**K-NEAREST NEIGHBOUR Regression:**

**Pseudo code:**

* **Load the dataset:** Load the dataset that you want to build a k-NN regressor for. The dataset should have instances represented as rows and features represented as columns, along with corresponding numerical values for the target variable.
* **Split the dataset:** Split the dataset into a training set and a test set. This is done to evaluate the performance of the k-NN regressor on unseen data.
* **Standardize the features:** Since k-NN is a distance-based algorithm, it is important to standardize the features to have zero mean and unit variance. This is done to ensure that all features contribute equally to the distance calculation.
* **Calculate distances:** For each instance in the test set, calculate the distances between that instance and all instances in the training set. The distance metric can be Euclidean distance, Manhattan distance, or any other suitable distance metric.
* **Find k nearest neighbor :** Select the k nearest neighbors to the test instance based on the calculated distances. The value of k is a hyperparameter that needs to be tuned for optimal performance.
* **Calculate the predicted value:** Once the k nearest neighbors have been identified, calculate the predicted value for the test instance based on the target variable values of the k nearest neighbors. This can be done by taking the mean or median of the target variable values, depending on the problem at hand.
* **Evaluate the k-NN regressor:** Use the test set to evaluate the performance of the k-NN regressor. This can be done by comparing the predicted values with the actual values and calculating a performance metric such as root mean squared error (RMSE) or coefficient of determination (R-squared).
* **Tune hyperparameters:** To optimize the performance of the k-NN regressor, you may need to tune the hyperparameters such as the value of k and the distance metric used.
* **Use the k-NN regressor:** Once you have built and evaluated the k-NN regressor, you can use it to predict the values of new instances. To do this, you calculate the distances between the new instance and all instances in the training set, select the k nearest neighbors, and calculate the predicted value based on the target variable values of the k nearest neighbors.
* These are the main steps involved in building a KNN regressor model. Keep in mind that there are many variations and extensions to this basic algorithm, such as using weighted distances, handling missing values, and dealing with categorical features.

Sample code:

import numpy as np

class KNNRegressor:

def \_\_init\_\_(self, n\_neighbors=5):

self.n\_neighbors = n\_neighbors

def fit(self, X, y):

self.X\_train = X

self.y\_train = y

def predict(self, X\_test):

y\_pred = []

for i in range(X\_test.shape[0]):

distances = []

for j in range(self.X\_train.shape[0]):

distance = np.sqrt(np.sum((X\_test[i] - self.X\_train[j])\*\*2))

distances.append((distance, self.y\_train[j]))

distances.sort()

neighbors = distances[:self.n\_neighbors]

neighbors\_y = [n[1] for n in neighbors]

y\_pred.append(np.mean(neighbors\_y))

return y\_pred