In [1]:

```
import matplotlib.pyplot as plt
```

In [2]:

```
import numpy as np
```

In [3]:

```
import random
```

In [4]:

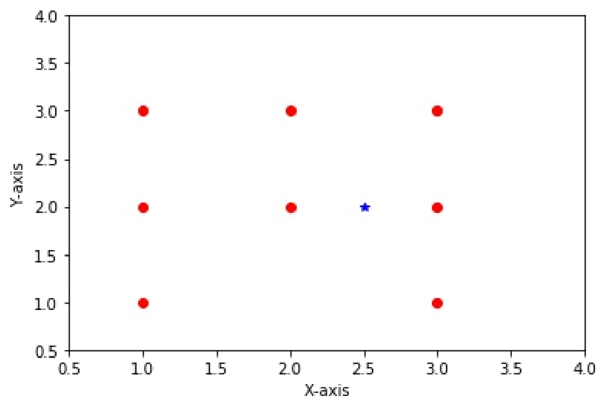
```
points = np.array([[1,1], [1,2], [1,3], [2,2], [2,3], [3,1], [3,2], [3,3]])

p = np.array([2.5, 2])
```

In [5]:

```
plt.plot(points[:,0], points[:,1], "ro")
plt.plot(p[0], p[1], "b*");
plt.axis([0.5, 4, 0.5, 4]);
plt.ylabel('Y-axis')
plt.xlabel('X-axis')
```

Out[5]:

```
Text(0.5, 0, 'X-axis')
```



In [6]:

```
points.shape
```

Out[6]:

```
(8, 2)
```

In [7]:

```
points.shape[0]
```

Out[7]:

```
8
```

In [8]:

```
len(points)
```

Out[8]:

```
8
```

In [9]:

```
def distance(p1, p2):
    return np.sqrt(np.sum(np.power(p2 -p1, 2)))
```

In [10]:

```
distance(p, points[2])
```

Out[10]:

```
1.8027756377319946
```

In [11]:

```
def majority_vote(votes):
    """This function creates a dictionary of counts and returns the key that has the highest counts
    If more than one key has the hightest counts, it picks one at random"""

    vote_counts = {}
    for vote in votes:
        if vote in vote_counts:
            vote_counts[vote] += 1
        else:
            vote_counts[vote] = 1

    winners = []
    max_count = max(vote_counts.values())

    for vote, count in vote_counts.items():
        if count == max_count:
            winners.append(vote)

    return random.choice(winners)
```

In [12]:

```
def find_nkk(p, points, k=5):
    distances = np.zeros(points.shape[0])
    for i in range(len(distances)):
        distances[i] = distance(p, points[i])
    ind = np.argsort(distances)
    return ind[:k]
```

In [13]:

```
find_nkk(p, points, k=5)
```

Out[13]:

```
array([3, 6, 4, 5, 7], dtype=int64)
```

In [14]:

```
ind = find_nkk(p, points, k=5)
```

In [ ]:

```
def knn_predict(p, points, outcomes, k=5):
    """find k newarest neighbors, then assign p to the class of the point that has the highest vote"""
    ind = find_nkk(p, points, k)
    return majority_vote(outcomes[ind])
```

In [ ]:

```
knn_predict(p, points, outcomes, k = 2)
```

In [ ]:

```
outcomes = np.array([0,0,0,0,1,1,1,1,1])
```

In [ ]:

```
knn_predict(np.array([2.5, 2.7]), points, outcomes, k=2)
```

In [ ]:

```

```