Weighted KNN

Import dataset

In [18]:
```
import sklearn
from sklearn.datasets import fetch_california_housing
# as_frame=True loads the data in a dataframe format, with other metadata besides i
california_housing = fetch_california_housing(as_frame=True)
# Select only the dataframe part and assign it to the df variable
df = california_housing.frame
```

In [19]:
```
import pandas as pd
df.head()
```

Out[19]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Med
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	

Preprocessing Data for KNN Regression

In [20]:
```
y = df['MedHouseVal']
X = df.drop(['MedHouseVal'], axis = 1)
```

In [21]:
```
# .T transposes the results, transforming rows into columns
X.describe().T
```

Out[21]:

	count	mean	std	min	25%	50%	75%
MedInc	20640.0	3.870671	1.899822	0.499900	2.563400	3.534800	4.74325
HouseAge	20640.0	28.639486	12.585558	1.000000	18.000000	29.000000	37.00000
AveRooms	20640.0	5.429000	2.474173	0.846154	4.440716	5.229129	6.05238
AveBedrms	20640.0	1.096675	0.473911	0.333333	1.006079	1.048780	1.09952
Population	20640.0	1425.476744	1132.462122	3.000000	787.000000	1166.000000	1725.00000
AveOccup	20640.0	3.070655	10.386050	0.692308	2.429741	2.818116	3.28226
Latitude	20640.0	35.631861	2.135952	32.540000	33.930000	34.260000	37.71000
Longitude	20640.0	-119.569704	2.003532	-124.350000	-121.800000	-118.490000	-118.01000

Splitting Data into Train and Test Sets

In [22]:
```
from sklearn.model_selection import train_test_split

SEED = 42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_st
```

In [23]:
```
print(len(X))       # 20640
print(len(X_train)) # 15480
print(len(X_test))  # 5160
```

```
20640
15480
5160
```

Feature Scaling for KNN Regression

In [24]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
# Fit only on X_train
scaler.fit(X_train)

# Scale both X_train and X_test
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [25]:
```
col_names=['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup',
scaled_df = pd.DataFrame(X_train, columns=col_names)
scaled_df.describe().T
```

Out[25]:

	count	mean	std	min	25%	50%	75%	max
MedInc	15480.0	2.172968e-16	1.000032	-1.774632	-0.688854	-0.175663	0.464450	5.842113
HouseAge	15480.0	-1.254954e-16	1.000032	-2.188261	-0.840224	0.032036	0.666407	1.855852
AveRooms	15480.0	-1.148163e-16	1.000032	-1.877586	-0.407008	-0.083940	0.257082	56.357392
AveBedrms	15480.0	1.239408e-16	1.000032	-1.740123	-0.205765	-0.108332	0.007435	55.925392
Population	15480.0	-7.874838e-17	1.000032	-1.246395	-0.558886	-0.227928	0.262056	29.971725
AveOccup	15480.0	2.672550e-17	1.000032	-0.201946	-0.056581	-0.024172	0.014501	103.737365
Latitude	15480.0	8.022581e-16	1.000032	-1.451215	-0.799820	-0.645172	0.971601	2.953905
Longitude	15480.0	2.169625e-15	1.000032	-2.380303	-1.106817	0.536231	0.785934	2.633738

Training and Predicting KNN Regression

In [26]:
```
from sklearn.neighbors import KNeighborsRegressor
regressor = KNeighborsRegressor(n_neighbors=5, weights="distance")
regressor.fit(X_train, y_train)
```

Out[26]:
```
KNeighborsRegressor(weights='distance')
```

In [27]:
```python
y_pred = regressor.predict(X_test)
```

Evaluating the Algorithm for KNN Regression

In [28]:
```python
from sklearn.metrics import mean_absolute_error, mean_squared_error

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)

print(f'mae: {mae}')
print(f'mse: {mse}')
print(f'rmse: {rmse}')
```

```
mae: 0.44330658993325084
mse: 0.4284245302766481
rmse: 0.6545414656663457
```

In [29]:
```python
regressor.score(X_test, y_test)
```

Out[29]:
```
0.6762253110912666
```

In [30]:
```python
y.describe()
```

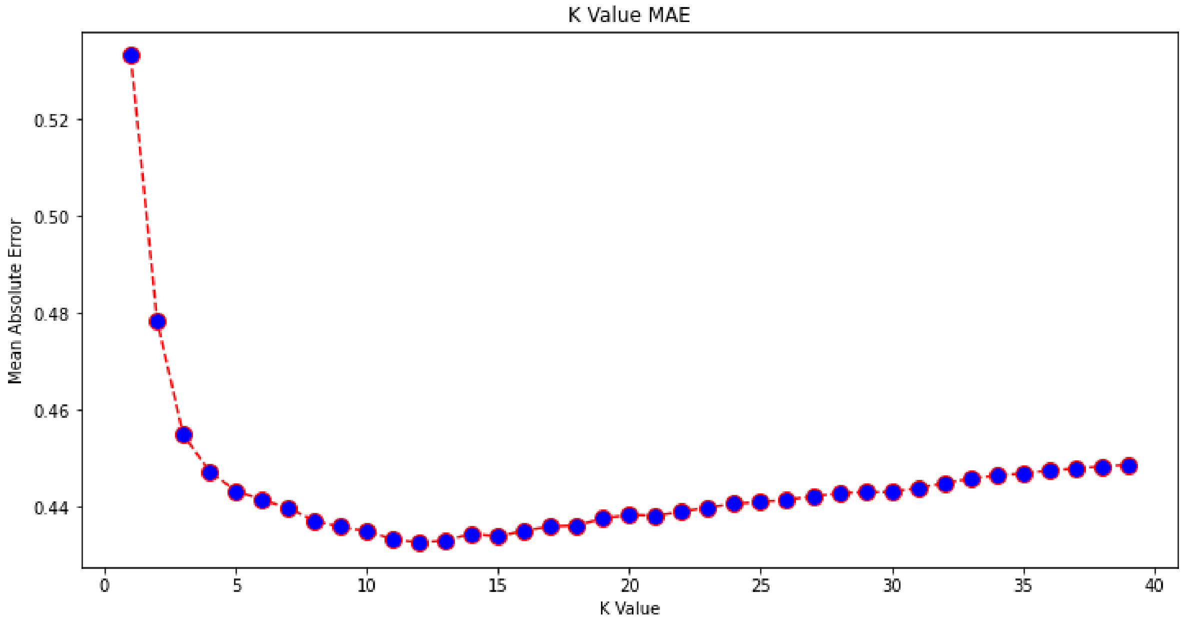
Out[30]:
```
count    20640.000000
mean         2.068558
std          1.153956
min          0.149990
25%          1.196000
50%          1.797000
75%          2.647250
max          5.000010
Name: MedHouseVal, dtype: float64
```

Finding the Best K for KNN Regression

In [31]:
```python
error = []

# Calculating MAE error for K values between 1 and 39
for i in range(1, 40):
    knn = KNeighborsRegressor(n_neighbors=i, weights="distance")
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    mae = mean_absolute_error(y_test, pred_i)
    error.append(mae)
```

In [32]:
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red',
         linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)

plt.title('K Value MAE')
plt.xlabel('K Value')
plt.ylabel('Mean Absolute Error')
```

Out[32]: Text(0, 0.5, 'Mean Absolute Error')



In [33]:
```
import numpy as np

print(min(error))
print(np.array(error).argmin())
```

```
0.43265872078512396
11
```

KNN with 12 neighbours

In [34]:
```
knn_reg12 = KNeighborsRegressor(n_neighbors=12, weights="distance")
knn_reg12.fit(X_train, y_train)
y_pred12 = knn_reg12.predict(X_test)
r2 = knn_reg12.score(X_test, y_test)

mae12 = mean_absolute_error(y_test, y_pred12)
mse12 = mean_squared_error(y_test, y_pred12)
rmse12 = mean_squared_error(y_test, y_pred12, squared=False)
print(f'r2: {r2}, \nmae: {mae12} \nmse: {mse12} \nrmse: {rmse12}')
```

```
r2: 0.6925746041555878,
mae: 0.43265872078512396
mse: 0.40679084969140783
rmse: 0.6378015754852036
```

Classification using K-Nearest Neighbors with Scikit-Learn

Preprocessing Data for Classification

In [35]:
```
# Creating 4 categories and assigning them to a MedHouseValCat column
df["MedHouseValCat"] = pd.qcut(df["MedHouseVal"], 4, retbins=False, labels=[1, 2, 3
```

In [36]:
```
y = df['MedHouseValCat']
X = df.drop(['MedHouseVal', 'MedHouseValCat'], axis = 1)
```

Splitting Data into Train and Test Sets

In [37]:
```
from sklearn.model_selection import train_test_split

SEED = 42
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_st
```

Feature Scaling for Classification

In [38]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Training and Predicting for Classification

In [39]:
```
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(weights="distance")
classifier.fit(X_train, y_train)
```

Out[39]:
```
KNeighborsClassifier(weights='distance')
```

In [40]:
```
y_pred = classifier.predict(X_test)
```

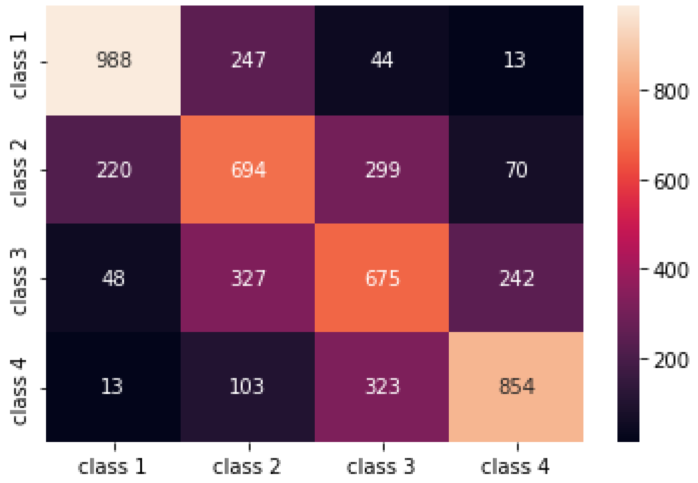
Evaluating KNN for Classification

In [41]:
```
acc =  classifier.score(X_test, y_test)
print(acc)
```

```
0.6222868217054264
```

In [42]:
```
from sklearn.metrics import classification_report, confusion_matrix
#importing Seaborn's to use the heatmap
import seaborn as sns

# Adding classes names for better interpretation
classes_names = ['class 1','class 2','class 3', 'class 4']
cm = pd.DataFrame(confusion_matrix(y_test, y_pred),
                  columns=classes_names, index = classes_names)

# Seaborn's heatmap to better visualize the confusion matrix
sns.heatmap(cm, annot=True, fmt='d');

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.78      0.76      0.77      1292
           2       0.51      0.54      0.52      1283
           3       0.50      0.52      0.51      1292
           4       0.72      0.66      0.69      1293

    accuracy                           0.62      5160
   macro avg       0.63      0.62      0.62      5160
weighted avg       0.63      0.62      0.62      5160
```

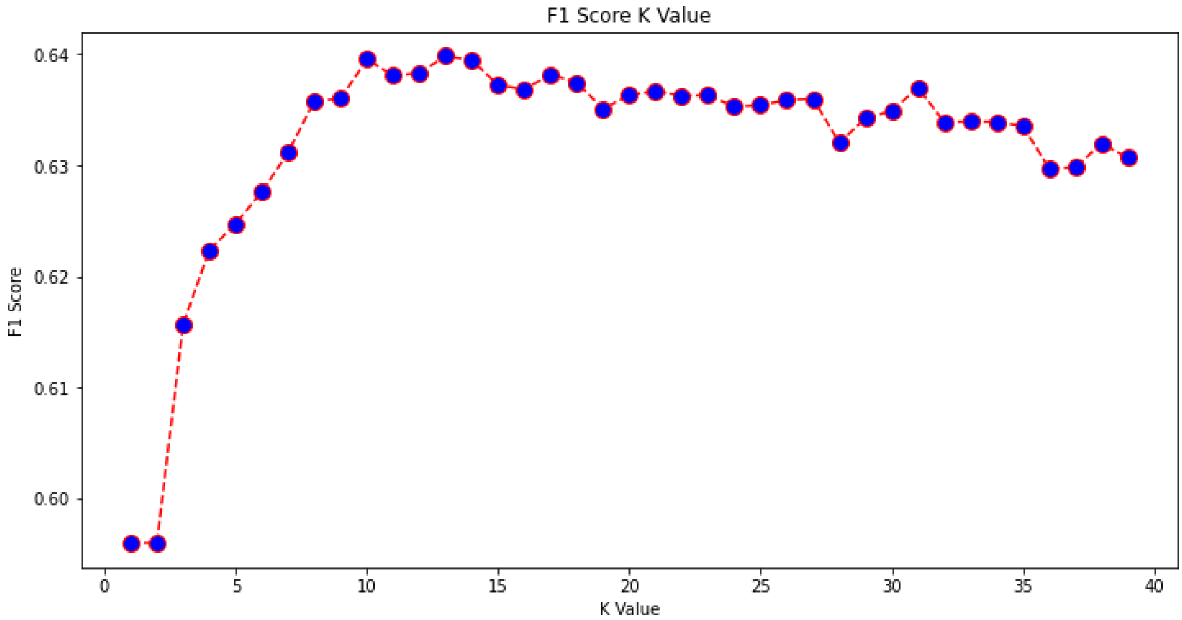


Finding the Best K for KNN Classification

In [43]:
```
from sklearn.metrics import f1_score

f1s = []

# Calculating f1 score for K values between 1 and 40
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i, weights="distance")
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    # using average='weighted' to calculate a weighted average for the 4 classes
    f1s.append(f1_score(y_test, pred_i, average='weighted'))
```

In [44]:
```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), f1s, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('F1 Score K Value')
plt.xlabel('K Value')
plt.ylabel('F1 Score')
```

Out[44]: Text(0, 0.5, 'F1 Score')



From the output, we can see that the f1-score is the highest when the value of the K is 10.

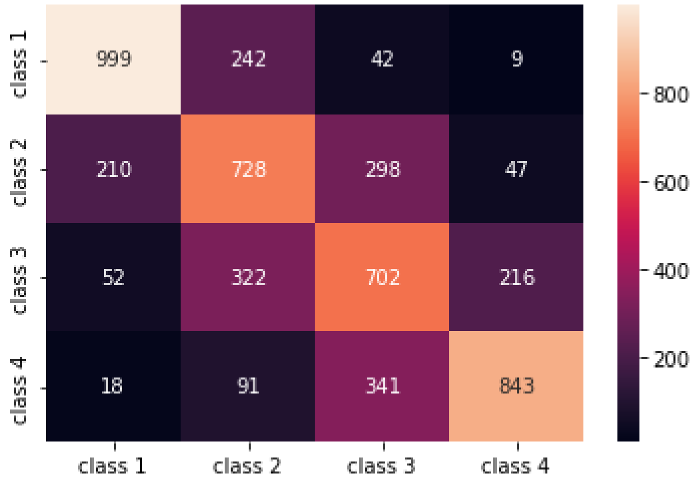
In [48]:
```
classifier15 = KNeighborsClassifier(n_neighbors=10, weights="distance")
classifier15.fit(X_train, y_train)
y_pred15 = classifier15.predict(X_test)
print(classification_report(y_test, y_pred15))
```

```
              precision    recall  f1-score   support

           1       0.79      0.77      0.78      1292
           2       0.53      0.56      0.54      1283
           3       0.51      0.55      0.53      1292
           4       0.74      0.67      0.70      1293

    accuracy                           0.64      5160
   macro avg       0.64      0.64      0.64      5160
weighted avg       0.64      0.64      0.64      5160
```

In [49]:
```
acc =  classifier.score(X_test, y_pred15)
print(acc)
```

```
0.8560077519379845
```

In [47]:
```
cm = pd.DataFrame(confusion_matrix(y_test, y_pred15),
                  columns=classes_names, index = classes_names)

sns.heatmap(cm, annot=True, fmt='d');
```


class 1
class 2
class 3
class 4
999
242
42
9
210
728
298
47
52
322
702
216
18
91
341
843
class 1
class 2
class 3
class 4
800
600
400
200