KNN

Import dataset

In [43]:
```
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 [44]:
```
import pandas as pd
df.head()
```

Out[44]:

	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 [45]:
```
y = df['MedHouseVal']
X = df.drop(['MedHouseVal'], axis = 1)
```

In [46]:
```
# .T transposes the results, transforming rows into columns
X.describe().T
```

Out[46]:

	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 [47]:
```
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 [48]:
```
print(len(X))       # 20640
print(len(X_train)) # 15480
print(len(X_test))  # 5160
```

```
20640
15480
5160
```

Feature Scaling for KNN Regression

In [49]:
```
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 [50]:
```
col_names=['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup',
scaled_df = pd.DataFrame(X_train, columns=col_names)
scaled_df.describe().T
```

Out[50]:

	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 [51]:
```
from sklearn.neighbors import KNeighborsRegressor
regressor = KNeighborsRegressor(n_neighbors=5)
regressor.fit(X_train, y_train)
```

Out[51]:
```
KNeighborsRegressor()
```

In [52]:
```
y_pred = regressor.predict(X_test)
```

Evaluating the Algorithm for KNN Regression

In [53]:
```
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: 0.4460739527131783
mse: 0.4316907430948294
rmse: 0.6570317671884894
```

In [54]:
```
regressor.score(X_test, y_test)
```

Out[54]:
```
0.6737569252627673
```

In [55]:
```
y.describe()
```

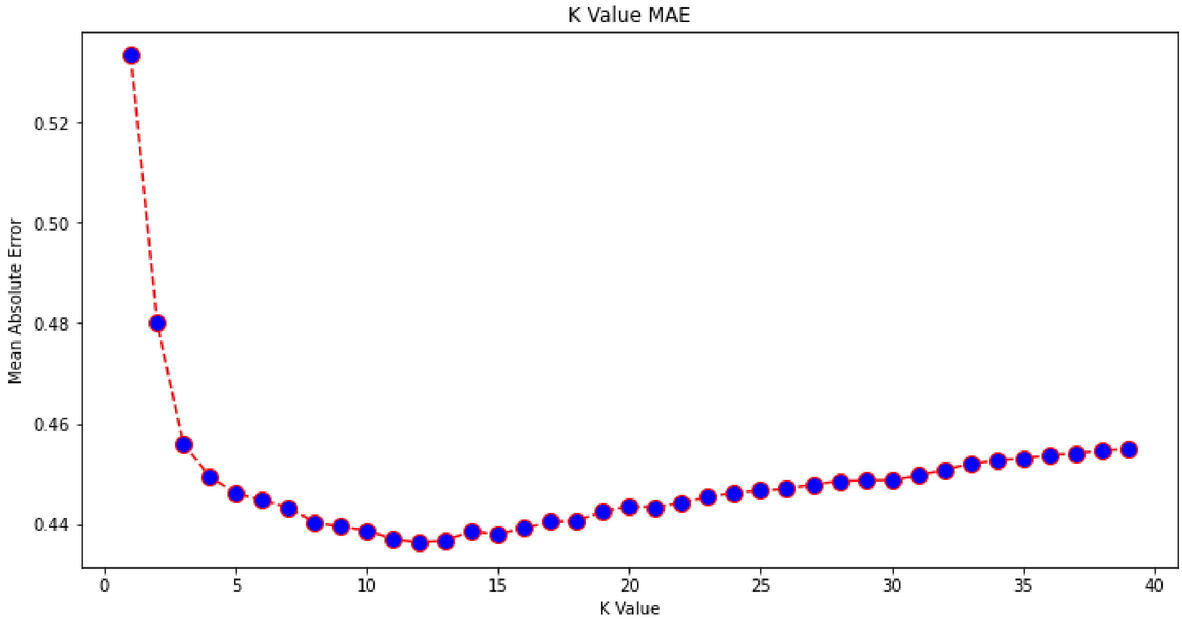
Out[55]:
```
count    20640.000000
mean         2.068558
std          1.153956
min          0.149990
25%          1.196000
50%          1.797000
75%          2.647250
max          5.000010
Name: MedHouseVal, dtype: float64
```

Finding the Best K for KNN Regression

In [56]:
```
error = []

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

In [57]:
```
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')
```

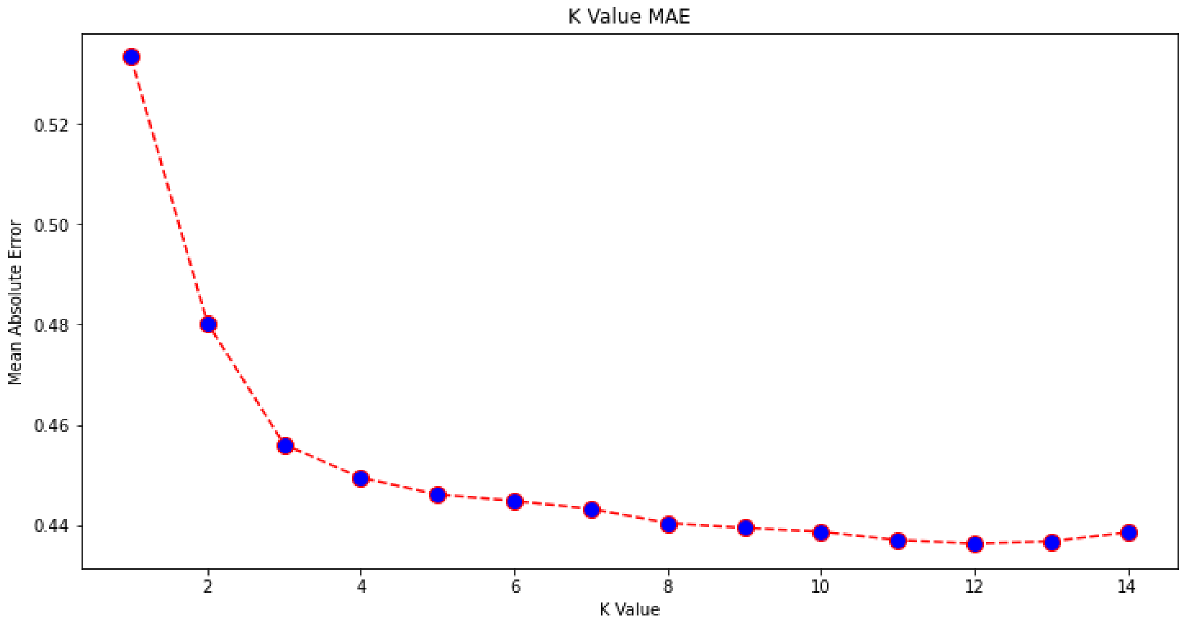
Out[57]: Text(0, 0.5, 'Mean Absolute Error')



Looking at the plot, it seems the lowest MAE value is when K is 12. Let's get a closer look at the plot to be sure by plotting less data

In [58]:
```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 15), error[:14], 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[58]: Text(0, 0.5, 'Mean Absolute Error')



In [59]:
```
import numpy as np

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

```
0.43631325936692505
11
```

KNN with 12 neighbours

In [60]:
```
knn_reg12 = KNeighborsRegressor(n_neighbors=12)
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: 0.6887495617137436,
mae: 0.43631325936692505
mse: 0.4118522151025172
rmse: 0.6417571309323467
```

Classification using K-Nearest Neighbors with Scikit-Learn

Preprocessing Data for Classification

In [61]:
```
# Creating 4 categories and assigning them to a MedHouseValCat column
df["MedHouseValCat"] = pd.qcut(df["MedHouseVal"], 4, retbins=False, labels=[1, 2, 3
```

In [62]:
```
y = df['MedHouseValCat']
X = df.drop(['MedHouseVal', 'MedHouseValCat'], axis = 1)
```

Splitting Data into Train and Test Sets

In [63]:
```
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 [64]:
```
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 [65]:
```
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier()
classifier.fit(X_train, y_train)
```

Out[65]:
```
KNeighborsClassifier()
```

In [66]:
```
y_pred = classifier.predict(X_test)
```

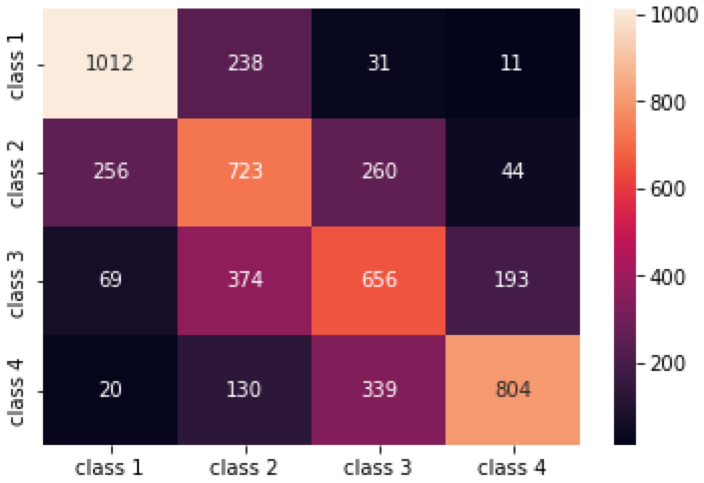
Evaluating KNN for Classification

In [67]:
```
acc =  classifier.score(X_test, y_test)
print(acc) # 0.6191860465116279
```

```
0.6191860465116279
```

In [68]:
```
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       0.75      0.78      0.76      1292
           2       0.49      0.56      0.53      1283
           3       0.51      0.51      0.51      1292
           4       0.76      0.62      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
```



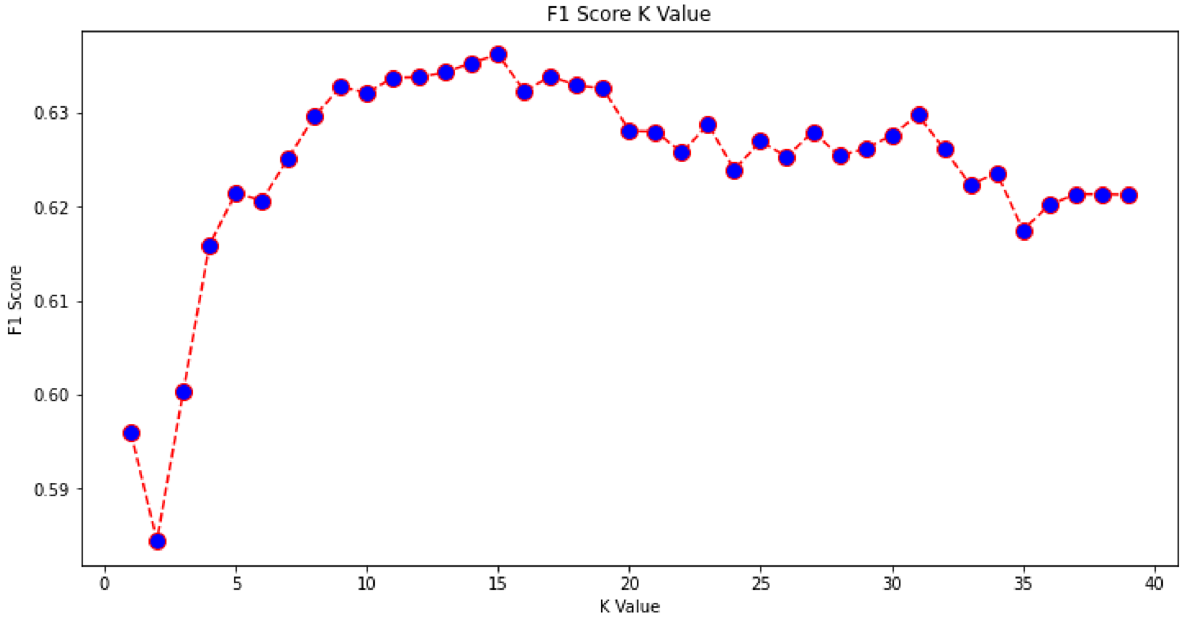
Finding the Best K for KNN Classification

In [69]:
```
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)
    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 [70]:
```
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[70]:
```
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 15.

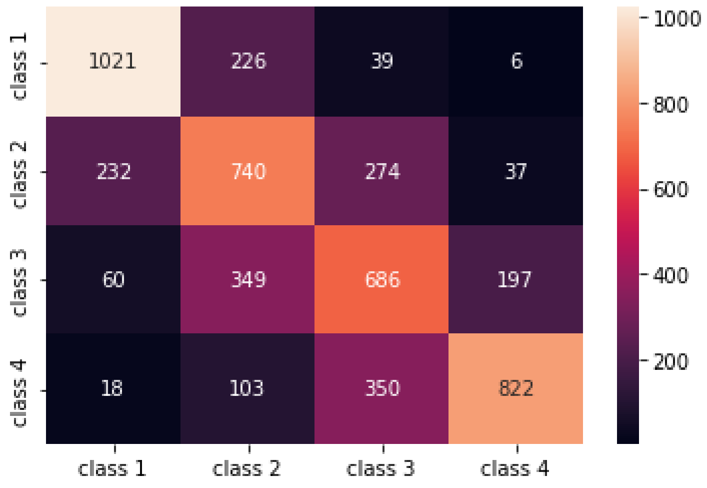
In [71]:
```
classifier15 = KNeighborsClassifier(n_neighbors=15)
classifier15.fit(X_train, y_train)
y_pred15 = classifier15.predict(X_test)
print(classification_report(y_test, y_pred15))
```

```
              precision    recall  f1-score   support

           1       0.77      0.79      0.78      1292
           2       0.52      0.58      0.55      1283
           3       0.51      0.53      0.52      1292
           4       0.77      0.64      0.70      1293

    accuracy                           0.63      5160
   macro avg       0.64      0.63      0.64      5160
weighted avg       0.64      0.63      0.64      5160
```

In [72]:
```
acc =  classifier.score(X_test, y_pred15)
print(acc)
```

```
0.7874031007751938
```

In [73]:
```
cm = pd.DataFrame(confusion_matrix(y_test, y_pred15),
                  columns=classes_names, index = classes_names)

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



Implementing KNN for Outlier Detection with Scikit-Learn

In [74]:
```
from sklearn.neighbors import NearestNeighbors

nbrs = NearestNeighbors(n_neighbors = 5)
nbrs.fit(X_train)
# Distances and indexes of the 5 neighbors
distances, indexes = nbrs.kneighbors(X_train)
```

In [75]:
```
distances[:3], distances.shape
```

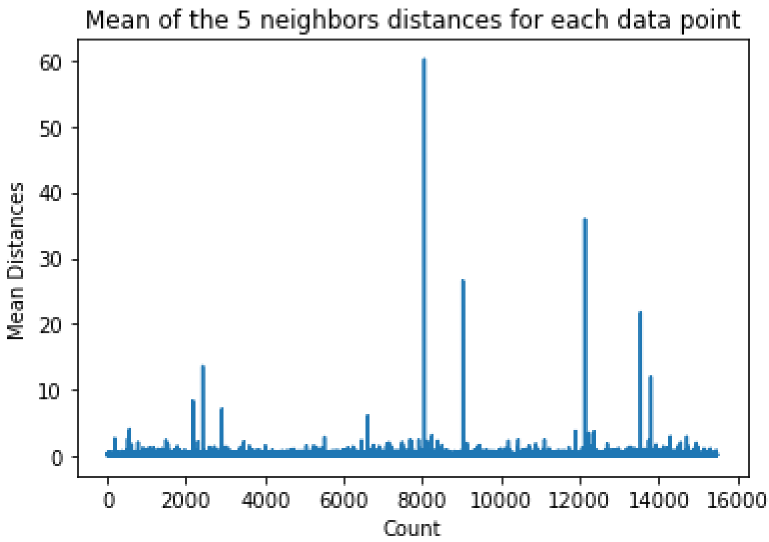
Out[75]:
```
(array([[0.        , 0.12998939, 0.15157687, 0.16543705, 0.17750354],
        [0.        , 0.25535314, 0.37100754, 0.39090243, 0.40619693],
        [0.        , 0.27149697, 0.28024623, 0.28112326, 0.30420656]]),
 (15480, 5))
```

In [76]:
```
indexes[:3], indexes[:3].shape
```

Out[76]:
```
(array([[    0,  8608, 12831,  8298,  2482],
        [    1,  4966,  5786,  8568,  6759],
        [    2, 13326, 13936,  3618,  9756]], dtype=int64),
 (3, 5))
```

In [77]:
```
dist_means = distances.mean(axis=1)
plt.plot(dist_means)
plt.title('Mean of the 5 neighbors distances for each data point')
plt.xlabel('Count')
plt.ylabel('Mean Distances')
```

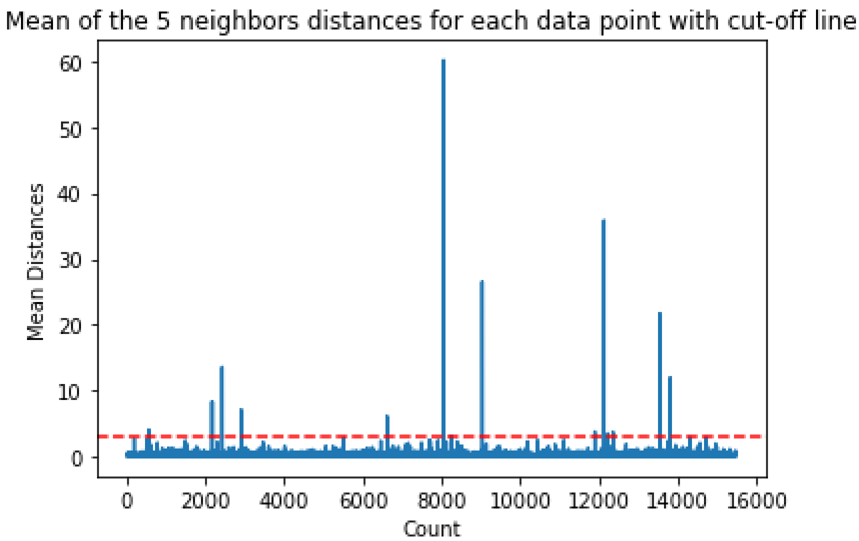
Out[77]:
```
Text(0, 0.5, 'Mean Distances')
```



mean distance is 3. Let's plot the graph again with a horizontal dotted line to be able to spot it

In [78]:
```
dist_means = distances.mean(axis=1)
plt.plot(dist_means)
plt.title('Mean of the 5 neighbors distances for each data point with cut-off line'
plt.xlabel('Count')
plt.ylabel('Mean Distances')
plt.axhline(y = 3, color = 'r', linestyle = '--')
```

Out[78]: <matplotlib.lines.Line2D at 0x20444b5a9a0>



In [79]:
```
import numpy as np

# Visually determine cutoff values > 3
outlier_index = np.where(dist_means > 3)
outlier_index
```

Out[79]:
```
(array([  564,  2167,  2415,  2902,  6607,  8047,  8243,  9029, 11892,
        12127, 12226, 12353, 13534, 13795, 14292, 14707], dtype=int64),)
```

In [80]:
```
# Filter outlier values
outlier_values = df.iloc[outlier_index]
outlier_values
```

Out[80]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
564	4.8711	27.0	5.082811	0.944793	1499.0	1.880803	37.75	-122.24
2167	2.8359	30.0	4.948357	1.001565	1660.0	2.597809	36.78	-119.83
2415	2.8250	32.0	4.784232	0.979253	761.0	3.157676	36.59	-119.44
2902	1.1875	48.0	5.492063	1.460317	129.0	2.047619	35.38	-119.02
6607	3.5164	47.0	5.970639	1.074266	1700.0	2.936097	34.18	-118.14
8047	2.7260	29.0	3.707547	1.078616	2515.0	1.977201	33.84	-118.17
8243	2.0769	17.0	3.941667	1.211111	1300.0	3.611111	33.78	-118.18
9029	6.8300	28.0	6.748744	1.080402	487.0	2.447236	34.05	-118.78
11892	2.6071	45.0	4.225806	0.903226	89.0	2.870968	33.99	-117.35
12127	4.1482	7.0	5.674957	1.106998	5595.0	3.235975	33.92	-117.25
12226	2.8125	18.0	4.962500	1.112500	239.0	2.987500	33.63	-116.92
12353	3.1493	24.0	7.307323	1.460984	1721.0	2.066026	33.81	-116.54
13534	3.7949	13.0	5.832258	1.072581	2189.0	3.530645	34.17	-117.33
13795	1.7567	8.0	4.485173	1.120264	3220.0	2.652389	34.59	-117.42
14292	2.6250	50.0	4.742236	1.049689	728.0	2.260870	32.74	-117.13
14707	3.7167	17.0	5.034130	1.051195	549.0	1.873720	32.80	-117.05