```
import pandas as pd
import numpy as np
```

```
X = [
    (2,4),
    (4,6),
    (4,2),
    (4,4),
    (6,4),
    (6,2)
]
y = ['Y','Y','Y','B','Y','B']
```

```
class KNN:
    def __init__(self,k):
        self.k=k

    def distances(self,x,y):
        return ((x[0]-y[0])**2 + (x[1]-y[1])**2)**0.5

    def get_distances(self,X,y,x):
        distances = []

        for i in range(len(X)):
            distances.append((self.distances(X[i],x),y[i]))
        distances.sort()
        distances = distances[:self.k]

        counts = {}
        for d in distances:
            try:
                counts[d[1]]+=1
            except:
                counts[d[1]]=1
        return max(counts, key = lambda i: counts[i])

    def get_distances_weighted(self,X,y,x):
        distances = []

        for i in range(len(X)):
            distances.append((self.distances(X[i],x),y[i]))
        distances.sort()
        distances = distances[:self.k]

        counts = {}
        for d in distances:
            try:
                counts[d[1]]+=(1/d[0])
            except:
                counts[d[1]]=(1/d[0])
        return max(counts, key = lambda i: counts[i])


    def get_distances_locally(self,X,y,x):
        distances = []

        for i in range(len(X)):
            distances.append((self.distances(X[i],x),y[i]))
        distances.sort()
        distances = distances[:self.k]

        counts = {}
        for d in distances:
            try:
                counts[d[1]].append(1/d[0])
            except:
                counts[d[1]]=[(1/d[0])]

        for c in counts:
            counts[c]=np.mean(counts[c])
        return max(counts, key = lambda i: counts[i])
```

```
cl=KNN(3)
```

```
cl.get_distances(X,y,(6,6))
```

```
'Y'
```

```
cl.get_distances_weighted(X,y,(6,6))
```

```
'Y'
```

```
cl.get_distances_locally(X,y,(6,6))
```

```
'Y'
```

```
class KNN:
    def __init__(self,k):
        self.k=k

    def distances(self,x,y):
        return ((x[0]-y[0])**2 + (x[1]-y[1])**2)**0.5

    def get_distances(self,X,y,x):
        distances = []

        for i in range(len(X)):
            distances.append((self.distances(X[i],x),y[i]))
        distances.sort()
        distances = distances[:self.k]
        print(distances)
        counts = {}
        for d in distances:
            try:
                counts[d[1]]+=(1/d[0])
                print(counts[d[1]])
            except:
                counts[d[1]]=(1/d[0])
                print(counts[d[1]])
        return max(counts, key = lambda i: counts[i])


c1=KNN(3)
c1.get_distances(X,y,(6,6))
```

```
[(2.0, 'Y'), (2.0, 'Y'), (2.8284271247461903, 'B')]
0.5
1.0
0.35355339059327373
'Y'
```