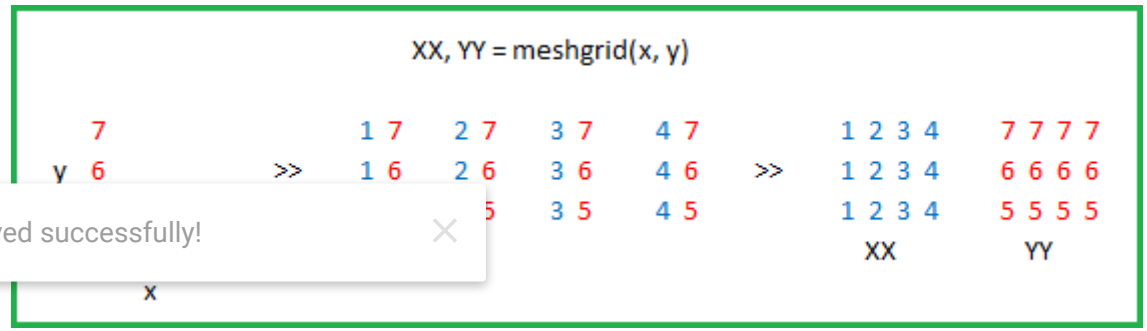
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier


from sklearn import datasets, neighbors


from mlxtend.plotting import plot_decision_regions


from google.colab import files
from google.colab import drive
drive.mount("/content/drive/")
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.a

Enter your authorization code:
··········
Mounted at /content/drive/
```

```
from IPython.display import Image


Image("/content/drive/My Drive/Colab Notebooks/knn/demo_data/meshgrid_image.png")
```



Saved successfully!

```
data_ushape=pd.read_csv("/content/drive/My Drive/Colab Notebooks/knn/demo_data/1_ushape.csv",names=["x1","x2","y"])
```

```
data_ushape=pd.read_csv("/content/drive/My Drive/Colab Notebooks/knn/demo_data/1.ushape.csv",names=["x1","x2","y"])
```

```
data_ushape.columns=["x1","x2","y"];
```

```
data_ushape.head()
```

```
def knn(data,i):
  x=data[["x1","x2"]].values
  y=data["y"].astype(int).values
  clf=neighbors.KNeighborsClassifier(n_neighbors=i)
  clf.fit(x,y)
  plot_decision_regions(x,y,clf=clf,legend=2)
  plt.xlabel("X1")
  plt.ylabel("X2")
  plt.title("The plot of k:"+str(i))
  plt.show()
```

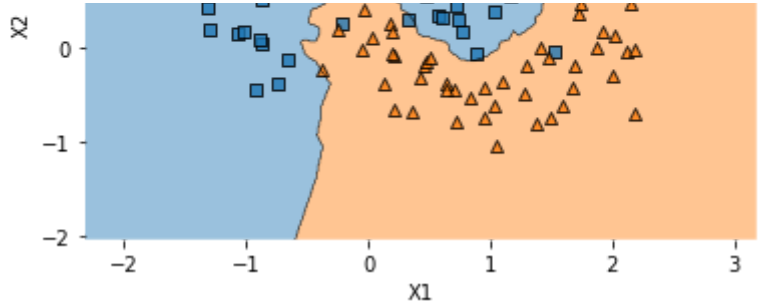
```
data_new=data_ushape[["x1","x2"]].values
data_new.shape
```

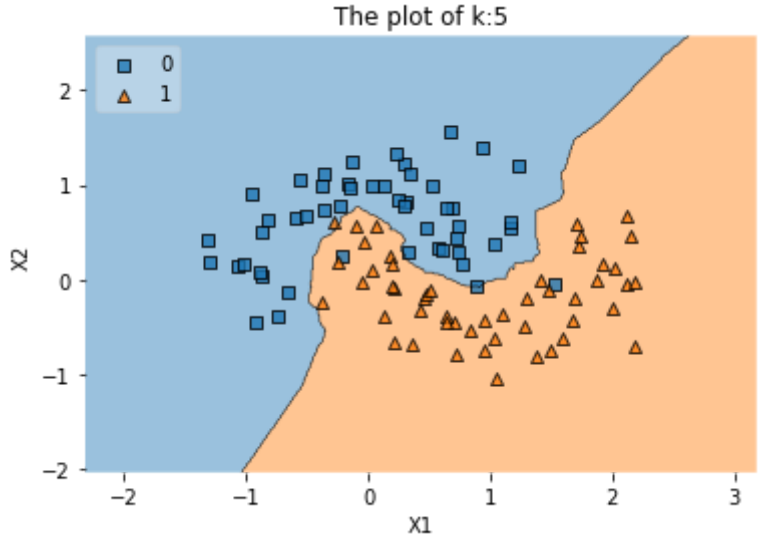
```
(100, 2)
```

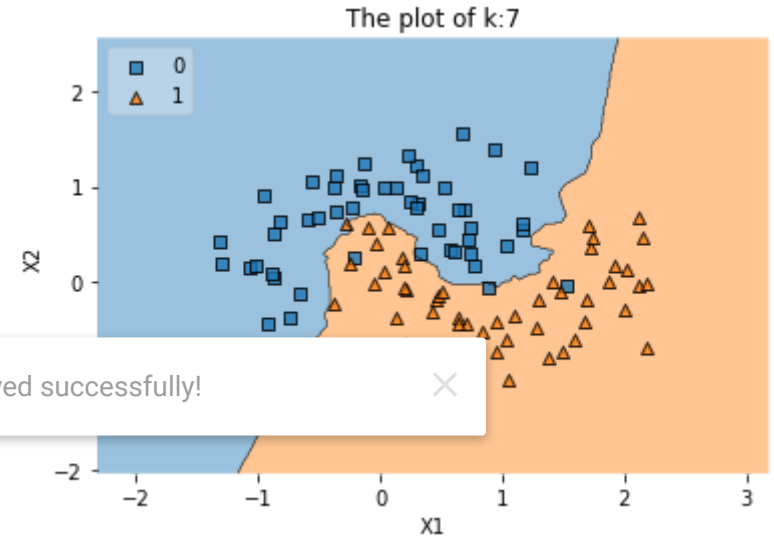
```
y_data=data_ushape["y"].astype(int).values
y_data
```

```
array([0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1,
       0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0,
       1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0])
```

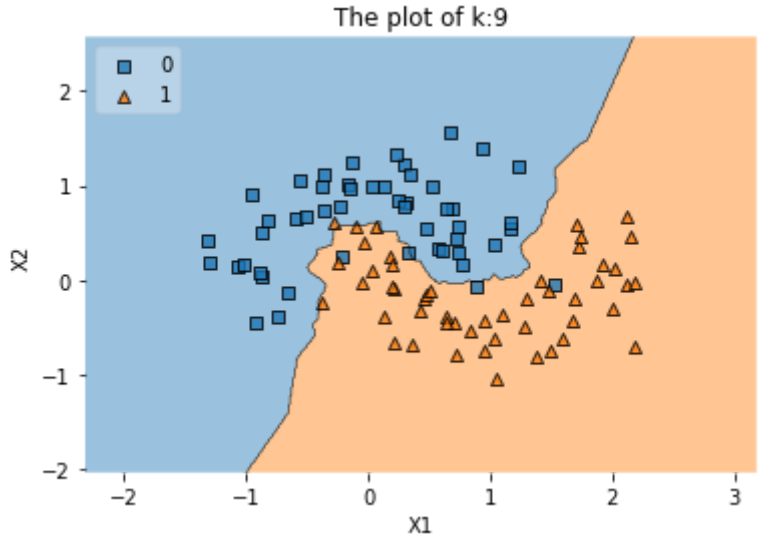
Saved successfully!







Saved successfully!



Saved successfully!

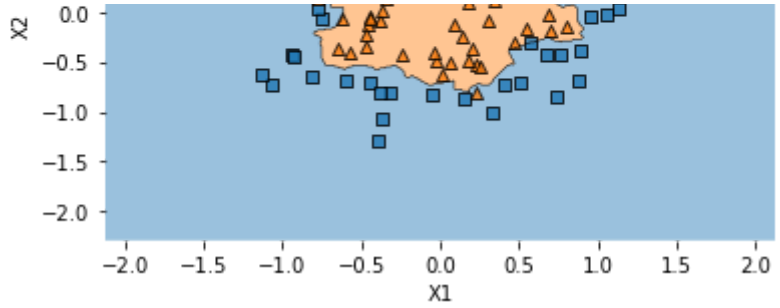
```
data_concen=pd.read_csv('/content/drive/My Drive/Colab Notebooks/knn/demo_data/2.concertricir1.csv',names=['x1','x2','y'])
```

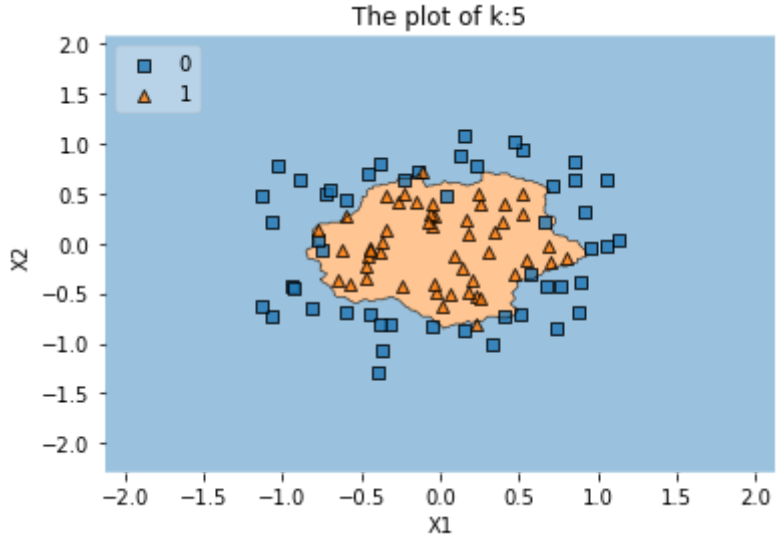
```
data_concen.shape
```

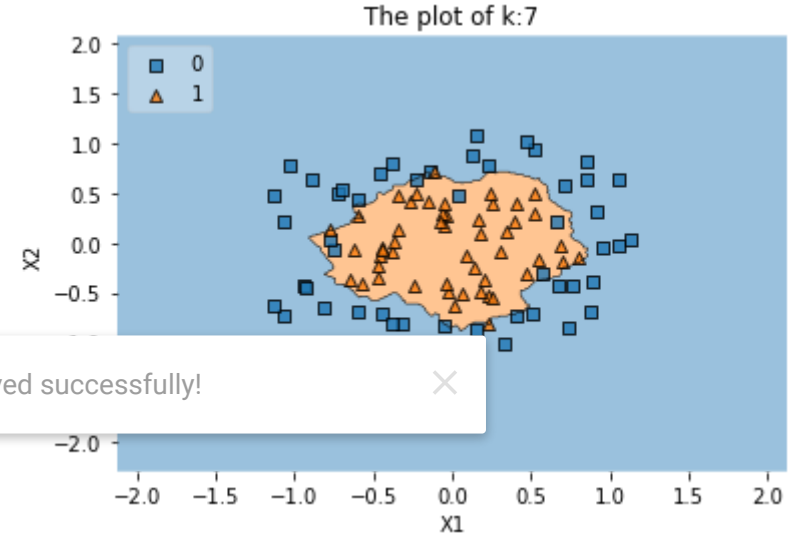
```
(100, 3)
```

```
for i in [3,5,7,11]:
  knn(data_concen,i)
```

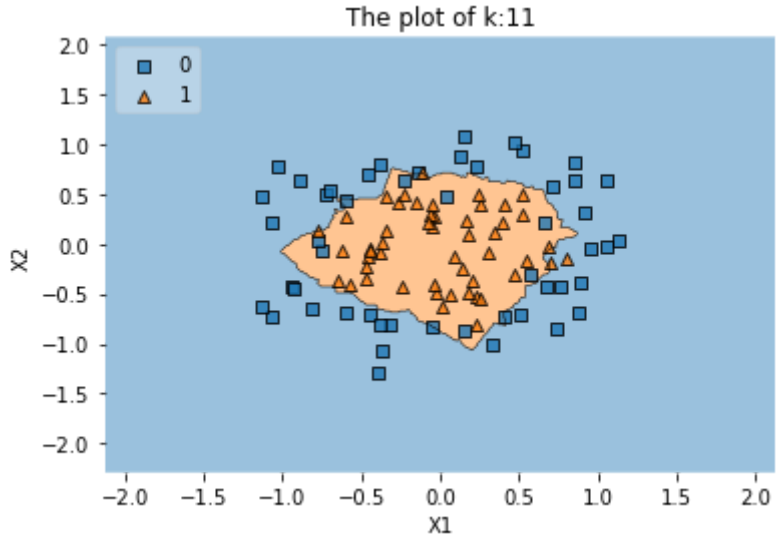
Saved successfully!







Saved successfully!




```
x = data_ushape.iloc[:,:2]
y = data_ushape.iloc[:,2]
print(x)
print(y)
```

Saved successfully!

```
          x1        x2
0   0.031595  0.986988
1   2.115098 -0.046244
2   0.882490 -0.075756
3  -0.055144 -0.037332
4   0.829545 -0.539321
..       ...       ...
95  1.699453  0.587720
96  0.218623 -0.652521
97  0.952914 -0.419766
98 -1.318500  0.423112
99 -1.296818  0.184147

[100 rows x 2 columns]
0     0.0
1     1.0
2     0.0
3     1.0
4     1.0
     ...
95    1.0
96    1.0
97    1.0
98    0.0
99    0.0
Name: y, Length: 100, dtype: float64
```

Saved successfully!

```
x_min1=x.iloc[:, 0].min()


x_max=x.iloc[:,0].max()+1
```

```
x_max
```

```
3.1813716830490244
```

```
h=0.2
a=np.arange(x_min,x_max,h)
a
```

```
array([-2.31850034, -2.11850034, -1.91850034, -1.71850034, -1.51850034,
       -1.31850034, -1.11850034, -0.91850034, -0.71850034, -0.51850034,
       -0.31850034, -0.11850034,  0.08149966,  0.28149966,  0.48149966,
        0.68149966,  0.88149966,  1.08149966,  1.28149966,  1.48149966,
        1.68149966,  1.88149966,  2.08149966,  2.28149966,  2.48149966,
        2.68149966,  2.88149966,  3.08149966])
```

```
x[0,0] = 0  ;  y[0,0] = 0
x[0,1] = 1   ; y[0,1] = 0
x[0,2] = 2    ;y[0,2] = 0
x[0,3] = 3    ;y[0,3] = 0
x[0,4] = 4    ;y[0,4] = 0
x[1,0] = 0    ;y[1,0] = 1
x[1,1] = 1    ;y[1,1] = 1
```

```
np.arange(9).reshape(3,3).ravel()
```

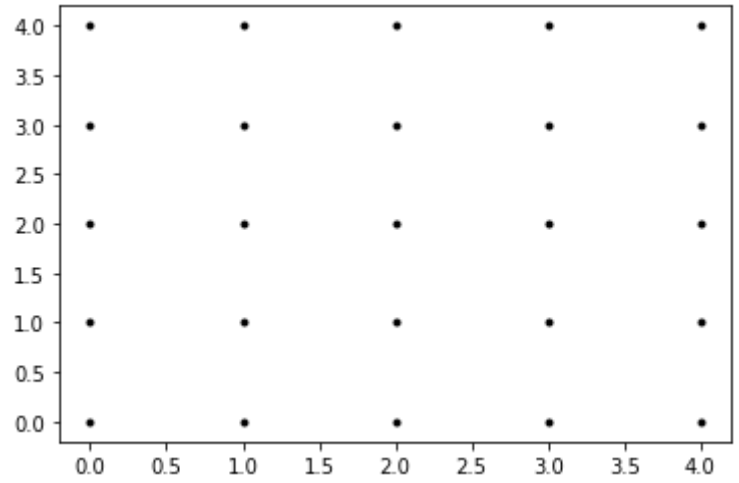
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

```
xvalues = np.array([0, 1, 2, 3, 4]);
yvalues = np.array([0, 1, 2, 3, 4]);
```

```
                                    s)
```

Saved successfully!

```
                                    k', linestyle='none')
plt.show()
```




```
def knn_cls(data,k):
  x=data[["x1","x2"]].values
  y=data["y"].astype(int).values
  h=0.02
  cmap_light=ListedColormap(["#66ccff","#00ff99"])
  cmap_bold=ListedColormap(["#ff0000","#00cc00"])
  clf=neighbors.KNeighborsClassifier(n_neighbors=k)
  clf.fit(x,y)
  x_min,y_min=x[:,0].min()-1,x[:,0].max()+1
  y_min,y_max=x[:,1].min()-1,x[:,1].max()+1
                                    in, x_max, h), np.arange(y_min, y_max, h))
  Z=clf.predict(np.c_[xx.ravel(),yy.ravel()])
  print("="*50)
  print(Z)
```

Saved successfully!

```
  print("="*50)
  z=Z.reshape(xx.shape)
  plt.pcolormesh(xx,yy,Z,cmap=cmap_light)
  sns.scatterplot(x.x1,x.x2,c=y,cmap=cmap_bold)
  plt.xlim(xx.min,xx.max)
  plt.ylim(yy.max,yy.min)
  """    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())"""
  plt.title('K value = '+str(n_neighbors))
  plt.show
```

for i in [1,3,5,7,9,11]:

```
data_outlier=pd.read_csv("/content/drive/My Drive/Colab Notebooks/knn/demo_data/5.outlier.csv",names=["x1","x2","y"])

knn_cls(data_outlier,1)
```

Saved successfully!

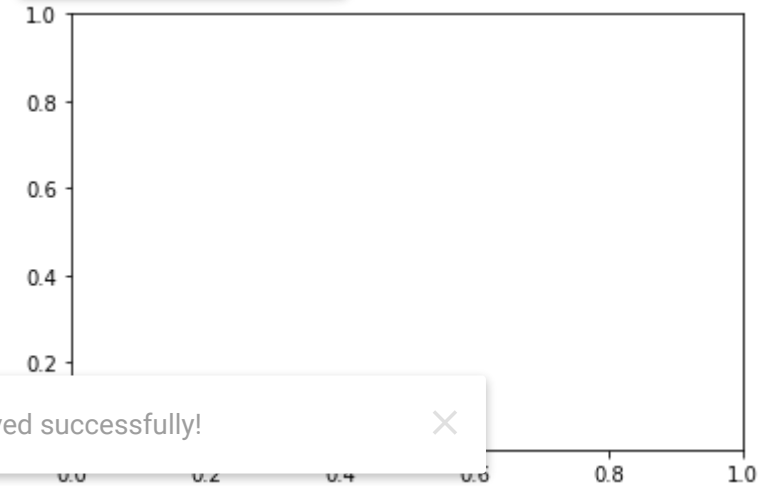
```
the shape of xx: (3344, 1708)
the shape of yy: (3344, 1708)
==================================================
[0 0 0 ... 1 1 1]
==================================================
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-64-df8bba436792> in <module>()
      1 data_outlier=pd.read_csv("/content/drive/My Drive/Colab Notebooks/knn/demo_data/5.outlier.csv",names=["x1","x2","y"])
      2
----> 3 knn_cls(data_outlier,1)
```

4 frames

```
/usr/local/lib/python3.6/dist-packages/matplotlib/axes/_axes.py in _pcolorargs(funcname, allmatch, *args)
   5732                 if isinstance(Y, np.ma.core.MaskedArray):
   5733                     Y = Y.data
-> 5734             numRows, numCols = C.shape
   5735         else:
   5736             raise TypeError(

ValueError: not enough values to unpack (expected 2, got 1)
```

SEARCH STACK OVERFLOW



Saved successfully!

```
from matplotlib.colors import ListedColormap



def knn_comparison(data, n_neighbors = 15):
    '''
    This function finds k-NN and plots the data.
    '''
    X = data[["x1","x2"]].values
    y = data["y"].astype(int).values


    # grid cell size
    h = .02
    cmap_light = ListedColormap(['#FFAAAA',  '#AAAAFF'])
    cmap_bold = ListedColormap(['#FF0000', '#0000FF'])


    # the core classifier: k-NN
    clf = neighbors.KNeighborsClassifier(n_neighbors)
    clf.fit(X, y)

    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    # we create a mesh grid (x_min,y_min) to (x_max y_max) with 0.02 grid spaces
                                    _min, x_max, h), np.arange(y_min, y_max, h))
    Saved successfully!                                 e)
                                                        e)

    # we predict the value (either 0 or 1) of each element in the grid
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
```

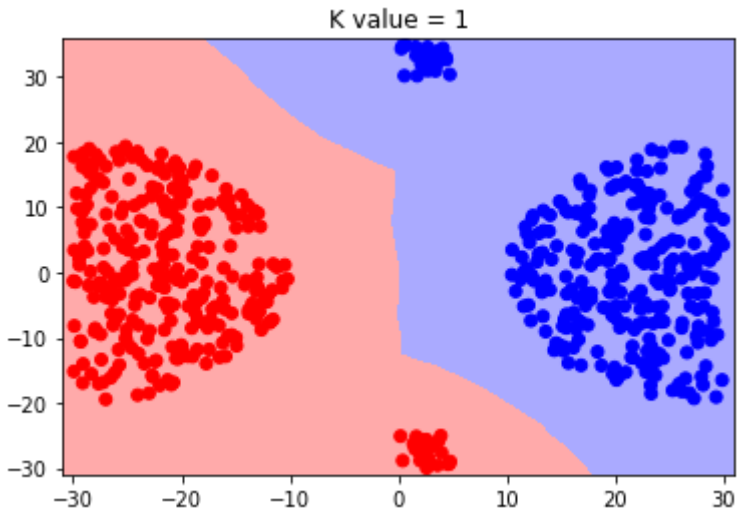
```
print("the shaape of Z",Z.shape)

# xx.ravel() will give a flatten array

# np.c_ : Translates slice objects to concatenation along the second axis.
# > np.c_[np.array([1,2,3]), np.array([4,5,6])]
# > array([[1, 4],
#          [2, 5],
#          [3, 6]])   (source: np.c_ documentation)




# convert the out back to the xx shape (we need it to plot the decission boundry)
Z = Z.reshape(xx.shape)
print("the shape of Z after  ravel",Z.shape)


# pcolormesh will plot the (xx,yy) grid with colors according to the values of Z
# it looks like decision boundry
plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

# scatter plot of with given points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold)

#defining scale on both axises
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())

# set the title
plt.title('K value = '+str(n_neighbors))
```



```
plt.show()
```

```
%%time
```

```
knn_comparison(data_outlier, 1)
```

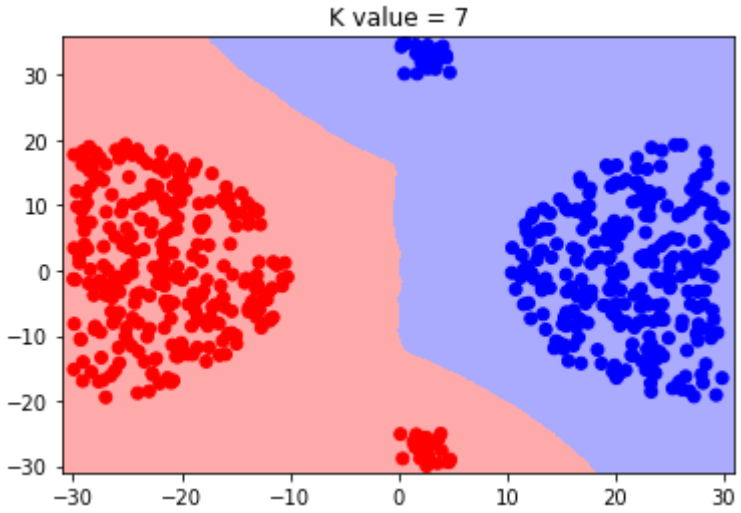



```
CPU times: user 4min 32s, sys: 445 ms, total: 4min 33s
Wall time: 4min 33s
```

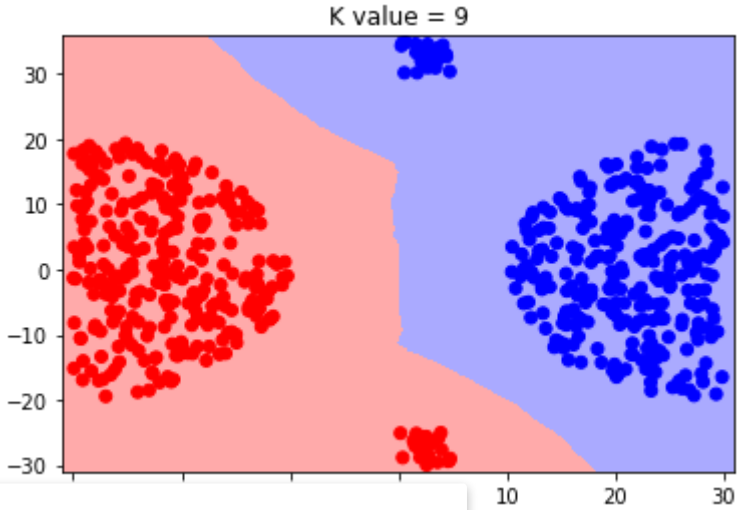
```
%%time
for i in range(3,12,2):
  knn_comparison(data_outlier,i)
```

Saved successfully!

```
the snaape of Z (10356368,)
the shape of Z after  ravel (3344, 3097)
```

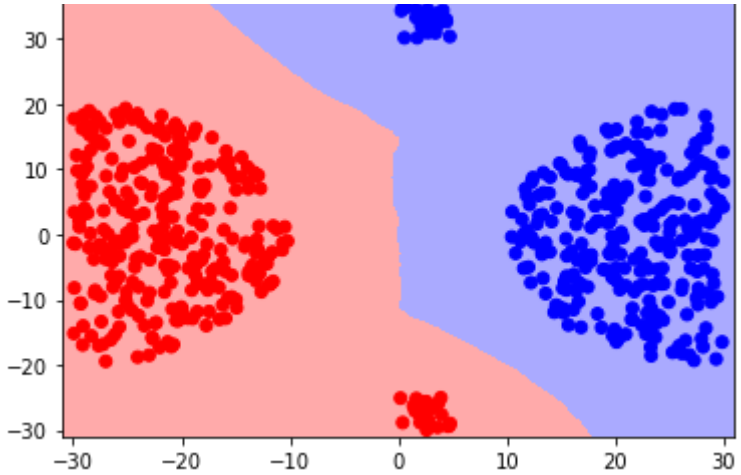



```
the shape of xx: (3344, 3097)
the shape of yy: (3344, 3097)
the shaape of Z (10356368,)
the shape of Z after  ravel (3344, 3097)
```



Saved successfully!

```
the shaape of Z (10356368,)
the shape of Z after  ravel (3344, 3097)
```




```
CPU times: user 23min 44s, sys: 4.27 s, total: 23min 48s
Wall time: 23min 48s
```

Saved successfully!

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score

from collections import Counter
from sklearn.metrics import accuracy_score


from sklearn.model_selection import cross_validate

data_2spiral=pd.read_csv("/content/drive/My Drive/Colab Notebooks/knn/demo_data/8.twospirals.csv",names=["x1","x2","y"])

data_2spiral.shape
```

```
(2000, 3)
```

Saved successfully!

```
print(X[:10])
y=data_2spiral.iloc[:,2].astype(int).values
print(y)
```

```
[[ -2.54345625 -10.81635752]
 [  9.43446606  -2.57200001]
 [  3.36864566 -10.19467054]
 [  1.34140667  -4.20414019]
 [  9.54775752  -2.22057988]
 [ -3.53329125   6.42435062]
 [  0.35424044   7.93259062]
 [  3.44791268  -0.63615042]
 [  9.24295221   1.40749598]
 [  6.47526715   5.97466035]]
[0 0 0 ... 1 1 1]
```

```
x_tr,x_test,y_tr,y_test=train_test_split(x,y,test_size=0.3,random_state=42)
print(x_tr.shape)
print(x_test.shape)
print(y_tr.shape)
print(y_test.shape)
```

```
(1400, 2)
(600, 2)
(1400,)
(600,)
```

```
x_train,x_cv,y_train,y_cv=train_test_split(x_tr,y_tr,test_size=0.33,random_state=0)
print(x_train.shape)
print(x_cv.shape)
print(y_train.shape)
print(y_cv.shape)
```

```
(938, 2)
(462, 2)
(938,)
(462,)
```

Saved successfully!

```
=50)
knn=clf.fit(x_train,y_train)
```

```
pred=knn.predict(x_cv)
```

```
accuracy=accuracy_score(y_cv,pred,normalize=True)*float(100)
accuracy
```

```
98.26839826839827
```

```
pred_test=knn.predict(x_test)
accuracy_test=accuracy_score(y_test,pred_test)*float(100)
print("the accuracy of test from train cv is :",accuracy_test)
```

```
the accuracy of test from train cv is : 97.5
```

```
lst=list(range(50))
print(lst)
neigh=list(filter(lambda x:x%2!=0,lst))
print(neigh)
c_scores=[]
len(neigh)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 3
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49]
25
```

```
for i in neigh:
  knn=KNeighborsClassifier(n_neighbors=i)
  c_sc=cross_val_score(knn,x_tr,y_tr,cv=3,scoring="accuracy")
  c_scores.append(c_sc.mean())
print(len(c_scores))
```

```
25
```

Saved successfully!

```
print(MSE)
```

```
3743754885, 0.00142908345663583 8, 0.0021428593310113264, 0.004997962828513169, 0.006427046285148896, 0.011428072529431788, 0.0171
```

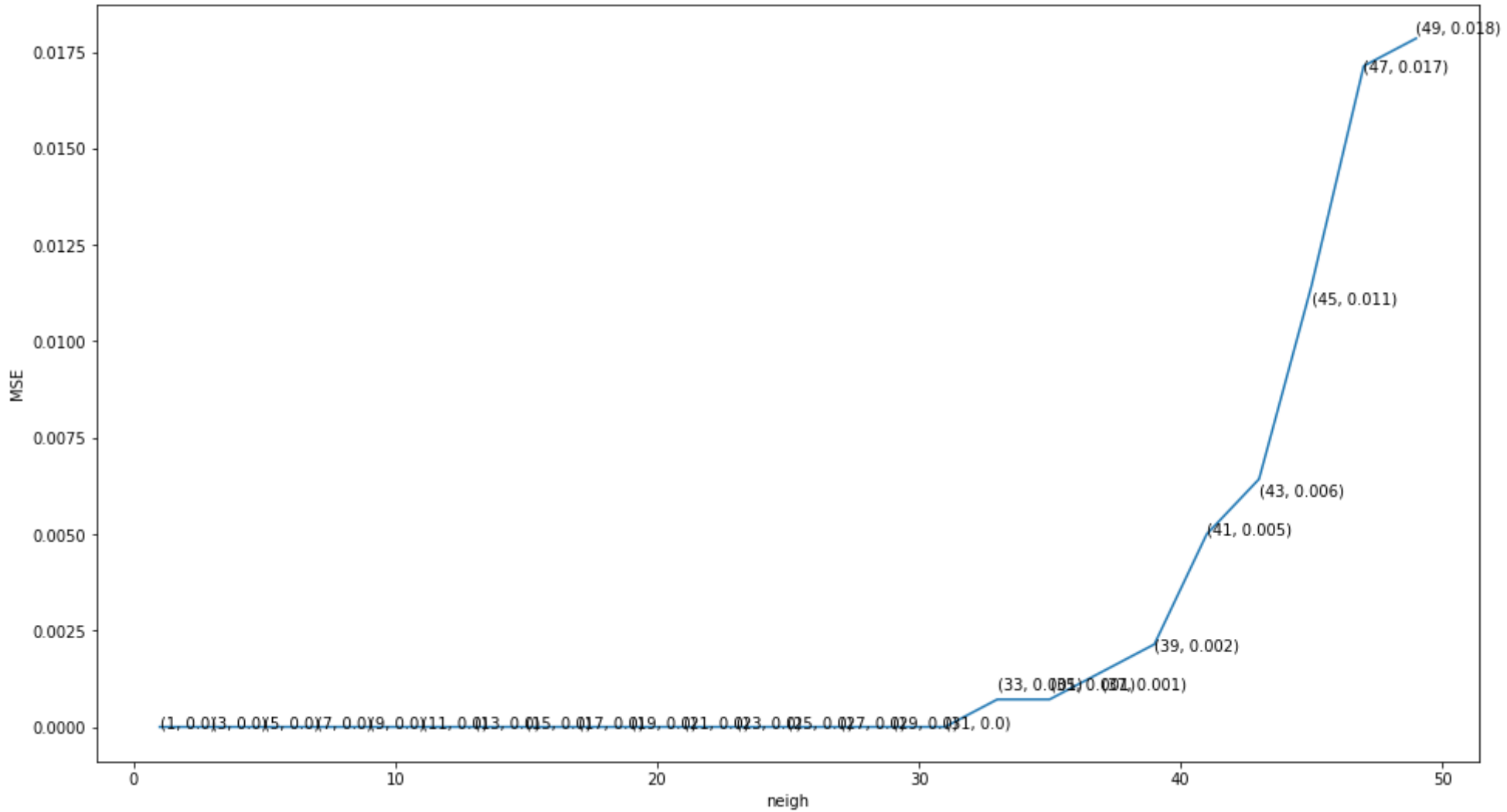
```
"""a=[1,2,3,4,5]
fruits = [40, 55, 64, 32, 16, 32]

x = a[fruits.index(min(fruits))]
x"""
optimal_k=neigh[MSE.index(min(MSE))]
print(optimal_k)
```

```
1
```

```
print(len(neigh))
print(len(MSE))
```

```
25
25
```

```
plt.figure(figsize=(16,9))
plt.plot(neigh,MSE)
for xy in zip(neigh, np.round(MSE,3)):
    plt.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.xlabel("neigh")
plt.ylabel("MSE")
plt.show()
```




```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.33)
print(x_train.shape)
```

Saved successfully!

```
(1340, 2)
(1340,)
(660, 2)
(660,)
```

```
a=[1,2,3,4,6]
fruits = [40, 55, 64, 32, 16, 32]

x = a[fruits.index(min(fruits))]
```

```
6
```

```
# creating odd list of K for KNN
myList = list(range(0,50))
neighbors = list(filter(lambda x: x % 2 != 0, myList))

# empty list that will hold cv scores
cv_scores = []

# perform 10-fold cross validation
for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, x_train, y_train, cv=3, scoring='accuracy')
    cv_scores.append(scores.mean())

# changing to misclassification error
MSE = [1 - x for x in cv_scores]

# determining best k
optimal_k = neighbors[MSE.index(min(MSE))]
print('The optimal number of neighbors is %d.' % optimal_k)

# plot misclassification error vs k
plt.plot(neighbors, MSE)
```

Saved successfully!