.

(a) We are given n = 7 observations in p = 2 dimensions. For each observation, there is an associated class label. Sketch (by hand is OK) the observations.

In [7]:
```
# Design Matricies
X = np.array([[3,4],[2,2],[4,4],[1,4],[2,1],[4,3],[4,1]])
y = ['red','red','red','red','blue','blue','blue']

# visualize
plt.figure(figsize=(12,8))
plt.xlabel('$X_1$',size=12,weight='bold')
plt.ylabel('$X_2$',size=12,weight='bold')
plt.scatter(x=X.transpose()[0],y=X.transpose()[1],c=y)
plt.vlines(0,-1,6,color='black')
plt.hlines(0,-1,6,color='black')
plt.grid()
plt.show()
```

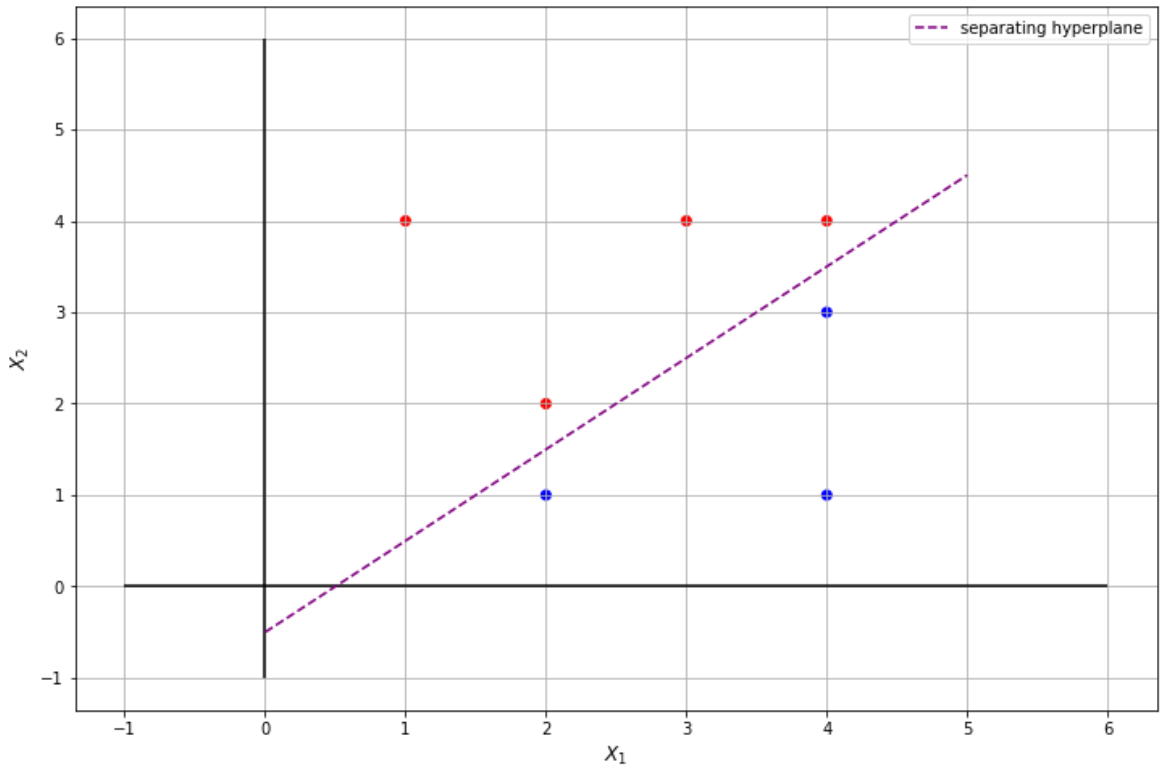


(b) Sketch (by hand is OK) the optimal separating hyperplane, and provide the equation for this hyperplane (of the form (9.1)).

Equation: $-\frac{1}{2} + 1X_1 - 1X_2$

In [8]:

```
# Design Matricies
X = np.array([[3,4],[2,2],[4,4],[1,4],[2,1],[4,3],[4,1]])
y = ['red','red','red','red','blue','blue','blue']

hyperplane = 1*np.arange(0,6) - 1/2

# visualize w/ hyperplane
plt.figure(figsize=(12,8))
plt.xlabel('$X_1$',size=12,weight='bold')
plt.ylabel('$X_2$',size=12,weight='bold')
plt.scatter(x=X.transpose()[0],y=X.transpose()[1],c=y)
plt.plot(hyperplane,linestyle='--',color='purple',label='separating hyperpla
plt.vlines(0,-1,6,color='black')
plt.hlines(0,-1,6,color='black')
plt.legend()
plt.grid()
plt.show()
```



(c) Describe the classification rule for the maximal margin classifier. It should be something along the lines of "Classify to Red if $\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$, and classify to Blue otherwise." Provide the values for β_0, β_1, and β_2.

Classification Rule: Class = 'Red' if

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$$

Class = 'Blue' if

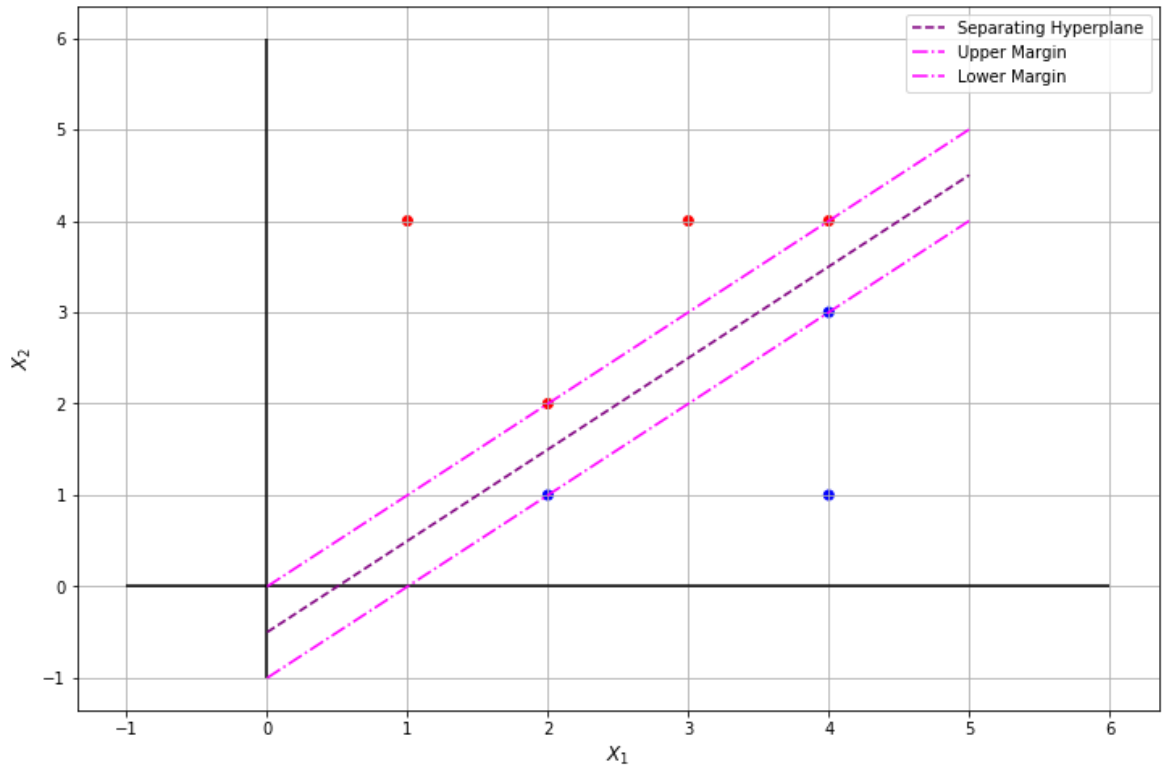
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 < 0$$

Where

$$\beta_0 = -\frac{1}{2}, \beta_1 = 1, \beta_2 = -1$$

(d) On your sketch, indicate the margin for the maximal margin hyperplane.

In [9]:
```
# Design Matricies
X = np.array([[3,4],[2,2],[4,4],[1,4],[2,1],[4,3],[4,1]])
y = ['red','red','red','red','blue','blue','blue']

# Hyper planes
hyperplane = 1*np.arange(0,6) - 1/2
upper = 1*np.arange(0,6)
lower = 1*np.arange(0,6) - 1

# visualize w/ hyperplane
plt.figure(figsize=(12,8))
plt.xlabel('$X_1$',size=12,weight='bold')
plt.ylabel('$X_2$',size=12,weight='bold')
plt.scatter(x=X.transpose()[0],y=X.transpose()[1],c=y)
plt.plot(hyperplane,linestyle='--',color='purple',label='Separating Hyperpla
plt.plot(upper,linestyle='-.',color='magenta',label='Upper Margin')
plt.plot(lower,linestyle='-.',color='magenta',label='Lower Margin')
plt.vlines(0,-1,6,color='black')
plt.hlines(0,-1,6,color='black')
plt.legend()
plt.grid()
plt.show()
```



(e) Indicate the support vectors for the maximal margin classifier. How will the number of support vectors depend on the dimensionality of the space.

The Support vectors are given by:

$$0 = -1 + 1X_1 - 1X_2$$
$$0 = +0 + 1X_1 - 1X_2$$

For N dimensions, there will be N support vectors

(f) Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane.

The 7th data point is located at $X_1 = 4$, $X_2 = 1$. The maximal margin classifier relatively far from this oint. If we were to change the value of either feature by 1 unit in any direction, the perturbed sample would still lie on same side of a separating hyperplane. Thus, a "slight movement" change would have the same result.

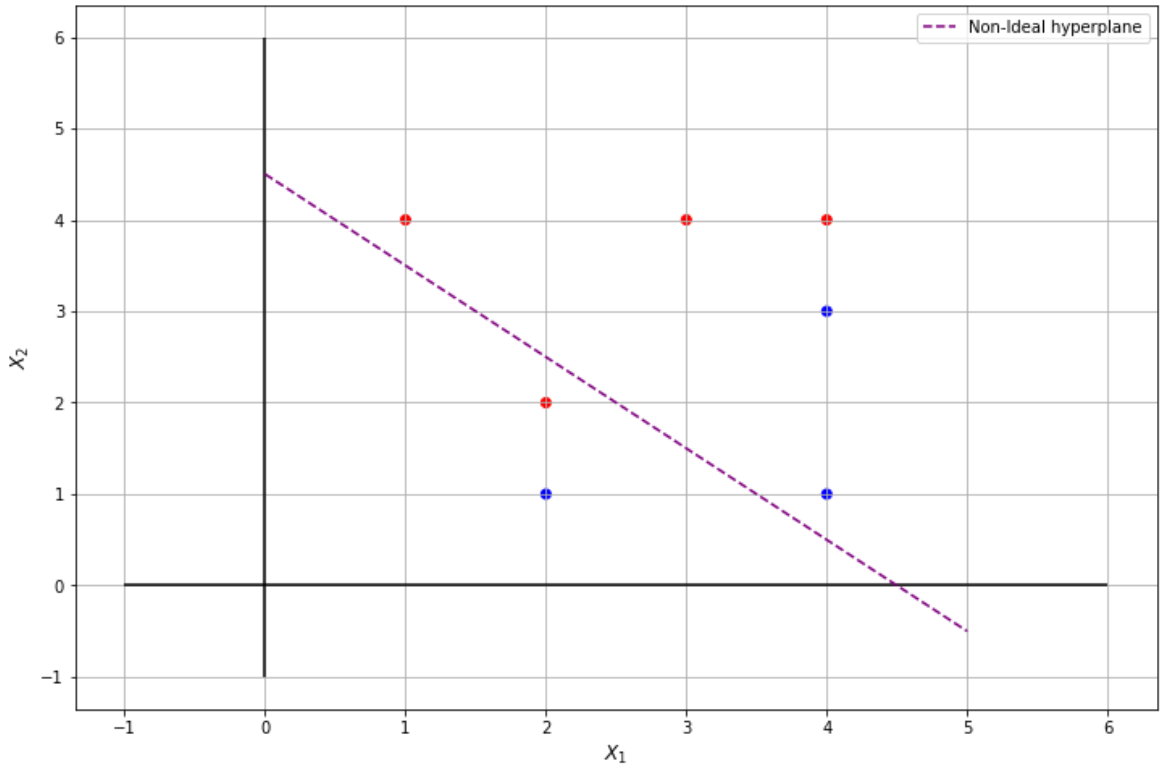
(g) Sketch a hyperplane that is not the optimal separating hyper-plane, and provide the equation for this hyperplane.

A Non-ideal hyperplane would be given by the equation:

$$0 = +4.5 + 1X_1 - 1X_1$$

Plotted Below:

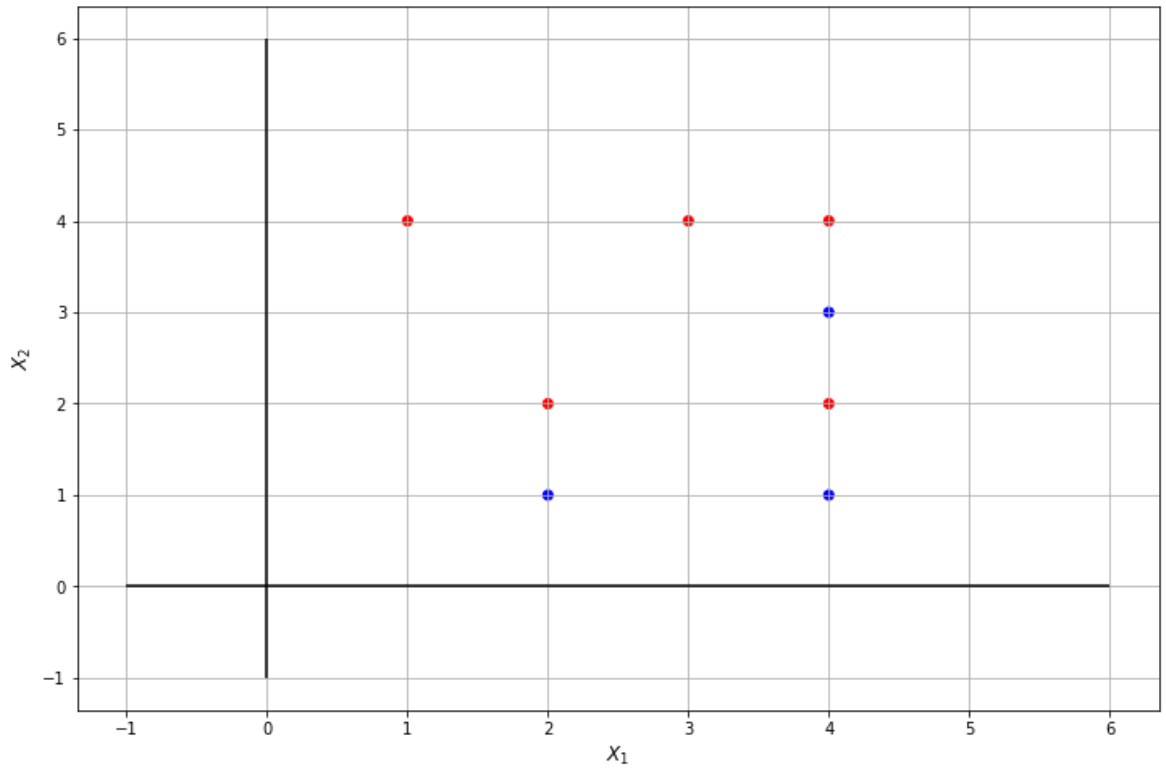
In [12]:
```
# Design Matricies
X = np.array([[3,4],[2,2],[4,4],[1,4],[2,1],[4,3],[4,1]])
y = ['red','red','red','red','blue','blue','blue']

hyperplane = -1*np.arange(0,6) + 4.5

# visualize w/ hyperplane
plt.figure(figsize=(12,8))
plt.xlabel('$X_1$',size=12,weight='bold')
plt.ylabel('$X_2$',size=12,weight='bold')
plt.scatter(x=X.transpose()[0],y=X.transpose()[1],c=y)
plt.plot(hyperplane,linestyle='--',color='purple',label='Non-Ideal hyperplan
plt.vlines(0,-1,6,color='black')
plt.hlines(0,-1,6,color='black')
plt.legend()
plt.grid()
plt.show()
```



(h) Draw an additional observation on the plot so that the two classes are no longer separable by a hyperplane.

In [17]:

```
# Design Matricies
X = np.array([[3,4],[2,2],[4,4],[1,4],[2,1],[4,3],[4,1]])
y = ['red','red','red','red','blue','blue','blue']

# Add new Points
X = np.append(X,[4,2]).reshape(-1,2)
y.append('red')

# visualize
plt.figure(figsize=(12,8))
plt.xlabel('$X_1$',size=12,weight='bold')
plt.ylabel('$X_2$',size=12,weight='bold')
plt.scatter(x=X.transpose()[0],y=X.transpose()[1],c=y)
plt.vlines(0,-1,6,color='black')
plt.hlines(0,-1,6,color='black')
plt.grid()
plt.show()
```

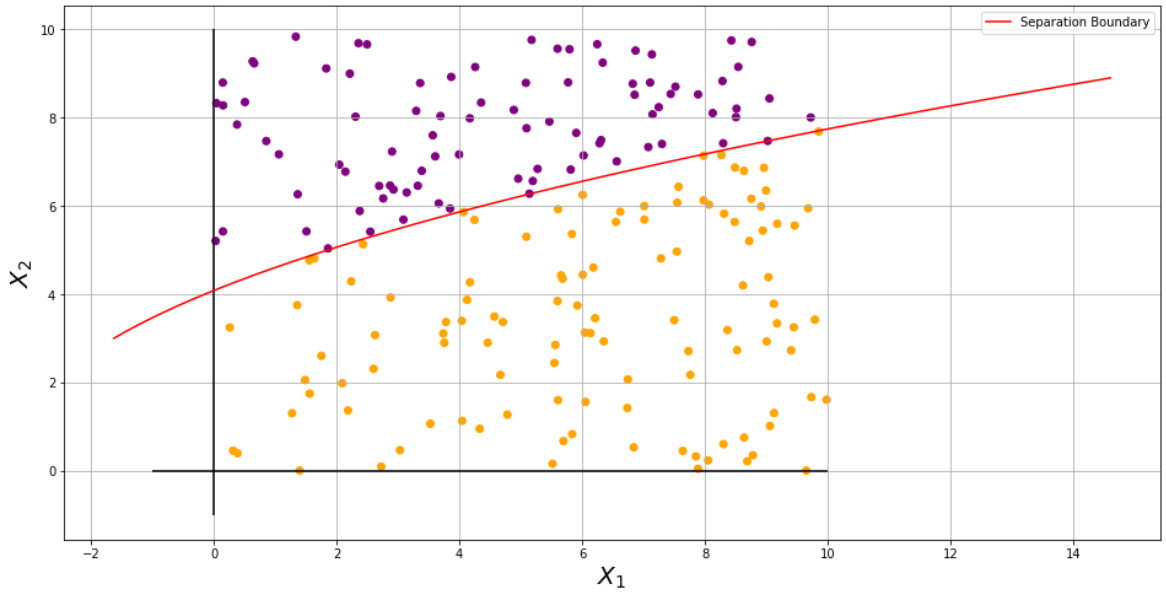


Problem 2 [33%]

Generate a simulated two-class data set with 200 observations and two features in which there is a visible but non-linear separation between the two classes. Explore whether in this setting, a support vector machine with a polynomial kernel (with degree greater than 1) or a radial kernel will outperform a support vector classifier on the training data. Which technique performs best on the test data? Make plots and report training and test error rates in order to back up your assertions.

In [19]:

```
""" Generate the Simulated Data set - Numpy.Random() """

X1 = np.random.random(size=200)*10
X2 = np.random.random(size=200)*10

labels = []

a,b,c,d = 0.7,-0.9,-2.7,-8

for x1,x2 in zip(X1,X2):
    if a*x2**2 + b*x2 + c*x1 + d > 0:
        labels.append('purple')
    else:
        labels.append('orange')

base = np.arange(3,9,0.1)
curve = -(a/c)*base**2 - (b/c)*base - (d/c)

# visualize w/ hyperplane
plt.figure(figsize=(16,8))
plt.xlabel('$X_1$',size=20,weight='bold')
plt.ylabel('$X_2$',size=20,weight='bold')

plt.scatter(x=X1,y=X2,c=labels)

plt.plot(curve,base,color='red',label='Separation Boundary')

plt.vlines(0,-1,10,color='black')
plt.hlines(0,-1,10,color='black')

plt.legend()
plt.grid()
plt.show()
```



In [21]:
```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

SVC_CLF = SVC(C=100,kernel='linear',degree=2,gamma='auto')

# Concatenate X1 & X2
X = np.array([X1,X1]).transpose()
y = labels[:]
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.4)

SVC_CLF = SVC_CLF.fit(X_train,y_train)
y_pred = SVC_CLF.predict(X_test)

#classification_report(y_test,y_pred)
classification_report(y_test,y_pred,output_dict=True)

# Not a great model!
```

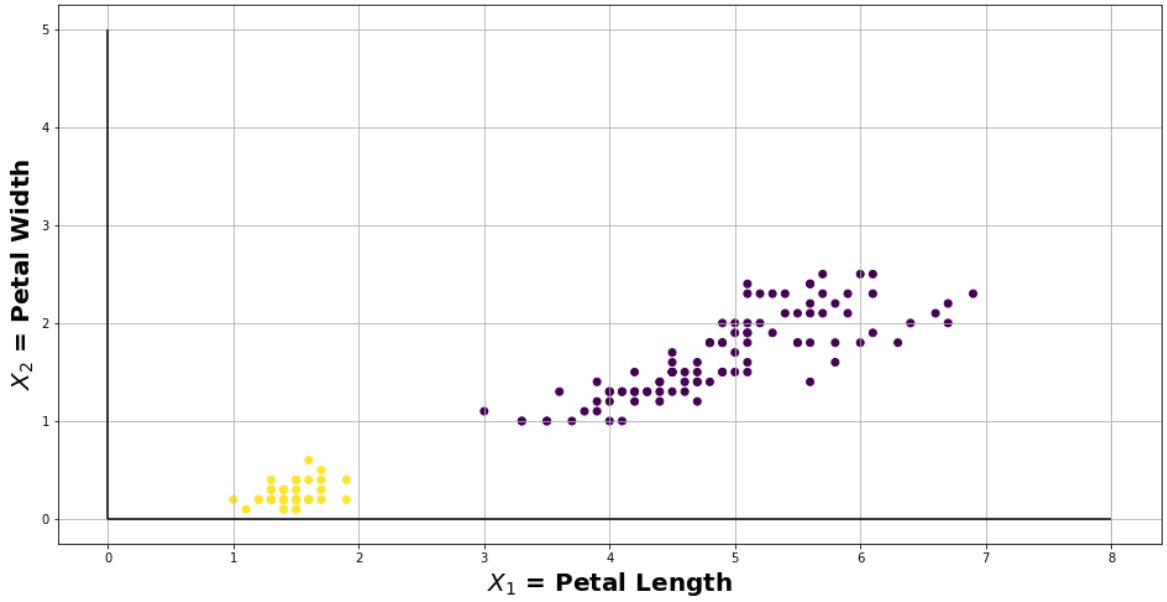
Out[21]:
```
{'orange': {'precision': 0.5918367346938775,
  'recall': 0.7631578947368421,
  'f1-score': 0.6666666666666667,
  'support': 38},
 'purple': {'precision': 0.7096774193548387,
  'recall': 0.5238095238095238,
  'f1-score': 0.6027397260273972,
  'support': 42},
 'accuracy': 0.6375,
 'macro avg': {'precision': 0.6507570770243581,
  'recall': 0.643483709273183,
  'f1-score': 0.634703196347032,
  'support': 80},
 'weighted avg': {'precision': 0.6537030941408821,
  'recall': 0.6375,
  'f1-score': 0.6331050228310502,
  'support': 80}}
```

Problem 3 [33%]

Apply SVMs and at least 3 different kernels to a data set of your choice. Use cross-validation to optimize the parameter C. Be sure to fit the models on a training set and to evaluate their performance on a test set. How accurate are the results compared to simple methods like linear or logistic regression? Which of these approaches yields the best performance?

In [84]:

```
# LOAD in sklearn IRIS dataset
# This is Adapted from Geron Hands-on Machine Learning (2017)
from sklearn.datasets import load_iris

X,y = load_iris(return_X_y=True)
# extract 2 features (petal length, petal width)
X = X[:,(2,3)]
# reduce to binary classificiation (Iris - Virginica)
y = (y == 0).astype(np.float64)

# visualize
plt.figure(figsize=(16,8))
plt.xlabel("$X_1$ = Petal Length",size=20,weight='bold')
plt.ylabel("$X_2$ = Petal Width",size=20,weight='bold')
plt.scatter(x=X.transpose()[0],y=X.transpose()[1],c=y)

plt.hlines(0,0,8,color='black')
plt.vlines(0,0,5,color='black')
plt.grid()
plt.show()

# We have a LINEARLY Separable dataset!
```



In [85]:
```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

from sklearn.model_selection import cross_val_score

# Organize train/test data
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.4,random_st
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(90, 2)
(60, 2)
(90,)
(60,)
```

In [86]:
```
# USE SVM w/ linear kernel
SVC_linear = SVC(C=10,kernel='linear')
SVC_linear.fit(X_train,y_train)
y_pred = SVC_linear.predict(X_test)
print(classification_report(y_test,y_pred))
# Concerningly good model>
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        44
         1.0       1.00      1.00      1.00        16

    accuracy                           1.00        60
   macro avg       1.00      1.00      1.00        60
weighted avg       1.00      1.00      1.00        60
```

In [87]:
```
# USE SVM w/ Polynomial kernel
SVC_poly = SVC(C=100,kernel='poly',degree=5,gamma='scale')
SVC_poly.fit(X_train,y_train)
y_pred = SVC_poly.predict(X_test)
print(classification_report(y_test,y_pred))
# Concerningly good model too?
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        44
         1.0       1.00      1.00      1.00        16

    accuracy                           1.00        60
   macro avg       1.00      1.00      1.00        60
weighted avg       1.00      1.00      1.00        60
```

In [88]:
```
# USE SVM w/ Radial Gaussian Basis Function kernel
SVC_RBF = SVC(C=100,kernel='rbf')
SVC_RBF.fit(X_train,y_train)
y_pred = SVC_RBF.predict(X_test)
print(classification_report(y_test,y_pred))
# Concerningly good model too?
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        44
         1.0       1.00      1.00      1.00        16

    accuracy                           1.00        60
   macro avg       1.00      1.00      1.00        60
weighted avg       1.00      1.00      1.00        60


C:\Users\Landon\Anaconda3\lib\site-packages\sklearn\svm\base.py:193: Future
Warning: The default value of gamma will change from 'auto' to 'scale' in v
ersion 0.22 to account better for unscaled features. Set gamma explicitly t
o 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

I'm not sure whats going on here, even when I change the hyperparamter, C, things go horribly right! I see that the data is perfectly lienarly separable, but even tweaking things causes perfect preicison/recall scores every time? This seems a bit too perfect. I would also expect linear regression to also work horribly as this is not a regression problem, and the result would be a best fit line, almost parallel the the separating hyperplane. Logisitic regression might work fine for this problem, but not as well due to the nature of a separation boundary.

In [ ]:
```
```