```
import random
import matplotlib.pyplot as plt
import math
list1=[]
res=[]
def initialize():
  for i in range(1,21):
    x=random.randrange(-60, 60)
    y=random.randrange(-60, 60)
    t1=(x,y)
    list1.append(t1)
  print(list1,end='\n')


def showPlot():
  for x,y in list1:
    plt.scatter(x, y,marker= ".",s=45)

  # x-axis label
  plt.xlabel('x - axis')
  # frequency label
  plt.ylabel('y - axis')
  # plot title
  plt.title('My scatter plot!')
  # showing legend
  plt.legend()

  # function to show the plot
  plt.show()

def caldistance(x1,y1,x2,y2):
  dist=math.sqrt(pow(x2-x1,2) + pow(y2-y1,2))
  return dist

def calKnear(x1,y1,k):
  d=0
  for x2,y2 in list1:
    d=caldistance(x1,y1,x2,y2)
    if(d<=k):
      t2=(x2,y2,d)
      res.append(t2)

def printres():
  for x1,y1,d in res:
    print(x1,' ',y1,' distance =',d)

initialize()

dx=int(input("Enter the x value from the above given list : "))
dy=int(input("Enter the y value from the above given list : " ))
k=int(input("Eneter the K value : "))
calKnear(dx,dy,k)
printres()
#1showPlot()
```

```
[(28, -50), (-53, 29), (-3, -44), (-43, 17), (44, -20), (59, 20), (14, 22), (-33, -5), (-44, 9), (4, 46), (-47, -28), (23, 11), (54
Enter the x value from the above given list : 44
Enter the y value from the above given list : -20
Eneter the K value : 3
44   -20  distance = 0.0
```

```
# Importing libraries

import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from scipy.stats import mode

from sklearn.neighbors import KNeighborsClassifier

# K Nearest Neighbors Classification

class K_Nearest_Neighbors_Classifier() :

  def __init__( self, K ) :

    self.K = K
  # Function to store training set

  def fit( self, X_train, Y_train ) :

    self.X_train = X_train

    self.Y_train = Y_train

    # no_of_training_examples, no_of_features

    self.m, self.n = X_train.shape

  # Function for prediction

  def predict( self, X_test ) :

    self.X_test = X_test

    # no_of_test_examples, no_of_features

    self.m_test, self.n = X_test.shape

    # initialize Y_predict

    Y_predict = np.zeros( self.m_test )

    for i in range( self.m_test ) :

      x = self.X_test[i]

      # find the K nearest neighbors from current test example

      neighbors = np.zeros( self.K )

      neighbors = self.find_neighbors( x )

      # most frequent class in K neighbors

      Y_predict[i] = mode( neighbors )[0][0]

    return Y_predict

  # Function to find the K nearest neighbors to current test example

  def find_neighbors( self, x ) :

    # calculate all the euclidean distances between current
    # test example x and training set X_train

    euclidean_distances = np.zeros( self.m )

    for i in range( self.m ) :

      d = self.euclidean( x, self.X_train[i] )

      euclidean_distances[i] = d

    # sort Y_train according to euclidean_distance_array and
    # store into Y_train_sorted

    inds = euclidean_distances.argsort()

    Y_train_sorted = self.Y_train[inds]

    return Y_train_sorted[:self.K]

  # Function to calculate euclidean distance

  def euclidean( self, x, x_train ) :

    return np.sqrt( np.sum( np.square( x - x_train ) ) )

# Driver code

def main() :

  # Importing dataset

  df = pd.read_csv( "diabetes.csv" )

  X = df.iloc[:,:-1].values
  Y = df.iloc[:,-1:].values

  # Splitting dataset into train and test set

  X_train, X_test, Y_train, Y_test = train_test_split(
  X, Y, test_size = 1/3, random_state = 0 )

  # Model training

  model = K_Nearest_Neighbors_Classifier( K = 3 )

  model.fit( X_train, Y_train )

  model1 = KNeighborsClassifier( n_neighbors = 3 )

  model1.fit( X_train, Y_train )

  # Prediction on test set

  Y_pred = model.predict( X_test )

  Y_pred1 = model1.predict( X_test )

  # measure performance

  correctly_classified = 0

  correctly_classified1 = 0

  # counter

  count = 0

  for count in range( np.size( Y_pred ) ) :

    if Y_test[count] == Y_pred[count] :

      correctly_classified = correctly_classified + 1

    if Y_test[count] == Y_pred1[count] :

      correctly_classified1 = correctly_classified1 + 1

    count = count + 1

  print( "Accuracy on test set by our model  : ", (
  correctly_classified / count ) * 100 )
  print( "Accuracy on test set by sklearn model : ", (
  correctly_classified1 / count ) * 100 )


if __name__ == "__main__" :

  main()
```