In [1]:

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

In [3]:

```
data = {
    'x': [2, 4, 4, 4, 6, 6],
    'y': [4, 2, 4, 6, 2, 4],
    'label':["orange", "orange", "blue", "orange", "blue", "orange"]
}
df = pd.DataFrame.from_dict(data)
label = 'label'
```

In [5]:

```
df.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%	max
x	6.0	4.333333	1.505545	2.0	4.0	4.0	5.5	6.0
y	6.0	3.666667	1.505545	2.0	2.5	4.0	4.0	6.0

In [6]:

```
def euclidean_distance(point_a, point_b):
    distance = 0
    for i in range(len(point_a)):
        distance = distance + (point_a[i]-point_b[i])**2
    return distance**0.5
```

In [7]:

```
def manhattan_distance(point_a, point_b):
    distance = 0
    for i in range(len(point_a)):
        distance = distance + abs(point_a - point_b)
    return distance
```

In [8]:

```
# Method to predict class based on the weighted frequency of nearest neighbors
def weighted_prediction(nearest_neighbors):
    '''
    params: {nearest_neighbors}
    - nearest_neighbors : { distance and class of k points from the dataset, nearest to inp
    - type : list - [(distance, class),(distance, class), ...]

    - returns: {prediction}
    - prediction : { Output label i.e blue or orange }
    - type: string
    '''
    label_frequency = {}
    # Calculate weight for each label
    for distance, label in nearest_neighbors:
        if int(distance) == 0:
            label_frequency[label] = sys.maxsize
            break
        if label in label_frequency:
            label_frequency[label] += 1/distance
        else: label_frequency[label] = 1/distance
    # Return label having maximum weight
    return max(label_frequency, key=label_frequency.get)
```

In [9]:

```
# Method to implement k nearest neighbors algorithm
def KNearestNeighbors(X, y, k, input_val, weight, metric):
    '''
    params: {X, y, k, input_val, weight, metric}
    - X : { Input data }
    - type : list

    - y : {Ouput label}
    - type : list

    - k : { number of neighbors }
    - type : int

    - input_val : { value for which output label is to be determined }
    - type : list - [x-coordinate, y-coordinate]

    - weight : { type of k-NN }
    - type : string

    - metric : { method for calculating distance }
    - type : string

    - returns: {prediction}
    - prediction : { Output label i.e blue or orange }
    - type: string
    '''
    # Calculate distance depending on the metrics
    distances = []
    if metric == 'manhattan':
        for i in range(len(X)):
            distances.append((manhattan_distance(X[i], input_val),y[i]))
    else:
        for i in range(len(X)):
            distances.append((euclidean_distance(X[i], input_val),y[i]))

    # Sort points according to the calculated distance
    distances.sort(key=lambda distance: distance[0])
    if weight == 'distance': # Perform prediction using distance as weight
        nearest_neighbors = distances[:k].copy()
        prediction = weighted_prediction(nearest_neighbors)
    else: # Perform prediction based on frequency of ouput label in nearest neighbors(unifo
        nearest_neighbors = distances[:k].copy()
        if int(distances[0][0]) == 0:
            prediction = distances[0][1]
            return prediction
        neighbor_classes = [label[1] for label in nearest_neighbors]
        prediction = max(set(neighbor_classes), key=neighbor_classes.count)
    return prediction
```

In [11]:

```
k = 3
metric = 'euclidean'
weight = 'uniform'
input_val = [6,6]
X=None
y=None
X = df.drop(columns=[label]).values.tolist()
y = df[label].values.tolist()
print("Predicted value: ",KNearestNeighbors(X, y, k, input_val, weight, metric))
```

```
Predicted value:  orange
```

Sklearn Implementation

In [12]:

```
X_num = df.drop(columns=[label]).to_numpy()
y_num = df[label].to_numpy()
```

In [16]:

```
knn_classifier = KNeighborsClassifier(n_neighbors=3, weights='distance',p=2) # p=1 for manh
knn_classifier.fit(X_num, y_num)
print("Predicted value: ",knn_classifier.predict(np.array([[2.0, 4.0]]))[0])
```

```
Predicted value:  orange
```

In [ ]:

```

```