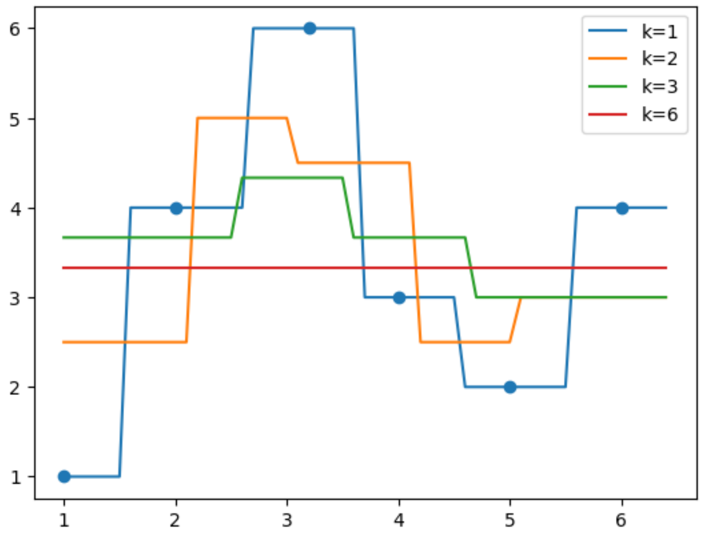
```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
import math
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

```
hw1_data_x = [[1], [2], [3.2], [4], [5], [6]]
hw1_data_y = [1, 4, 6, 3, 2, 4]

x = [[n/10] for n in range(10, 65, 1)]
n1 = KNeighborsRegressor(n_neighbors=1)
n1.fit(hw1_data_x, hw1_data_y)
y1 = n1.predict(x)
plt.plot(x, y1, label="k=1")

n2 = KNeighborsRegressor(n_neighbors=2)
n2.fit(hw1_data_x, hw1_data_y)
y2 = n2.predict(x)
plt.plot(x, y2, label="k=2")

n3 = KNeighborsRegressor(n_neighbors=3)
n3.fit(hw1_data_x, hw1_data_y)
y3 = n3.predict(x)
plt.plot(x, y3, label="k=3")

n6 = KNeighborsRegressor(n_neighbors=6)
n6.fit(hw1_data_x, hw1_data_y)
y6 = n6.predict(x)
plt.plot(x, y6, label="k=6")

plt.scatter(hw1_data_x, hw1_data_y)
plt.legend()
plt.show()
```



```
#hw2 q1
print("rmse for k = 1: " + str(math.sqrt(mean_squared_error(hw1_data_y, n1.predict(hw1_data_x)))))
print("rmse for k = 2: " + str(math.sqrt(mean_squared_error(hw1_data_y, n2.predict(hw1_data_x)))))
print("rmse for k = 3: " + str(math.sqrt(mean_squared_error(hw1_data_y, n3.predict(hw1_data_x)))))
print("rmse for k = 6: " + str(math.sqrt(mean_squared_error(hw1_data_y, n6.predict(hw1_data_x)))))
#since k=1 is just the data, it it the best option here
x = [[1.25], [3.4], [4.25]]
y = [2, 5, 2.5]
print("rmse for k = 1: " + str(math.sqrt(mean_squared_error(y, n1.predict(x)))))
print("rmse for k = 2: " + str(math.sqrt(mean_squared_error(y, n2.predict(x)))))
print("rmse for k = 3: " + str(math.sqrt(mean_squared_error(y, n3.predict(x)))))
print("rmse for k = 6: " + str(math.sqrt(mean_squared_error(y, n6.predict(x)))))
#for the new data points k=2 works best
```

```
    rmse for k = 1: 0.0
    rmse for k = 2: 1.3070322617798436
```

```
rmse for k = 3: 1.4401645996461911
rmse for k = 6: 1.5986105077709065
rmse for k = 1: 0.8660254037844386
rmse for k = 2: 0.408248290463863
rmse for k = 3: 1.2360330811826103
rmse for k = 6: 1.3228756555322954
```

```
#q2
x = np.array([[1], [2], [3], [4]])
y = np.array([[1], [2], [3], [3.5]])
reg = LinearRegression().fit(x,y)
print(reg.score(x,y))
print("B1 = " + str(reg.coef_))
print("B0 = " + str(reg.intercept_))
print("r2 score = " + str(r2_score(y, reg.predict(x))))
```

```
0.9796610169491525
B1 = [[0.85]]
B0 = [0.25]
r2 score = 0.9796610169491525
```

image link to first part of question: https://drive.google.com/file/d/1Gcti-y3lYkhLnRY7s2vTuMQWWDDCBonM/view?usp=share_link

```
#q3
#a)one hot encoding is a way to describe categorical data qualitatively
#by assigning that label a one in a array of 1/0s where all other values
#are 0
#b)i)Zipcode numbers are not quantitative so we could should use one hot encoding
#ii)The price of a house is a quantitative variable so we do not use OHE
#iii)The city where a house is located is a qualitative data point so we use OHE
#iv)The name of the home owner is also a qualitative value so we use OHE
#v)The year the house was built is a quantitative value so we do not use OHE
```

```
#q4
#a)This model is overfitted as it follows the training data exactly and thus would not
#be able to predict new value very well
#b)This model is a good fit as it captures the trends in the training data whithout
#capturing every value exactly
#c)This model is underfitted as it does not capture the form of the data at all
```

```
#q5
#a)True, we can solve linear regression via brute force methods but they are not recommended
#b)False, testing error is always expected to be higher than training error.
#c)True, R^2 is 1 minus the between the model residuals and the mean residuals.  Thus when
#the model performs well the ratio will be zero and R^2 will equal 1
#d)True, multilinear regression uses multiple different predictors to calculate y.
#Polynomial regression uses a single predictor to multiple different powers to
#calculate y but it treats each of those powers as a different predictor.
#e)Incorrect, the best K often depends on the data itself, but actually tends to over
#generalize and underfit as K gets larger
```

● ×