Department of Computer Science and Engineering (Data Science)

Lab Manual

Subject: Foundations of Data Analysis Laboratory (DJ19DSL303)

Semester: III

Experiment 8

(Outlier Detection)

NAME: Dev Patel **SAP ID: 60009200016**

BATCH: K/K1 **DATE: 22/01/22**

Aim: Perform different outlier detection methods on given data.

Theory:

Anomaly detection (aka outlier analysis) is a step in data mining that identifies data points, events, and/or observations that deviate from a dataset's normal behavior. Anomalous data can indicate critical incidents, such as a technical glitch, or potential opportunities, for instance a change in consumer behavior.

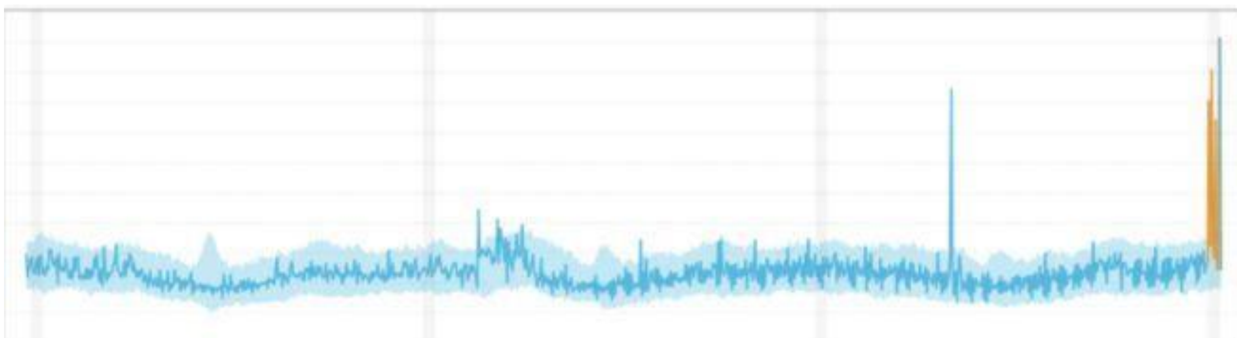
With all the analytics programs and various management software available, it's now easier than ever for companies to effectively measure every single aspect of business activity. This includes the operational performance of applications and infrastructure components as well as key performance indicators (KPIs) that evaluate the success of the organization. With millions of metrics that can be measured, companies tend to end up with quite an impressive dataset to explore the performance of their business.

Within this dataset are data patterns that represent business as usual. An unexpected change within these data patterns, or an event that does not conform to the expected data pattern, is considered an anomaly. In other words, an anomaly is a deviation from business as usual.

Generally speaking, anomalies in your business data fall into **three main categories** — global outliers, contextual outliers, and collective outliers.

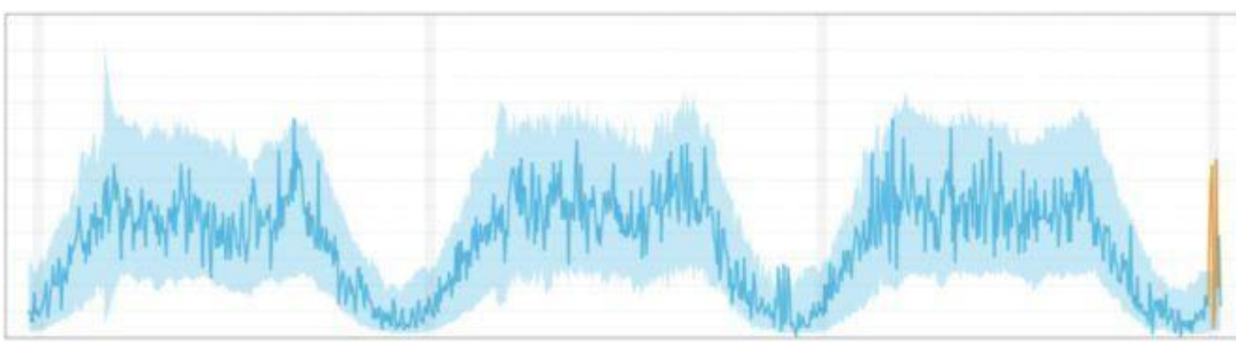
1. Global outliers

Also known as point anomalies, these outliers exist far outside the entirety of a data set.



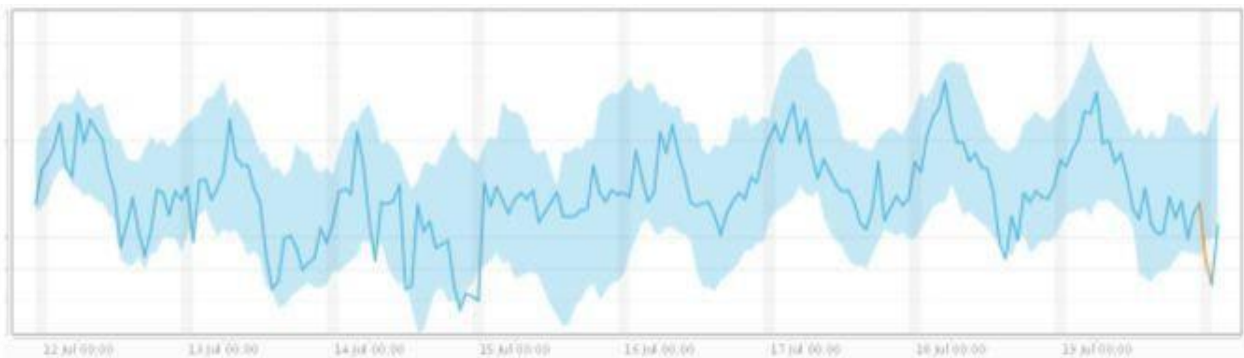
2. Contextual outliers

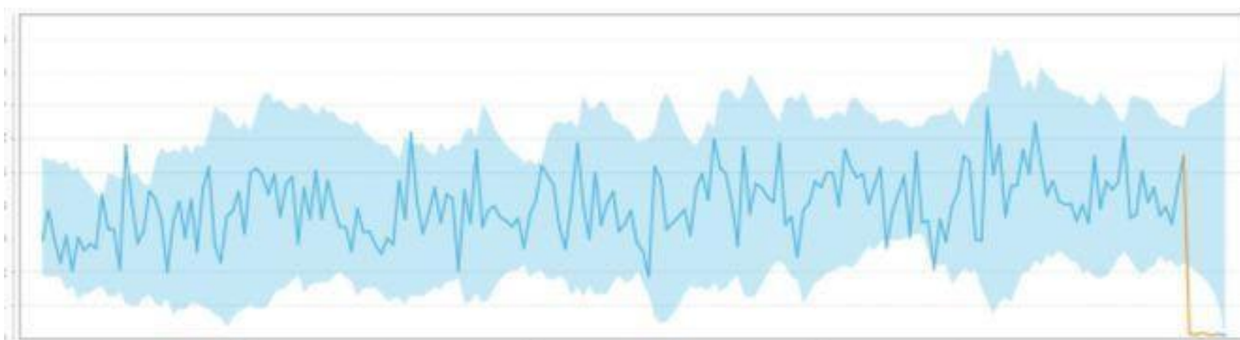
Also called conditional outliers, these anomalies have values that significantly deviate from the other data points that exist in the same context. An anomaly in the context of one dataset may not be an anomaly in another. These outliers are common in time series data because those datasets are records of specific quantities in a given period. The value exists within global expectations but may appear anomalous within certain seasonal data patterns.



3. Collective outliers

When a subset of data points within a set is anomalous to the entire dataset, those values are called collective outliers. In this category, individual values aren't anomalous globally or contextually. You start to see these types of outliers when examining distinct time series together. Individual behavior may not deviate from the normal range in a specific time series dataset. But when combined with another time series dataset, more significant anomalies become clear.





Z-score

Simply speaking, Z-score is a statistical measure that tells you how far is a data point from the rest of the dataset. In a more technical term, Z-score tells how many standard deviations away a given observation is from the mean.For example, a Z score of 2.5 means that the data point is 2.5 standard deviation far from the mean. And since it is far from the center, it’s flagged as an outlier/anomaly.

Z-score is a parametric measure and it takes two parameters — mean and standard deviation.

Once you calculate these two parameters, finding the Z-score of a data point is easy.

$$Z\text{-score} = \frac{x - mean}{Standard\ Deviation}$$

Note that mean and standard deviation are calculated for the whole dataset, whereas x represents every single data point. That means, every data point will have its own z-score, whereas mean/standard deviation remains the same everywhere.

kNN for anomaly detection

Although kNN is a supervised ML algorithm, when it comes to anomaly detection it takes an unsupervised approach. This is because there is no actual “learning” involved in the process and there is no pre-determined labeling of “outlier” or “not-outlier” in the dataset, instead, it is entirely based upon threshold values. Data scientists arbitrarily decide the cutoff values beyond which all observations are called anomalies (as we will see later). That is also why there is no train-test-split of data or an accuracy report.

Local Outlier Factor (LOF)

LOF is an unsupervised (well, semi-supervised) machine learning algorithm that uses the density of data points in the distribution as a key factor to detect outliers.
LOF compares the density of any given data point to the density of its neighbors. Since outliers come from low-density areas, the ratio will be higher for anomalous data points. As a rule of thumb, a normal data point has a LOF between 1 and 1.5 whereas anomalous observations will have much higher LOF. The higher the LOF the more likely it is an outlier. If the LOF of point X is 5, it means the average density of X’s neighbors is 5 times higher than its local density.

In mathematical terms,

LOF(X)=[(LRD(1st neighbor) + LRD(2nd neighbor) + ...+ LRD(kth neighbor))/LRD(X)]/k
where LRD is Local Reachability Distance and is computed as follows. LRD(X) = 1/(sum of Reachability Distance (X, n))/k)where n is neighbors upto k

Lab Assignments to complete in this session

1. For SAT dataset show the outliers using z-score and modifier z-score.
2. For Football data show the outliers using z-score and modifier z-score.
3. Generate 8 random points and perform KNN and LOF for K=1 and K=3. Write your observations.
4.

A. Create a function `do_nn_avg_scores(obs, n_neighbors=1)` that computes outlier scores using arithmetic mean distance from the point to each of the `n_neighbors` nearest neighbors as the score.

B. Do the same thing as in part (A) to create `do_nn_harm_scores(obs, n_neighbors=1)`, where you use the harmonic mean instead of the mean. The harmonic mean of n points is defined as

$$\text{harmonic}(X_1, X_2, \ldots, X_n) = \frac{n}{(1/x_1) + (1/x_2) + \ldots + (1/x_n)} = \frac{\left(\prod X_i\right)^{1/n}}{\bar{X}} = \frac{(X_1 X_2 \cdot X_n)^{1/n}}{\bar{X}}$$

Note that `scipy.stats` contains a `hmean` function you can use.

5. This exercise refers to density-based methods (Local Outlier Factor; Section 3).
   A. Create a function do_lof_outlier_scores(obs, n_neighbors=3) that computes outlier scores using the LOF method. Recall that the values returned by sklearn's implementation are negatives of what we want.

Code:

Link to this lab's Google Colab file:

https://colab.research.google.com/drive/1wc9A13Lag25oaq-MlX_Fr38shfQ2CG3j?usp=sharing

Q1) For SAT dataset show the outliers using z-score and modifier z-score.

```
%matplotlib inline

import sys
import scipy
import scipy.stats as ss
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import random
```

```
ct_test = pd.read_csv('/content/sample_data/SAT_CT_District_Participation_2012.csv')
print(ct_test.shape)
ct_test.head()
```

```
(130, 2)
```

	District	Participation Rate
0	Ansonia	0.66
1	Avon	0.90
2	Berlin	0.81
3	Bethel	0.86
4	Bloomfield	0.81

```
mean_rate = ct_test['Participation Rate'].mean()


# ddof is the degrees of freedom correction
# in the calculation of the standard deviation;
# for population standard deviation ddof=0
stdev_rate = ct_test['Participation Rate'].std(ddof=0)
```

```
print('Mean participation rate is {:.3f}'.format(mean_rate))
print('Standard deviation is {:.3f}'.format(stdev_rate))
```

```
Mean participation rate is 0.741
Standard deviation is 0.136
```

Calculate the z-score and add the result to the dataframe.

```
zscore_rate = ss.zscore(ct_test['Participation Rate'], ddof=0)
ct_test = ct_test.assign(zscore=zscore_rate)
```

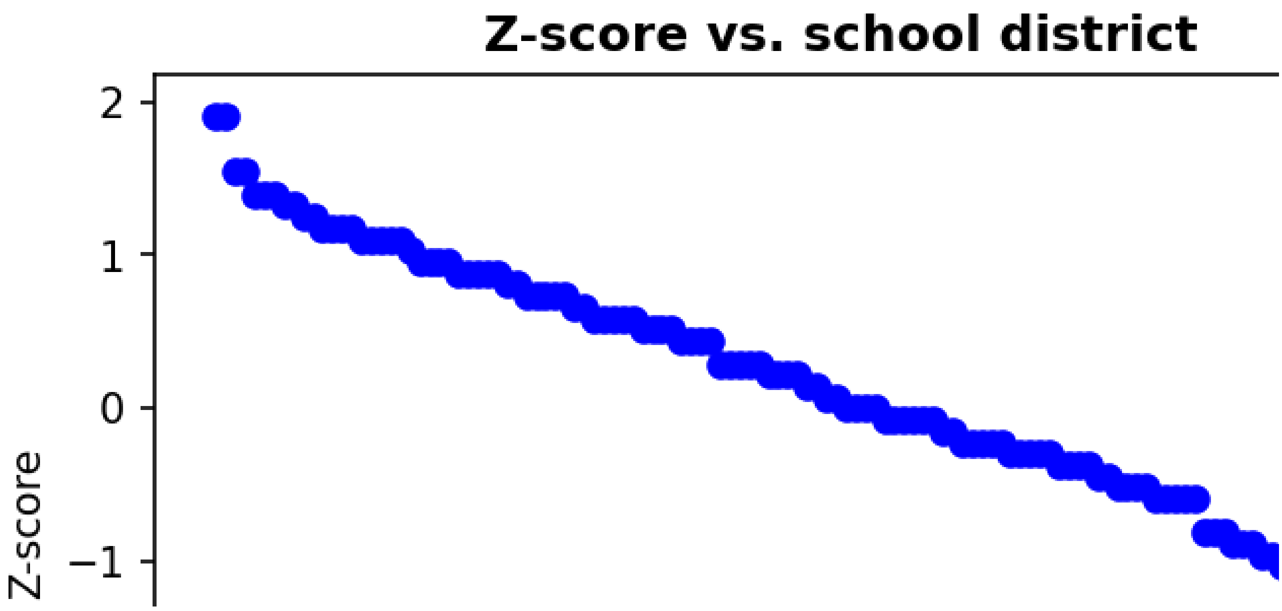
```
ct_test.head(8)
```

	District	Participation Rate	zscore
0	Ansonia	0.66	-0.593779
1	Avon	0.90	1.175649
2	Berlin	0.81	0.512113
3	Bethel	0.86	0.880744
4	Bloomfield	0.81	0.512113
5	Bolton	0.85	0.807018
6	Branford	0.77	0.217209
7	Bridgeport	0.58	-1.183589

Now identify the anomalies and plot the results.

```
def plot_anomaly(score_data, threshold):
    # Mask to plot values above and below threshold in different colors
    score_data = score_data.copy().sort_values(ascending=False).values
    ranks = np.linspace(1, len(score_data), len(score_data))
    mask_outlier = (score_data < threshold)


    plt.figure(dpi=150)
    plt.plot(ranks[~mask_outlier], score_data[~mask_outlier],'o', color='b',label='OK scho
    plt.plot(ranks[mask_outlier], score_data[mask_outlier],'o', color='r', label='anomalie
    plt.axhline(threshold,color='r',label='threshold', alpha=0.5)
    plt.legend(loc = 'lower left')
    plt.title('Z-score vs. school district', fontweight='bold')
    plt.xlabel('Ranked School district')
    plt.ylabel('Z-score')
    plt.show()
```

```
plot_anomaly(ct_test['zscore'], -2)
```



Finally, get a list of the schools that are anomalies.

```
zscore_anomalies = ct_test[(ct_test['zscore'] < -2)]
zscore_anomalies
```

	District	Participation Rate	zscore
50	New Britain	0.40	-2.510660
99	Windham	0.37	-2.731839
121	Eastern Connecticut Regional Educational Servi...	0.39	-2.584386
125	Stamford Academy	0.28	-3.395374

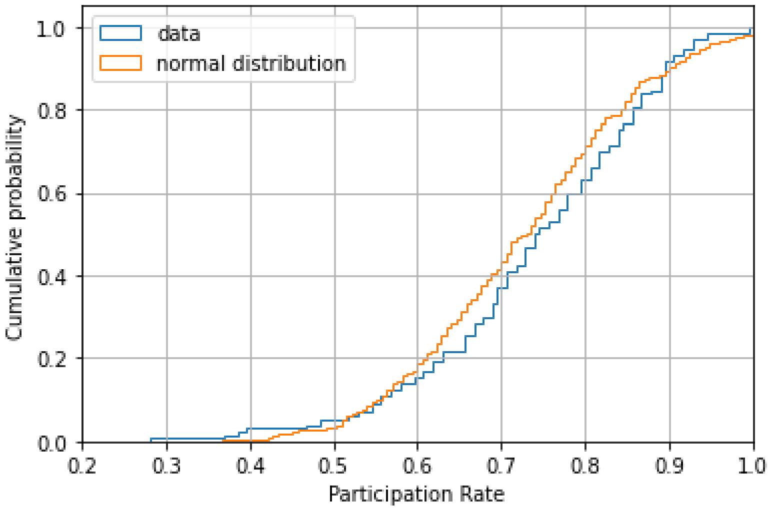
We have found our anomalies, but we still have one thing to do: check our assumption that the data can be modeled approximately as a normal distribution. If this is the case, then we have completed our test. If it isn't, then we cannot connect the z-score with probabilities as we did earlier in this notebook. First, let's bin the data and see what it looks like as a histogram.

```
nbins= 20
n_hist, bins_hist, patches_hist = plt.hist(ct_test['Participation Rate'], nbins, density=F
                           cumulative=False, linewidth=1.0, label='data')
```

This histogram has two maxima and is skewed left, so it is not likely to be normal.

We can also compare the cumulative distribution function for our data with the CDF of a normal distribution with the same mean and standard deviation of our data.

```
num_bins = 130
normal_dist = [random.gauss(mean_rate, stdev_rate) for _ in range(500)]
n, bins, patches = plt.hist(ct_test['Participation Rate'], num_bins, density=True, histtyp
                            cumulative=True, linewidth=1.0, label='data')
plt.hist(normal_dist, num_bins, density=True, histtype='step',
                            cumulative=True, linewidth=1.0, label='normal distribution')
plt.grid(True)
plt.legend(loc='upper left')
axes = plt.gca()
axes.set_xlim([0.2,1.0])
plt.xlabel('Participation Rate')
plt.ylabel('Cumulative probability')
plt.show()
```



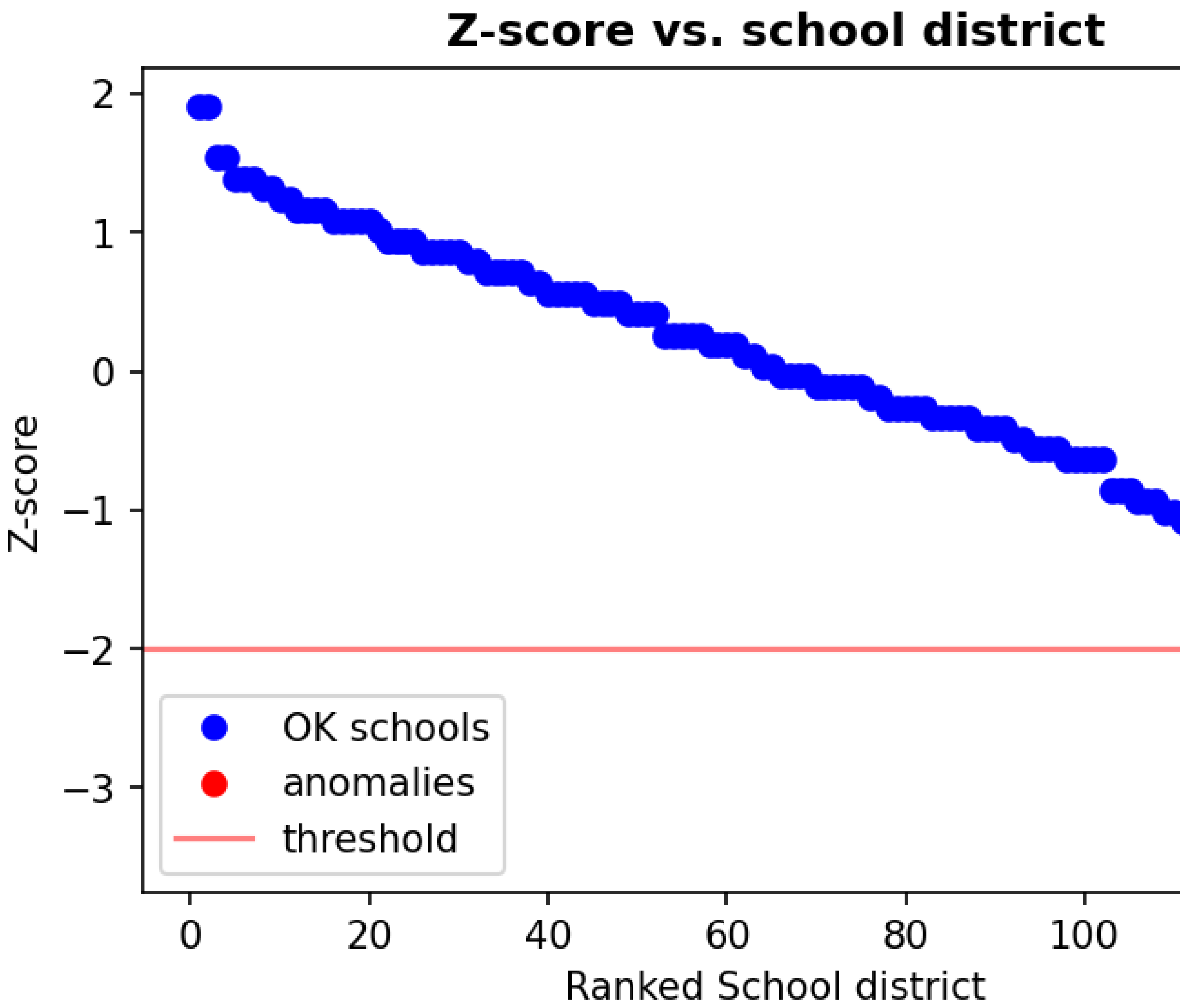

```
def modified_zscore(data, consistency_correction=1.4826):
    """
    Returns the modified z score and Median Absolute Deviation (MAD) from the scores in da
    The consistency_correction factor converts the MAD to the standard deviation for a giv
    distribution. The default value (1.4826) is the conversion factor if the underlying da
    is normally distributed
    """
    median = np.median(data)

    deviation_from_med = np.array(data) - median

    mad = np.median(np.abs(deviation_from_med))
    mod_zscore = deviation_from_med/(consistency_correction*mad)
    return mod_zscore, mad
```

```
mad_zscore, mad_participation = modified_zscore(ct_test['Participation Rate'])
z_sc_participation = ct_test.assign(zscore_median=mad_zscore)
```

```
mod_zscore_anomalies = z_sc_participation[(z_sc_participation['zscore_median'] < -2)]
mod_zscore_anomalies
```

	District	Participation Rate	zscore	zscore_
50	New Britain	0.40	-2.510660	-2
55	New London	0.47	-1.994577	-2
99	Windham	0.37	-2.731839	-2
121	Eastern Connecticut Regional Educational Servi...	0.39	-2.584386	-2
125	Stamford Academy	0.28	-3.395374	-3

```
plot_anomaly(z_sc_participation['zscore_median'], -2)
```




```
z_sc_participation['zscore_median'].describe()
```

```
count    130.000000
mean      -0.033436
std        1.020443
min       -3.484869
25%       -0.618283
50%        0.000000
75%        0.711962
max        1.911057
Name: zscore_median, dtype: float64
```

```
z_sc_participation['zscore'].describe()
```

```
count    1.300000e+02
mean    -4.208172e-16
std      1.003868e+00
min     -3.395374e+00
25%     -5.753478e-01
50%      3.289322e-02
75%      7.332920e-01
max      1.912911e+00
Name: zscore, dtype: float64
```

Q2) For Football data show the outliers using z-score and modifier z-score.

```
# First row is a header row, but the header names are cumbersome.
# Skip first row and manually label
# Encoding ensures that accents in sames are rendered properly
top_goals = pd.read_csv('/content/sample_data/world_cup_top_goal_scorers.csv',
                        encoding='utf-8',
                        names=['Year', 'Player(s)', 'Goals'], skiprows=1)
top_goals
```

	Year	Player(s)	Goals
0	1930	Guillermo Stábile	8
1	1934	Oldřich Nejedlý	5
2	1938	Leônidas	7
3	1950	Ademir	8
4	1954	Sándor Kocsis	11
5	1958	Just Fontaine	13
6	1962	Garrincha, Vavá, Leonel Sánchez, Flórián Alber...	4
7	1966	Eusébio	9
8	1970	Gerd Müller	10

Once again will start by using the z-score to identify anomalies. As we are interested in the superstars, this time we will have an upper threshold. We choose z=+2 . Above this z-score, any player will be labeled as an anomaly. As before, we calculate the mean and standard deviation.

```
mean_goals = top_goals['Goals'].mean()
stdev_goals = top_goals['Goals'].std(ddof=0)
print('Mean number of goals is {:.2f}'.format(mean_goals))
print('Standard deviation is {:.2f}'.format(stdev_goals))
```

```
Mean number of goals is 7.05
Standard deviation is 2.15
```

17 2006 Miroslav Klose 5

Calculate the z-score for each player and add the result to the dataframe.

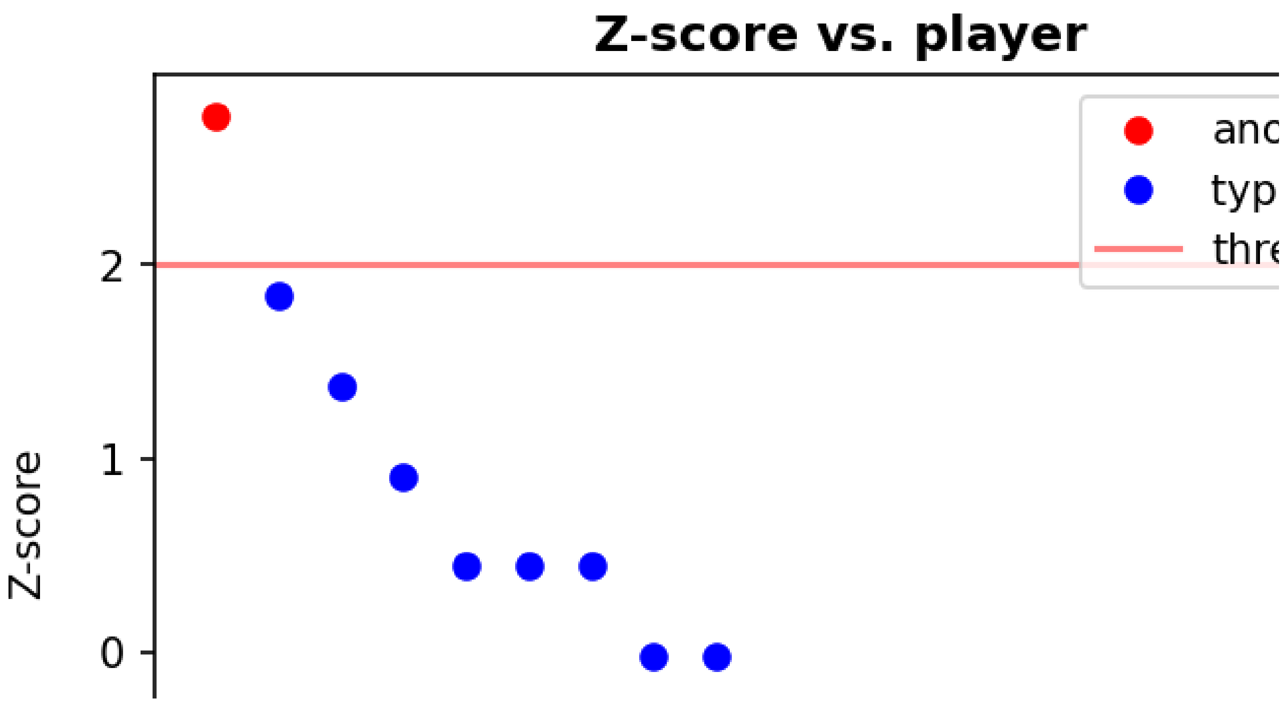
```
zscore_goals = ss.zscore(top_goals['Goals'], ddof=0)
top_goals = top_goals.assign(zscore=zscore_goals)
top_goals.head(21)
```

	Year	Player(s)	Goals	zscore
0	1930	Guillermo Stábile	8	0.443242
1	1934	Oldřich Nejedlý	5	-0.952971
2	1938	Leônidas	7	-0.022162
3	1950	Ademir	8	0.443242
4	1954	Sándor Kocsis	11	1.839455
5	1958	Just Fontaine	13	2.770264
6	1962	Garrincha, Vavá, Leonel Sánchez, Flórián Alber...	4	-1.418375
7	1966	Eusébio	9	0.908647
8	1970	Gerd Müller	10	1.374051
9	1974	Grzegorz Lato	7	-0.022162
10	1978	Mario Kempes	6	-0.487566

Now, modify the previous plotting function to display the results.

```
def plot_anomaly_goals(score_data, threshold):
    score_data = score_data.copy().sort_values(ascending=False).values
    ranks = np.linspace(1, len(score_data), len(score_data))
    mask_outlier = (score_data > threshold)

    plt.figure(dpi=150)
    plt.plot(ranks[mask_outlier], score_data[mask_outlier], 'o', color='r', label='anomali
    plt.plot(ranks[~mask_outlier], score_data[~mask_outlier], 'o', color='b', label='typic
    plt.axhline(threshold,color='r', label='threshold', alpha=0.5)
    plt.legend(loc='upper right')
    plt.title('Z-score vs. player', fontweight='bold')
    plt.xticks(np.arange(0, 21, step=2.0))
    plt.xlabel('Player Rank')
    plt.ylabel('Z-score')
    plt.show()
```

```
plot_anomaly_goals(top_goals['zscore'], 2)
```



Only one player is picked out: Just Fontaine.

```
zscore_anomalies_players = top_goals[(top_goals['zscore'] > 2)]
zscore_anomalies_players
```

	Year	Player(s)	Goals	zscore
5	1958	Just Fontaine	13	2.770264

Fontaine was indeed an amazing player, but clearly our analysis is flawed. By looking at the plot, we see that in 12 out of 21 competitions, the top goalscorer(s) scored less than the mean number of goals (7.05).

Question: What's going on?

Answer: the mean and standard deviation are themselves susceptible to the presence of anomalies. With his 13 goals, the amazing Fontaine is raising the mean so much that most players fall below it. As a result, he becomes the only anomaly. Let's repeat this analysis with the modified z-score and see what happens.

```
median_goals = np.median(top_goals['Goals'])
median_goals
```

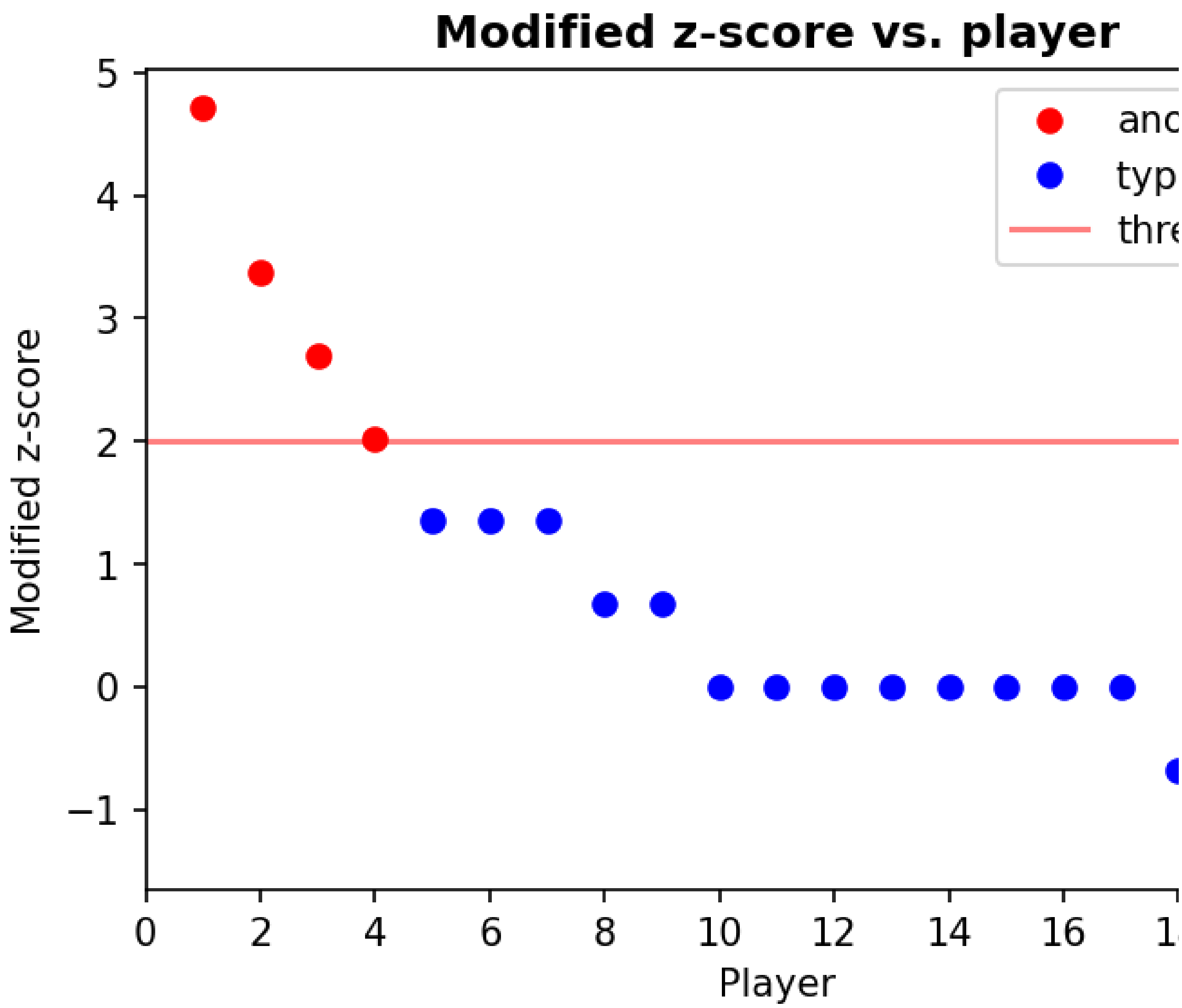
```
6.0
```

As before, compute the modified z-score for all players then plot and list the results. Note that the threshold remains the same at y=+2 .

```
mod_zscore_goals, mad_goals = modified_zscore(top_goals['Goals'])
top_goals = top_goals.assign(mod_zscore=mod_zscore_goals)
```

```
def plot_anomaly_goals_2(score_data, threshold):
    score_data = score_data.copy().sort_values(ascending=False).values
    ranks = np.linspace(1, len(score_data), len(score_data))
    mask_outliers = (score_data > threshold)

    plt.figure(dpi=150)
    plt.plot(ranks[mask_outliers], score_data[mask_outliers],'o', color='r',label='anomali
    plt.plot(ranks[~mask_outliers], score_data[~mask_outliers],'o', color='b', label='typi
    plt.axhline(threshold,color='r',label='threshold', alpha=0.5)
    plt.legend(loc = 'upper right')
    plt.title('Modified z-score vs. player', fontweight='bold')
    plt.xticks(np.arange(0, 21, step=2.0))
    plt.xlabel('Player')
    plt.ylabel('Modified z-score')
    plt.show()
```

```
plot_anomaly_goals_2(top_goals['mod_zscore'], 2)
```



```
mod_zscore_anomalies_players = top_goals[(top_goals['mod_zscore'] > 2)]
mod_zscore_anomalies_players
```

	Year	Player(s)	Goals	zscore	mod_zscore
4	1954	Sándor Kocsis	11	1.839455	3.372454
5	1958	Just Fontaine	13	2.770264	4.721435
7	1966	Eusébio	9	0.908647	2.023472
8	1970	Gerd Müller	10	1.374051	2.697963

```
print('The value of MAD is {:.2f}'.format(mad_goals))
```

```
The value of MAD is 1.00
```

and k∗MAD is 1.48, which is smaller than the standard deviation (2.05).

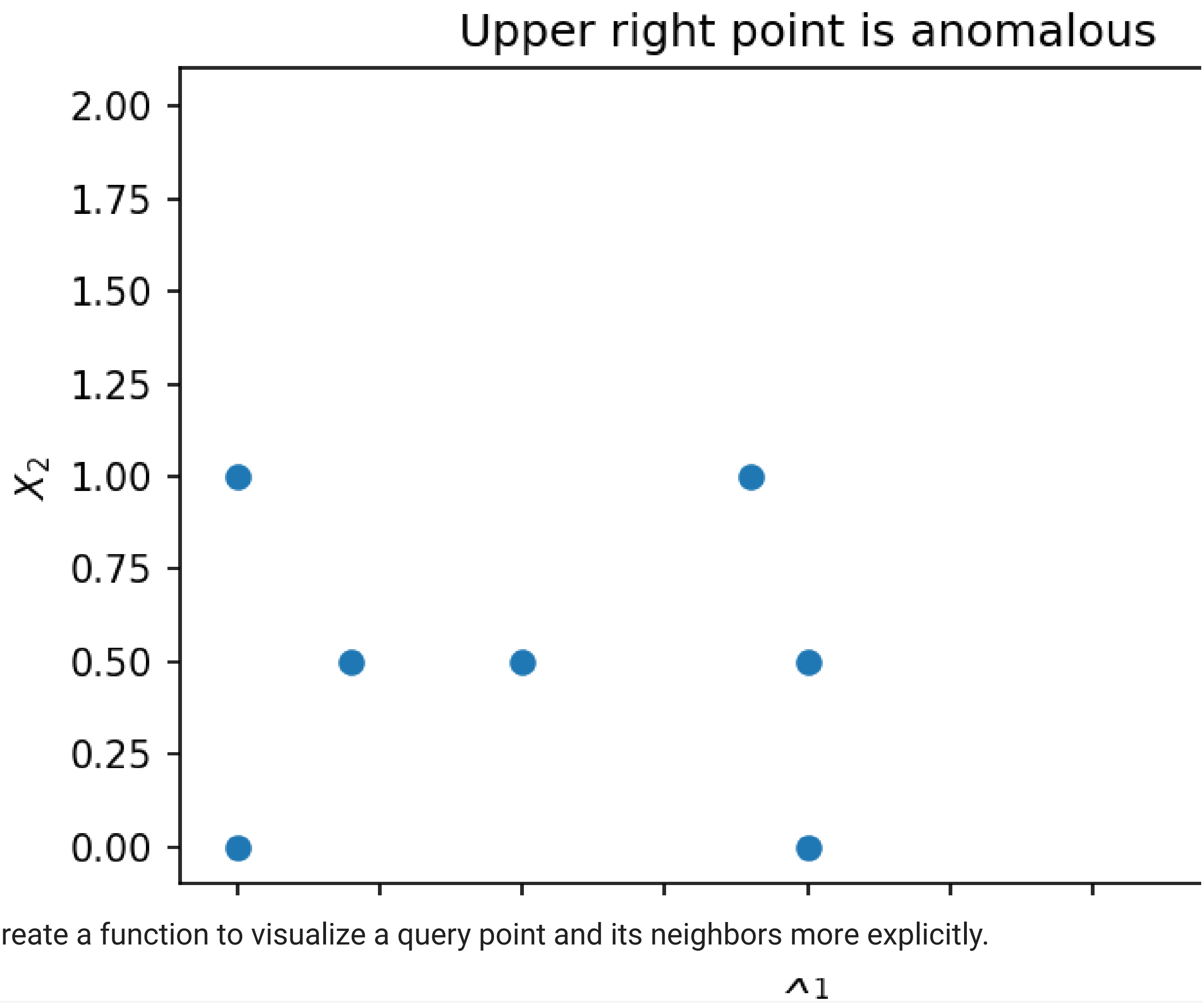
We see that the anomalies have a larger effect on the standard deviation, which depends on the square of the deviation from the mean (MAD depends linearly on the deviation).

Of course, the data for top goal scorers is not normally distributed, so we can't associated probabilities with our scores, but our analysis does show the need to think about the scoring method used with the data.

Q3) Generate 8 random points and perform KNN and LOF for K=1 and K=3. Write your observations

```
import sklearn.datasets as sk_data
import sklearn.neighbors as neighbors
import sys

np.set_printoptions(suppress=True, precision=4)
```

```
X = np.array([[0.9, 1], [0, 1], [1, 0], [0, 0], [0.5, 0.5], [0.2, 0.5], [1, 0.5], [2, 2]])

plt.figure(dpi=150)
plt.title('Upper right point is anomalous')
plt.xlabel('$X_1$')
plt.ylabel('$X_2$')
plt.plot(X[:, 0], X[:, 1], 'o');
```

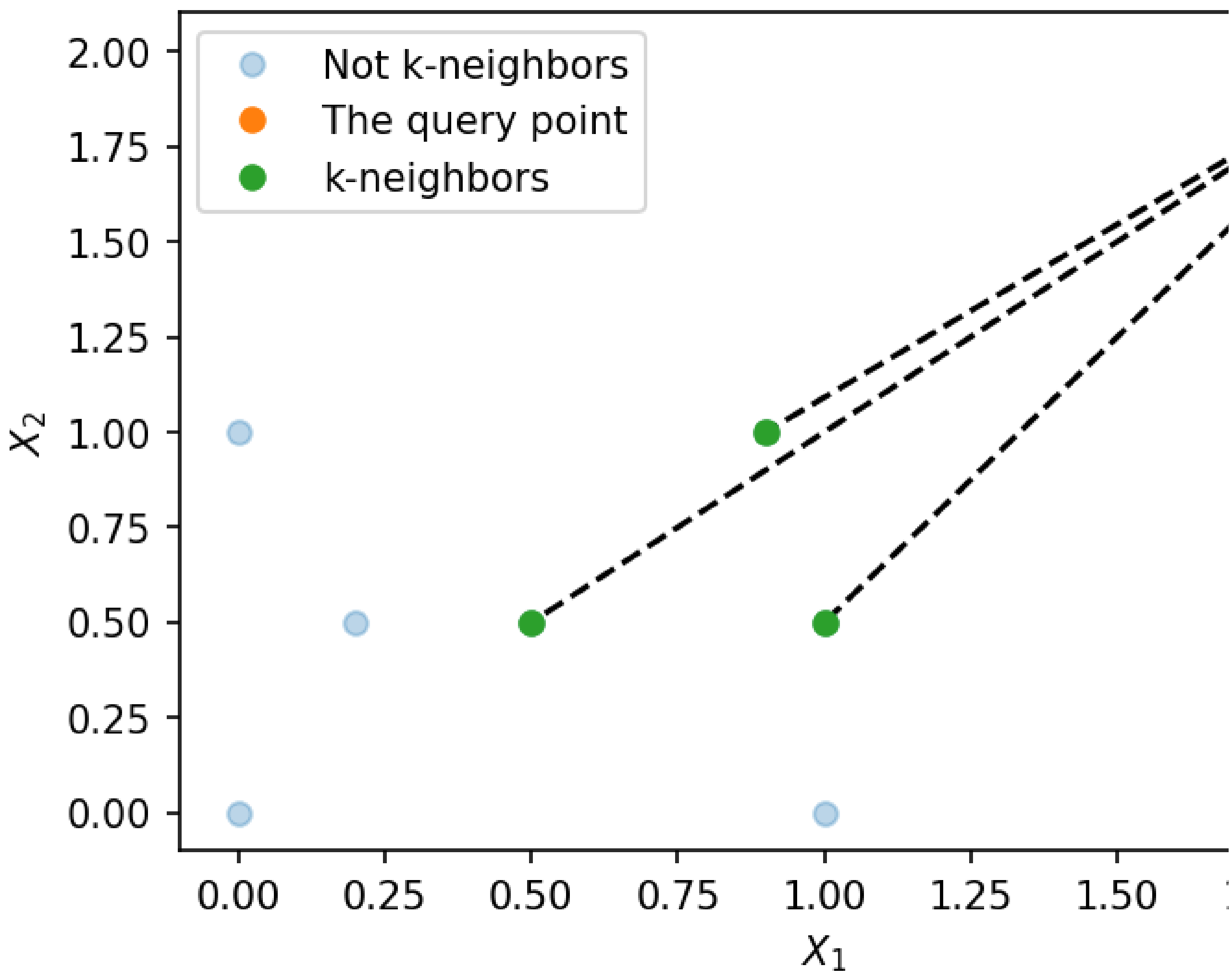


Let's create a function to visualize a query point and its neighbors more explicitly.

```
def plot_point_and_k_neighbors(X, highlight_index, n_neighbors=2):
    "Plots the points in X, and shows the n_neighbors of the highlight_index-th point"
    nn = neighbors.NearestNeighbors(n_neighbors=n_neighbors).fit(X)
    dist, index = nn.kneighbors()

    src_pt = X[highlight_index, :]

    plt.figure(dpi=150)
    # draw lines first, so points go on top
    for dest_index in index[highlight_index]:
        dest_pt = X[dest_index, :]
        plt.plot(*list(zip(src_pt, dest_pt)), 'k--')
    plt.plot(X[:, 0], X[:, 1], 'o', label='Not k-neighbors', alpha=0.3)
    plt.plot(*src_pt, 'o', label='The query point')
    plt.plot(X[index[highlight_index], 0], X[index[highlight_index], 1], 'o', label='k-nei
    plt.xlabel('$X_1$')
    plt.ylabel('$X_2$')
    plt.legend()
```

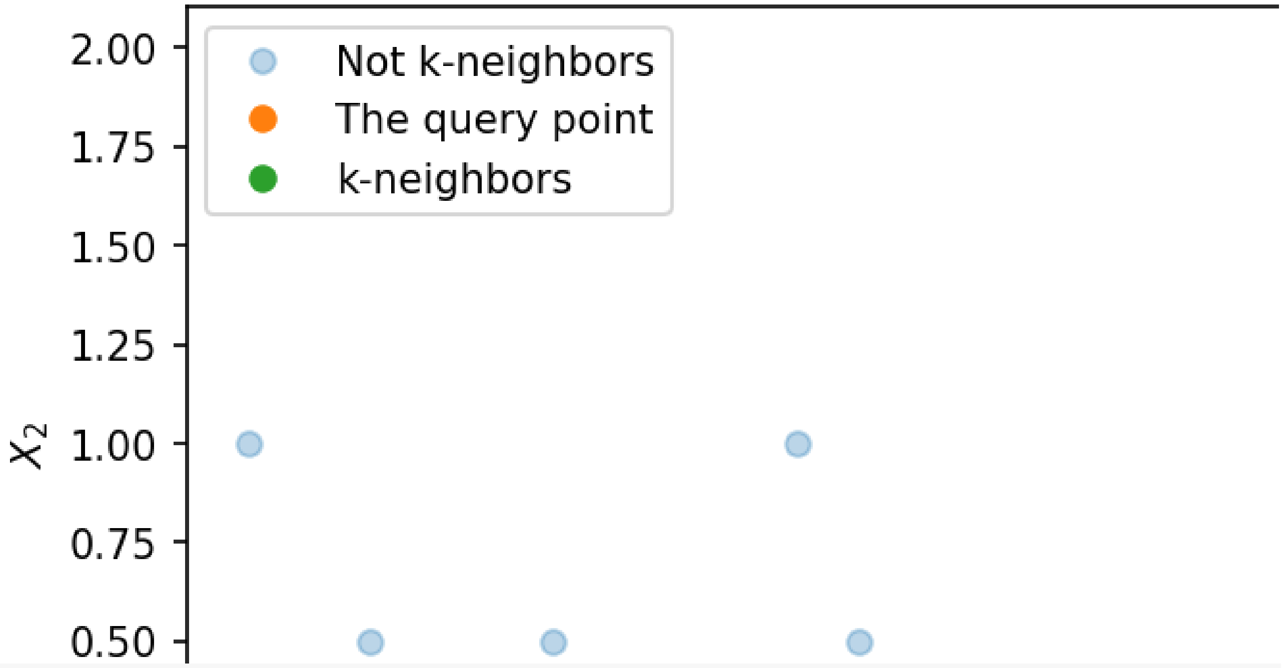
Here we can see the 4 points close to the query point (index of 0 in our data) are relatively close. Let's look at the anomalous point in the upper right (this point has an index of 7):

```
plot_point_and_k_neighbors(X, 7, n_neighbors=3)
```



We can see in this case all of the 4-nearest neighbors to our point are far away, so it seems reasonably unambiguous to call this point an anomaly. Let's add one more point next to the anomaly, and see how choosing "k" and the scoring function affects things.

```
# Toy dataset with two adjacent anomalies
X2 = np.concatenate([X, [[1.9, 2.0]]])

# Look at nearest neighbor (k=1)
plot_point_and_k_neighbors(X2, 7, n_neighbors=1)
```




```
def print_ranked_scores(obs, scores):
    scores_and_obs = sorted(zip(scores, obs), key=lambda t: t[0], reverse=True)
    print('Rank  Point\t\tScore')
    print('------------------------------')
    for index, score_ob in enumerate(scores_and_obs):
        score, point = score_ob
        print(f'{index+1:3d}.  {point}\t\t{score:6.4f}')

# Look at the outliers using 3 neighbors
```

```
# LOF for KNN 3

import sklearn.neighbors as neighbors
lof = neighbors.LocalOutlierFactor(n_neighbors=3, contamination='auto')
lof.fit(X2)
sk_lof = -lof.negative_outlier_factor_
print_ranked_scores(X2, sk_lof)
```

```
Rank  Point             Score
------------------------------
  1.  [2. 2.]           1.8608
  2.  [1.9 2. ]         1.8293
  3.  [0.2 0.5]         1.2154
  4.  [1.  0.5]         1.2091
  5.  [0. 0.]           1.0333
  6.  [0. 1.]           1.0120
  7.  [1. 0.]           0.9554
  8.  [0.9 1. ]         0.8891
  9.  [0.5 0.5]         0.8272
```

```
#LOF for KNN 1

def do_nn_outlier_scores(obs, n_neighbors=1):
    """
    Gives the score of a point as the distance from point to its k-th nearest neighbor.
```

```
    Larger score means more likely to be an outlier
    """
    nn = neighbors.NearestNeighbors(n_neighbors=n_neighbors)
    nn.fit(obs)
    dists, idx = nn.kneighbors()
    print(dists)
    scores = dists[:,-1]
    return scores

# Test
print_ranked_scores(X2, do_nn_outlier_scores(X2, 1))
```

```
[[0.5099]
 [0.5385]
 [0.5   ]
 [0.5385]
 [0.3   ]
 [0.3   ]
 [0.5   ]
 [0.1   ]
 [0.1   ]]
Rank  Point             Score
------------------------------
  1.  [0. 1.]           0.5385
  2.  [0. 0.]           0.5385
  3.  [0.9 1. ]         0.5099
  4.  [1. 0.]           0.5000
  5.  [1.  0.5]         0.5000
  6.  [0.5 0.5]         0.3000
  7.  [0.2 0.5]         0.3000
  8.  [2. 2.]           0.1000
  9.  [1.9 2. ]         0.1000
```

A. Create a function `do_nn_avg_scores(obs, n_neighbors=1)` that computes outlier scores using arithmetic mean distance from the point to each of the `n_neighbors` nearest neighbors as the score.

B. Do the same thing as in part (A) to create `do_nn_harm_scores(obs, n_neighbors=1)`, where you use the harmonic mean instead of the mean. The harmonic mean of n points is defined as

$$\text{harmonic}(X_1, X_2, \ldots, X_n) = \frac{n}{(1/x_1) + (1/x_2) + \ldots + (1/x_n)} = \frac{\left(\prod X_i\right)^{1/n}}{\bar{X}} = \frac{(X_1 X_2 \cdot X_n)^{1/n}}{\bar{X}}$$

Note that `scipy.stats` contains a `hmean` function you can use.

Q4)

```
def do_nn_avg_scores(obs, n_neighbors=1):

    nn = neighbors.NearestNeighbors(n_neighbors=n_neighbors)
    nn.fit(obs)
    dists, idx = nn.kneighbors()
```

```
    return np.mean(dists , axis=1)

# Test
print_ranked_scores(X2,do_nn_avg_scores(X2, 3))
```

```
Rank  Point            Score
------------------------------
  1.  [2. 2.]          1.1298
  2.  [1.9 2. ]        1.0878
  3.  [0. 0.]          0.7485
  4.  [1. 0.]          0.7168
  5.  [0. 1.]          0.7152
  6.  [0.9 1. ]        0.6701
  7.  [1.  0.5]        0.5033
  8.  [0.5 0.5]        0.4801
  9.  [0.2 0.5]        0.4590
```

```
from scipy.stats import hmean
def do_nn_harm_scores(obs, n_neighbors=1):

    nn = neighbors.NearestNeighbors(n_neighbors=n_neighbors)
    nn.fit(obs)
    dists, idx = nn.kneighbors()
    return hmean(dists , axis=1)

# Test
print_ranked_scores(X2, do_nn_harm_scores(X2, 3))
```

```
Rank  Point            Score
------------------------------
  1.  [0. 0.]          0.7024
  2.  [0. 1.]          0.6846
  3.  [1. 0.]          0.6705
  4.  [0.9 1. ]        0.6403
  5.  [1.  0.5]        0.5033
  6.  [0.5 0.5]        0.4351
  7.  [0.2 0.5]        0.4257
  8.  [2. 2.]          0.2672
  9.  [1.9 2. ]        0.2660
```

Q5) This exercise refers to density-based methods (Local Outlier Factor; Section 3)

Create a function do_lof_outlier_scores(obs, n_neighbors=3) that computes outlier scores using the LOF method. Recall that the values returned by sklearn's implementation are negatives of what we want.

```
def do_lof_outlier_scores(obs, n_neighbors=3):
    neigh = neighbors.NearestNeighbors(n_neighbors=3).fit(obs)

    #Return indices of and distances to the neighbors of each point
    topk_dist, my_kneigh = neigh.kneighbors()
```

```
    # Create list of distances of furthest (kth) neighbor
    k_dist = topk_dist[:,-1]

    # Reachability distance: maximum of true distance between query neighbor and query poi
    # and distance to kth nearest neighbor of query neighbor
    reach = np.maximum(topk_dist, k_dist[my_kneigh])

    # Local reacability density is reciprocal of average reachability distance
    lrd = 1.0 / np.mean(reach, axis=1)

    # Local outlier factor is given by
    # average local density of neighbors / local density of query point
    lrd_ratios = lrd[my_kneigh] / lrd[:, np.newaxis]
    lof = np.mean(lrd_ratios, axis=1)

    return lof
```

✓ 0s completed at 8:06 PM