In [29]:

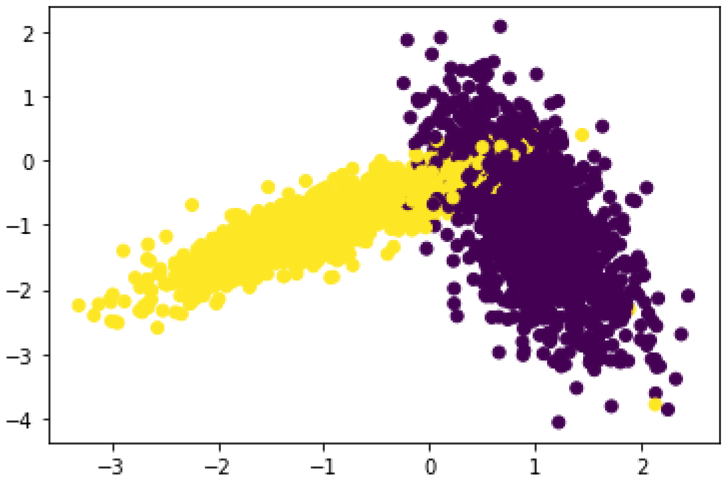
```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy
from tqdm import tqdm
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import random
import warnings
warnings.filterwarnings("ignore")
from matplotlib.colors import ListedColormap
import matplotlib.colors
```

In [30]:

```
x,y = make_classification(n_samples=10000, n_features=2, n_informative=2, n_redundant=
0, n_clusters_per_class=1, random_state=60)
X_train, X_test, y_train, y_test = train_test_split(x,y,stratify=y,random_state=42)

# del X_train,X_test
```

In [31]:

```
%matplotlib inline
import matplotlib.pyplot as plt
colors = {0:'red', 1:'blue'}
plt.scatter(X_test[:,0], X_test[:,1],c=y_test)
plt.show()
```



Implementing Custom RandomSearchCV

def RandomSearchCV(x_train,y_train,classifier, param_range, folds):

```
# x_train: its numpy array of shape, (n,d)
# y_train: its numpy array of shape, (n,) or (n,1)
# classifier: its typically KNeighborsClassifier()
# param_range: its a tuple like (a,b) a < b
# folds: an integer, represents number of folds we need to devide the data and t
est our model


#1.generate 10 unique values(uniform random distribution) in the given range "pa
ram_range" and store them as "params"
# ex: if param_range = (1, 50), we need to generate 10 random numbers in range 1
to 50
#2.devide numbers ranging from  0 to len(X_train) into groups= folds
# ex: folds=3, and len(x_train)=100, we can devide numbers from 0 to 100 into 3
 groups
  group 1: 0-33, group 2:34-66, group 3: 67-100
#3.for each hyperparameter that we generated in step 1:
    # and using the above groups we have created in step 2 you will do cross-val
idation as follows

    # first we will keep group 1+group 2 i.e. 0-66 as train data and group 3: 67
-100 as test data, and find train and
      test accuracies

    # second we will keep group 1+group 3 i.e. 0-33, 67-100 as train data and gr
oup 2: 34-66 as test data, and find
      train and test accuracies

    # third we will keep group 2+group 3 i.e. 34-100 as train data and group 1:
 0-33 as test data, and find train and
      test accuracies
    # based on the 'folds' value we will do the same procedure

    # find the mean of train accuracies of above 3 steps and store in a list "tr
ain_scores"
    # find the mean of test accuracies of above 3 steps and store in a list "tes
t_scores"
    #4. return both "train_scores" and "test_scores"

    #5. call function RandomSearchCV(x_train,y_train,classifier, param_range, fo
lds) and store the returned values into "train_score", and "cv_scores"
    #6. plot hyper-parameter vs accuracy plot as shown in reference notebook and
choose the best hyperparameter
    #7. plot the decision boundaries for the model initialized with the best hyp
erparameter, as shown in the last cell of reference notebook
```

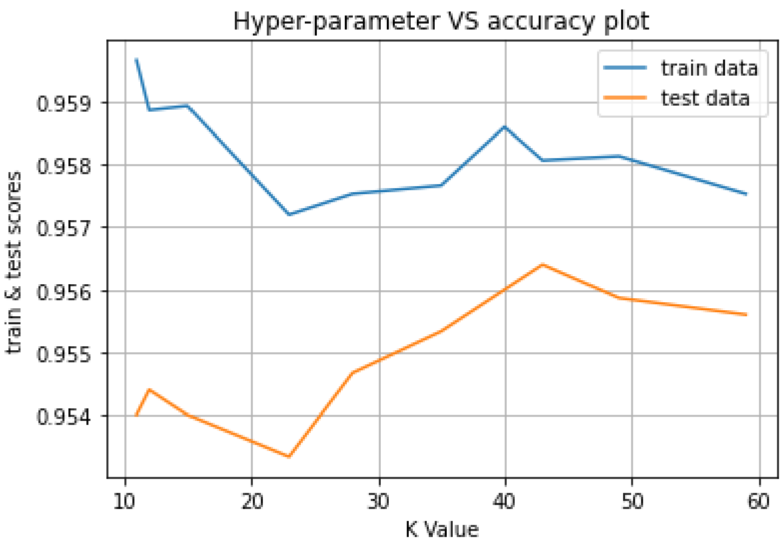
In [39]:

```
def RandomSearchCV(x_train,y_train,classifier,folds):
    group=[]
    grp=[]
    trainscores = []
    testscores  = []
    params_range = (10,60)
    params_list = [i for i in range(params_range[0], params_range[1])]
    params_list = random.sample(params_list,10)
    params_list.sort()

    for k in tqdm(params_list):
        trainscores_folds = []
        testscores_folds  = []
        for i in range(0, folds):
            value=(len(x_train)/(folds))
            boundary=int(value)
            test_indices=list(set(list(range((boundary*i), (boundary*(i+1))))))
            train_indices = list(set(list(range(0, len(x_train)))) - set(test_indices))
            X_train = x_train[train_indices]
            Y_train = y_train[train_indices]
            X_test  = x_train[test_indices]
            Y_test  = y_train[test_indices]
            classifier.n_neighbors = k
            classifier.fit(X_train,Y_train)
            Y_predicted = classifier.predict(X_test)
            testscores_folds.append(accuracy_score(Y_test, Y_predicted))
            Y_predicted = classifier.predict(X_train)
            trainscores_folds.append(accuracy_score(Y_train, Y_predicted))
        trainscores.append(np.mean(np.array(trainscores_folds)))
        testscores.append(np.mean(np.array(testscores_folds)))
    return trainscores,testscores,params_list
```

In [41]:

```
neighbrs = KNeighborsClassifier()
folds = 3
trainscores,testscores,params = RandomSearchCV(X_train, y_train, neighbrs,folds)

plt.plot(params,trainscores, label='train data')
plt.plot(params,testscores, label='test data')

plt.title('Hyper-parameter VS accuracy plot')
plt.legend()
plt.grid()

plt.xlabel('K Value')
plt.ylabel('train & test scores')

plt.show()
```

```
  0%|          | 0/10 [00:00<?, ?it/s]

[11, 12, 15, 23, 28, 35, 40, 43, 49, 59]

100%|██████████| 10/10 [00:07<00:00,  1.40it/s]
```

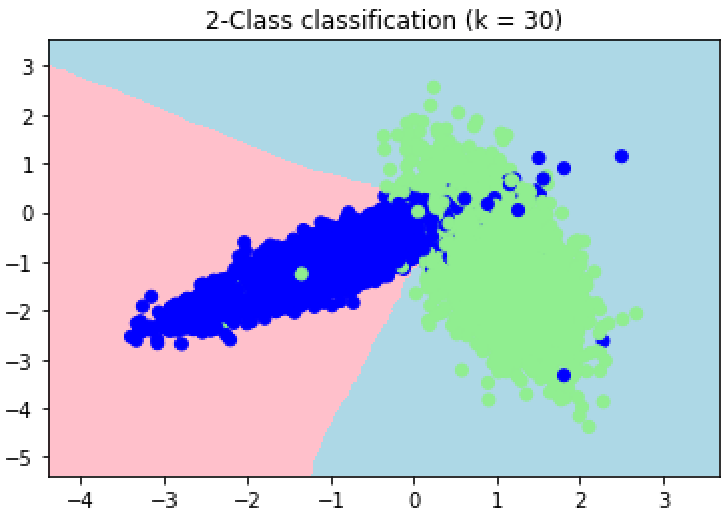


In [42]:

```
def plot_decision_boundary(X1, X2, y, clf):
        # Create color maps
    color1 =["lightblue", "gray",  'pink']
    color2 =['lightgreen', 'orange','blue']
    cmap_light = matplotlib.colors.ListedColormap(color1)
    cmap_bold = matplotlib.colors.ListedColormap(color2)

    x_min, x_max = X1.min() - 1, X1.max() + 1
    y_min, y_max = X2.min() - 1, X2.max() + 1

    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.figure()
    plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
    # Plot also the training points
    plt.scatter(X1, X2, c=y, cmap=cmap_bold)

    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.title("2-Class classification (k = %i)" % (clf.n_neighbors))
    plt.show()
```

In [43]:

```
knn_class = KNeighborsClassifier(n_neighbors = 30)
knn_class.fit(X_train, y_train)
plot_decision_boundary(X_train[:, 0], X_train[:, 1], y_train, knn_class)
```



In [ ]:

```

```