Assignment 6

U21CS089
Garvit Shah

1. Explore the impact of removal of outliers on accuracy.
2, Implement weighted KNN on the same data set and find accuracy improvement. (Hint : explore different parameters of the function KNeighborRegression()).

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

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

```
y = df['MedHouseVal']
X = df.drop(['MedHouseVal'], axis = 1)
# .T transposes the results, transforming rows into columns
X.describe().T
```

	count	mean	std	min	25%	50%	75%	max
MedInc	20640.0	3.870671	1.899822	0.499900	2.563400	3.534800	4.743250	15.000100
HouseAge	20640.0	28.639486	12.585558	1.000000	18.000000	29.000000	37.000000	52.000000
AveRooms	20640.0	5.429000	2.474173	0.846154	4.440716	5.229129	6.052381	141.909091
AveBedrms	20640.0	1.096675	0.473911	0.333333	1.006079	1.048780	1.099526	34.066667
Population	20640.0	1425.476744	1132.462122	3.000000	787.000000	1166.000000	1725.000000	35682.000000
AveOccup	20640.0	3.070655	10.386050	0.692308	2.429741	2.818116	3.282261	1243.333333
Latitude	20640.0	35.631861	2.135952	32.540000	33.930000	34.260000	37.710000	41.950000
Longitude	20640.0	-119.569704	2.003532	-124.350000	-121.800000	-118.490000	-118.010000	-114.310000

```
[3] 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_state=SEED)
```

```
print(len(X))       # 20640
print(len(X_train)) # 15480
print(len(X_test))  # 5160
```

```
20640
15480
5160
```

```
[5] 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)
```

```
[6] col_names=['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude', 'Longitude']
    scaled_df = pd.DataFrame(X_train, columns=col_names)
    scaled_df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
MedInc	15480.0	2.074711e-16	1.000032	-1.774632	-0.688854	-0.175663	0.464450	5.842113
HouseAge	15480.0	-1.232434e-16	1.000032	-2.188261	-0.840224	0.032036	0.666407	1.855852
AveRooms	15480.0	-1.620294e-16	1.000032	-1.877586	-0.407008	-0.083940	0.257082	56.357392
AveBedrms	15480.0	7.435912e-17	1.000032	-1.740123	-0.205765	-0.108332	0.007435	55.925392
Population	15480.0	-8.996536e-17	1.000032	-1.246395	-0.558886	-0.227928	0.262056	29.971725
AveOccup	15480.0	1.055716e-17	1.000032	-0.201946	-0.056581	-0.024172	0.014501	103.737365
Latitude	15480.0	7.890329e-16	1.000032	-1.451215	-0.799820	-0.645172	0.971601	2.953905
Longitude	15480.0	2.206676e-15	1.000032	-2.380303	-1.106817	0.536231	0.785934	2.633738

At neighbours 5

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
regressor = KNeighborsRegressor(n_neighbors=5)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)

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
```

```
[8] regressor.score(X_test, y_test)
```

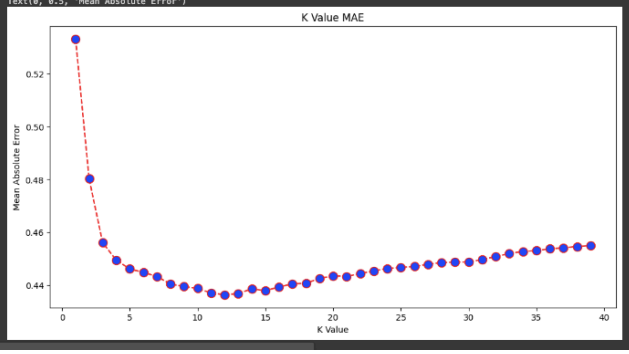
```
0.6737569252627673
```

Checking the optimum number of neighbours

```
[9] 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)
```

```
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')
```

```
Text(0, 0.5, 'Mean Absolute Error')
```




```
[11] import numpy as np

    print(min(error))              # 0.43631325936692505
    print(np.array(error).argmin()) # 11
```

```
0.43631325936692505
11
```

Hence the optimum number is 12 [11th index]

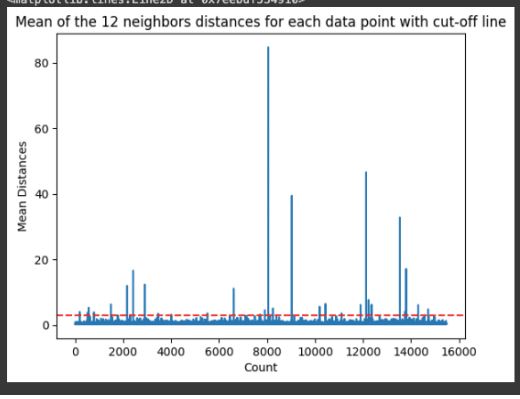
```
knn_reg12 = KNeighborsRegressor(n_neighbors=12)
knn_reg12.fit(X_train, y_train)
y_pred12 = knn_reg12.predict(X_test)
r2 = knn_reg12.score(X_test, y_test)

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
```

Removing Outliers

```
distances, indexes = knn_reg12.kneighbors(X_train)
dist_means = distances.mean(axis=1)
plt.plot(dist_means)
plt.title('Mean of the 12 neighbors distances for each data point with cut-off line')
plt.xlabel('Count')
plt.ylabel('Mean Distances')
plt.axhline(y = 3, color = 'r', linestyle = '--')
```

```
<matplotlib.lines.Line2D at 0x7eebdf534910>
```




```
import numpy as np

# Visually determine cutoff values > 3
outlier_index = np.where(dist_means > 3)
# Filter outlier values
df_no_outliers = df.drop(df.index[outlier_index])
```

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

y0 = df_no_outliers['MedHouseVal']
X0 = df_no_outliers.drop(['MedHouseVal'], axis = 1)
# .T transposes the results, transforming rows into columns
X0.describe().T


SEED = 42
X_train_out, X_test_out, y_train_out, y_test_out = train_test_split(X0, y0, test_size=0.25, random_state=SEED)


scaler_out = StandardScaler()
# Fit only on X_train
scaler_out.fit(X_train_out)

col_names=['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude', 'Longitude']
scaled_df_out = pd.DataFrame(X_train_out, columns=col_names)
scaled_df_out.describe().T


# Scale both X_train and X_test
X_train_out = scaler_out.transform(X_train_out)
X_test_out = scaler_out.transform(X_test_out)

knn_reg12_out = KNeighborsRegressor(n_neighbors=12)
knn_reg12_out.fit(X_train_out, y_train_out)
y_pred12_out = knn_reg12_out.predict(X_test_out)
r2_out = knn_reg12_out.score(X_test_out, y_test_out)

mae12_out = mean_absolute_error(y_test_out, y_pred12_out)
mse12_out = mean_squared_error(y_test_out, y_pred12_out)
rmse12_out = mean_squared_error(y_test_out, y_pred12_out, squared=False)
print(f'r2: {r2_out}, \nmae: {mae12_out} \nmse: {mse12_out} \nrmse: {rmse12_out}')
```

```
r2: 0.7072130369887215,
mae: 0.4327147497355073
mse: 0.38565014099569783
rmse: 0.6210073598563046
```

```
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: 0.637801575485203 6
```

```
knn_reg12_out = KNeighborsRegressor(n_neighbors=12, weights ="distance")
knn_reg12_out.fit(X_train_out, y_train_out)
y_pred12_out = knn_reg12_out.predict(X_test_out)
r2_out = knn_reg12_out.score(X_test_out, y_test_out)

mae12_out = mean_absolute_error(y_test_out, y_pred12_out)
mse12_out = mean_squared_error(y_test_out, y_pred12_out)
rmse12_out = mean_squared_error(y_test_out, y_pred12_out, squared=False)
print(f'r2: {r2_out}, \nmae: {mae12_out} \nmse: {mse12_out} \nrmse: {rmse12_out}')
```

```
r2: 0.713108672235521,
mae: 0.427617699116777
mse: 0.377884588387743
rmse: 0.6147231802915383
```

r2 : 0.6887495617137436

r2_w : 0.6925746041555878

r2_out : 0.7072130369887215

r2_out_w : 0.713108672235521

---

mae : 0.43631325936692505

mae_w : 0.43265872078512396

mae_out : 0.43271474977355073

mae_out_w : 0.427617699116777

---

mse : 0.4118522151025172

mse_w : 0.40679084969140783

mse_out : 0.38565014099569783

mse_out_w : 0.377884588387743

---

rmse : 0.6417571309323467

rmse_w : 0.6378015754852036

rmse_out : 0.6210073598563046

rmse_out_w : 0.6147231802915383