```
#implementation of k nearest neigbour regression
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


data=pd.read_csv('/content/bmd.csv')
data.head()
```

	id	age	sex	fracture	weight_kg	height_cm	medication	waiting_time
0	469	57.052768	F	no fracture	64.0	155.5	Anticonvulsant	18
1	8724	75.741225	F	no fracture	78.0	162.0	No medication	56
2	6736	70.778900	M	no fracture	73.0	170.5	No medication	10

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
data.shape
```

```
(169, 9)
```

```
data.describe()
```

	id	age	weight_kg	height_cm	waiting_time	bmd
count	169.000000	169.000000	169.000000	169.000000	169.000000	169.000000
mean	9102.556213	63.631531	64.665680	160.254438	19.739645	0.783104
std	8744.623598	12.356936	11.537171	7.928272	15.800570	0.166529
min	35.000000	35.814058	36.000000	142.000000	5.000000	0.407600
25%	2018.000000	54.424211	56.000000	154.000000	9.000000	0.670800
50%	6702.000000	63.487837	64.500000	160.500000	14.000000	0.786100
75%	17100.000000	72.080558	73.000000	166.000000	24.000000	0.888800
max	24208.000000	88.753795	96.000000	177.000000	96.000000	1.362400

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
```

```
 0   id            169 non-null    int64
 1   age           169 non-null    float64
 2   sex           169 non-null    object
 3   fracture      169 non-null    object
 4   weight_kg     169 non-null    float64
 5   height_cm     169 non-null    float64
 6   medication    169 non-null    object
 7   waiting_time  169 non-null    int64
 8   bmd           169 non-null    float64
dtypes: float64(4), int64(2), object(3)
memory usage: 12.0+ KB
```

```
data['sex'].unique()
```

```
array(['F', 'M'], dtype=object)
```

```
data['fracture'].unique()
```

```
array(['no fracture', 'fracture'], dtype=object)
```

```
data['medication'].unique()
```

```
array(['Anticonvulsant', 'No medication', 'Glucocorticoids'], dtype=object)
```

```
#We note that three columns sex,fracture and medication are categorical column
#sex & fracture columns are binary categorical columns
#medication is nominal categorical column
#we apply one hot encoding fore all of these categorical columns to convert them into num
```

```
#appling one hot encoding on all categorical  columns
new_data1=data.copy()
new_data1.head()
```

	id	age	sex	fracture	weight_kg	height_cm	medication	waiting_time
0	469	57.052768	F	no fracture	64.0	155.5	Anticonvulsant	18
1	8724	75.741225	F	no fracture	78.0	162.0	No medication	56
2	6736	70.778900	M	no fracture	73.0	170.5	No medication	10

Next steps: Generate code with new_data1 | View recommended plots | New interactive sheet

```
encoded_df=pd.get_dummies(new_data1,columns=['sex','fracture','medication'],prefix='is')
encoded_df.head()
```

	id	age	weight_kg	height_cm	waiting_time	bmd	is_F	is_M	is_fract
0	469	57.052768	64.0	155.5	18	0.8793	True	False	Fa
1	8724	75.741225	78.0	162.0	56	0.7946	True	False	Fa
2	6736	70.778900	73.0	170.5	10	0.9067	False	True	Fa
3	24180	78.247175	60.0	148.0	14	0.7112	True	False	Fa

Next steps: Generate code with encoded_df | View recommended plots | New interactive sheet

```
encoded_df.shape
```

```
(169, 13)
```

```
#applying stanadrd scaling (standardization)on the numerical columns
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()


col_names=encoded_df.columns
complete_data=pd.DataFrame(scaler.fit_transform(encoded_df),columns=col_names)
complete_data.head()
```

	id	age	weight_kg	height_cm	waiting_time	bmd	is_F	i
0	-0.990233	-0.533977	-0.057870	-0.601464	-0.110427	0.579369	1.017912	-1.017
1	-0.043419	0.982904	1.159205	0.220824	2.301696	0.069237	1.017912	-1.017
2	-0.271434	0.580128	0.724535	1.296122	-0.618243	0.744394	-0.982403	0.982
3	1.729320	1.186304	-0.405606	-1.550257	-0.364335	-0.433065	1.017912	-1.017

Next steps: Generate code with complete_data | View recommended plots | New interactive sheet

```
#we are asked to design k-nearest negbour to predict the bone mineral density 'bmd' targe


complete_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 13 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id                  169 non-null    float64
 1   age                 169 non-null    float64
 2   weight_kg           169 non-null    float64
```

```
 3   height_cm           169 non-null    float64
 4   waiting_time        169 non-null    float64
 5   bmd                 169 non-null    float64
 6   is_F                169 non-null    float64
 7   is_M                169 non-null    float64
 8   is_fracture         169 non-null    float64
 9   is_no fracture      169 non-null    float64
 10  is_Anticonvulsant   169 non-null    float64
 11  is_Glucocorticoids  169 non-null    float64
 12  is_No medication    169 non-null    float64
dtypes: float64(13)
memory usage: 17.3 KB
```

```
#separating x and y variables
x=complete_data.drop(columns=['id','height_cm','waiting_time','is_F','is_M','is_fracture'
y=complete_data['bmd']
```

x

	age	weight_kg
0	-0.533977	-0.057870
1	0.982904	1.159205
2	0.580128	0.724535
3	1.186304	-0.405606
4	-0.766186	-0.840276
...	...	...
164	1.164824	0.811469
165	-1.083269	-0.492540
166	-1.401896	0.202932
167	-0.717770	0.463733
168	0.516487	0.333333

169 rows × 2 columns

Next steps: Generate code with x | View recommended plots | New interactive sheet

y

	bmd
0	0.579369
1	0.069237
2	0.744394
3	-0.433065
4	0.046953
...	...
164	0.066226
165	0.084294
166	0.124045
167	0.145125
168	0.501675

169 rows × 1 columns

dtype: float64

```
#train_test splitting of data
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=54)
```

```
x_train.shape
```

```
(135, 2)
```

```
y_train.shape
```

```
(135,)
```

```
x_test.shape
```

```
(34, 2)
```

```
y_test.shape
```

```
(34,)
```

```
#KNN regression model training
from sklearn.neighbors import KNeighborsRegressor
```

```
knn=KNeighborsRegressor(n_neighbors=3) #k=3 neighbours
```

```
knn.fit(x_train,y_train)
y_pred=knn.predict(x_test)
y_pred
```

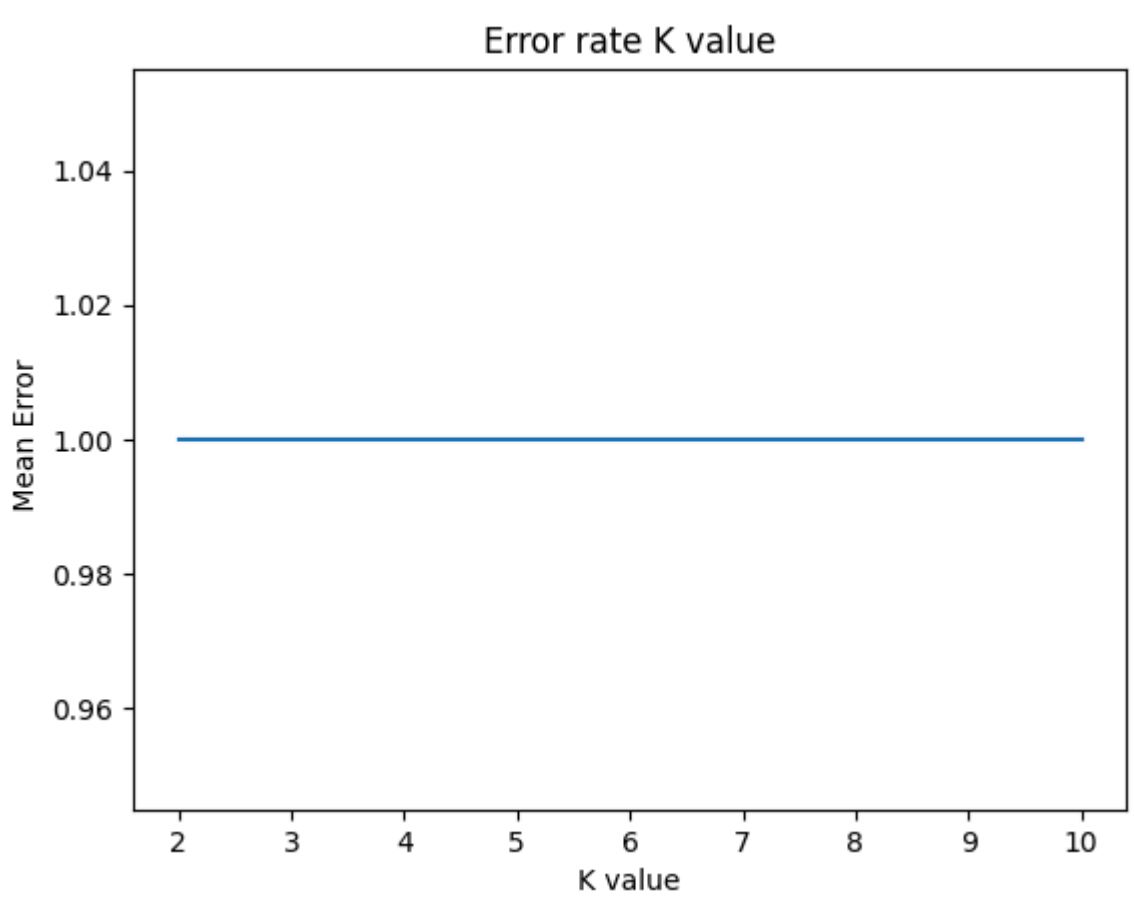
```
array([ 1.02425404, -1.33588417, -0.18351988,  0.85360797,  0.87147584,
        0.45168581,  0.09051797, -0.95584485, -0.97050019, -1.33588417,
       -1.32945984,  1.38983861,  0.54524005,  0.79177377,  0.86464976,
       -0.26543005, -0.65610972, -0.69565949,  0.18587914, -0.8612866 ,
       -0.69565949,  0.09051797, -1.37663851, -1.58563003, -0.65610972,
        0.29328594,  0.42016646,  0.38443114, -1.54547791,  0.09373007,
        0.68918518,  0.18587914,  0.78374335,  0.76647796])
```

```
#creatinf result table data frame
result=pd.DataFrame({'Actual':y_test,'Predicted':y_pred})
result.head()
```

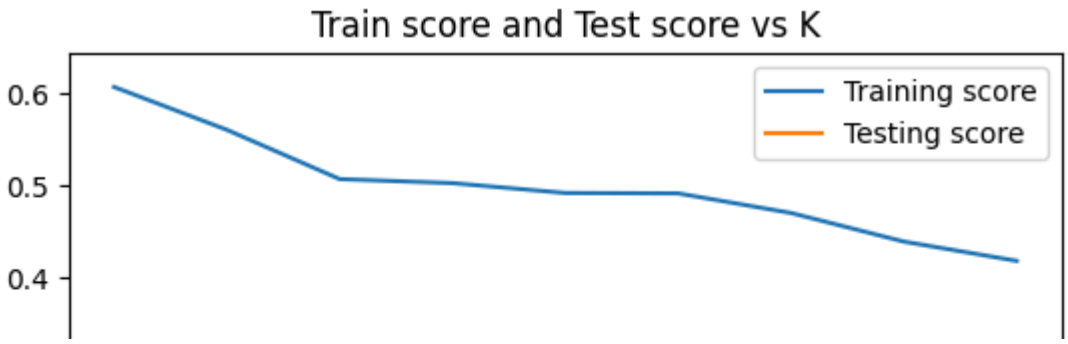
	Actual	Predicted
85	1.200321	1.024254
106	-1.029323	-1.335884
4	0.046953	-0.183520
163	-0.051821	0.853608
49	1.315357	0.871476

Next steps: Generate code with `result` View recommended plots New interactive sheet

```
#estimate best k value
error=[]
train_score=[]
test_score=[]

for i in range(2,11):
  nn=KNeighborsRegressor(n_neighbors=i)
  nn.fit(x_train,y_train)
  predicted=nn.predict(x_test)
  error.append(np.mean(predicted!=y_test)) #error is predicted not
  train_score.append(nn.score(x_train,y_train))
  test_score.append(nn.score(x_test,y_test))


plt.plot(range(2,11),error)
plt.title('Error rate K value')
plt.xlabel('K value')
plt.ylabel(' Mean Error')
plt.show()
```




```
#after so much k value , mean error value is constant still
#we oberve that mean error value is constant irrespective of constant irrespective of chn


plt.plot(range(2,11),train_score,label='Training score')
plt.plot(range(2,11),test_score,label='Testing score')
plt.legend()
plt.title('Train score and Test score vs K')
plt.xlabel('K value')
plt.ylabel('Score')
plt.legend(loc='upper right')
plt.show()
```




```
#MODEL PERFORMANCE EVOLUTION BY R^2 score
import statsmodels.api as sm #importing stats model; library for R2 adjusted score

x_col=complete_data[['age','weight_kg']]
y_col=complete_data['bmd']

x_col=sm.add_constant(x_col)
model=sm.OLS(y_col,x_col).fit() # ordinary least square method
#display adjust R-squared
print("Adjusted R^2",model.rsquared_adj)
```

```
Adjusted R^2 0.36705512988842304
```

```
from sklearn import metrics
```



```
meanAbErr=metrics.mean_absolute_error(y_test,y_pred)
meanSqErr=metrics.mean_squared_error(y_test,y_pred)
rootMeanSqErr=np.sqrt(metrics.mean_squared_error(y_test,y_pred))

print('R squared:',metrics.r2_score(y_test,y_pred))
print('Mean Absolute Error:',meanAbErr)
print('Mean Square Error:',meanSqErr)
print('Root Mean Square Error:',rootMeanSqErr)
```

```
R squared: -0.07078918369141607
Mean Absolute Error: 0.6988230670102836
Mean Square Error: 0.9251199353373702
Root Mean Square Error: 0.9618315524754687
```

```
#fitting is poor bcz of low r2 score and error is low bcz of error free model
```

Start coding or generate with AI.