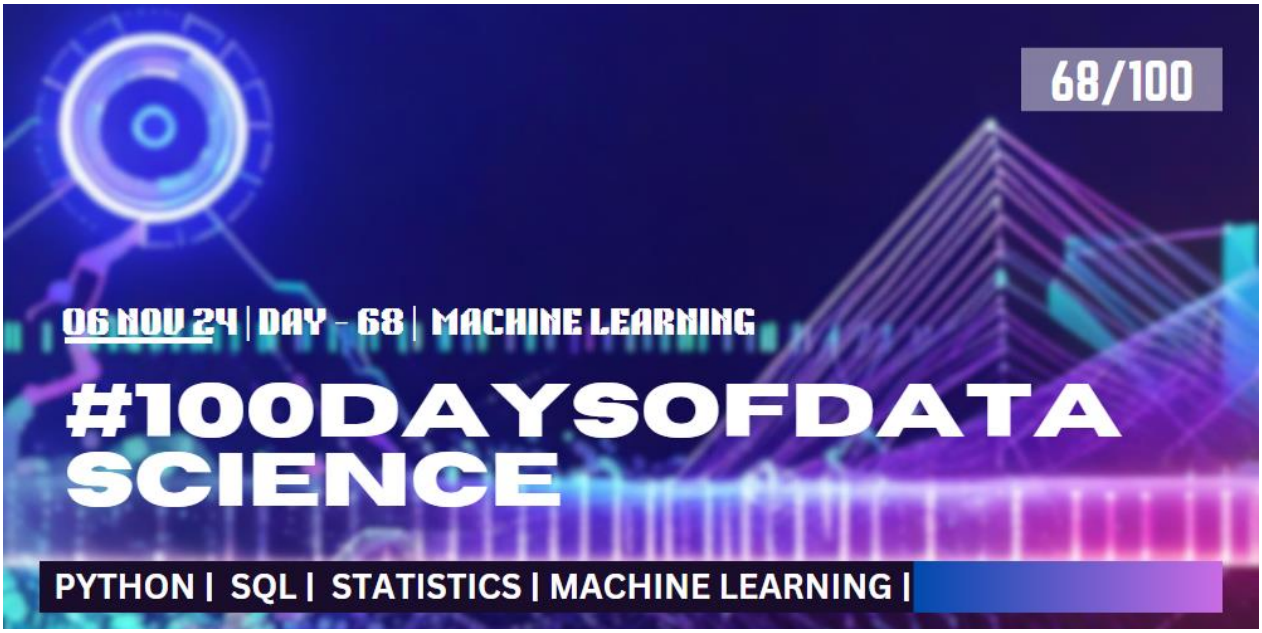



# Outlier Detection

**What is an Outlier?**

An outlier is a data point that lies an abnormal distance from other values in a dataset. It can occur in any type of data and may represent rare events, measurement errors, or genuine observations that are fundamentally different from the rest. Outlier analysis is a critical part of data preprocessing in machine learning, aimed at identifying unusual data points that deviate significantly from most of the data. Outliers can heavily impact model accuracy, as they can skew results and make models less reliable. Properly handling outliers enhances model robustness and ensures accurate insights.

Let's take an example to check what happens to a data set with and data set without outliers.

| | **Data without outliers** | **Data with outliers** |
|---|---|---|
| **Data** | 1,2,3,3,4,5,4 | 1,2,3,3,4,5,400 |
| **Mean** | 3.142 | 59.714 |
| **Median** | 3 | 3 |
| **Standard Deviation** | 1.345185 | 150.057 |

As you can see, data set with outliers has significantly different mean and standard deviation. In the first scenario, we will say that average is 3.14. But with the outlier, average soars to 59.71. This would change the estimate completely.

**Causes of Outliers**

1. **Measurement Errors**: Mistakes in data collection, input errors, or technical issues can introduce outliers.
2. **Data Entry Errors**: Incorrect entries, such as typos or improper formats, often produce extreme values.
3. **Natural Variation**: Some outliers are genuine and represent rare occurrences or extreme cases.
4. **Sampling Errors**: If the sample does not represent the population well, it may include extreme cases that are outliers.

**Importance of Outlier Analysis**

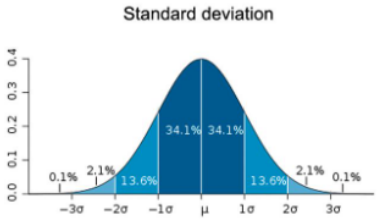
1. **Improving Model Accuracy**: Detecting and handling outliers helps reduce bias in models, improving predictions.
2. **Data Quality Assurance**: Identifying outliers helps ensure that data is clean, valid, and representative of the intended sample.
3. **Risk Identification**: In domains like finance, outliers can represent significant events, such as fraud or abnormal activity.

4. **Enhanced Model Interpretability**: Managing outliers provides a clearer understanding of patterns and relationships within the data.

## Techniques for Outlier Detection

1. **Statistical Methods**
   - **Z-Score**: Calculates how far a point deviates from the mean in terms of standard deviations. Points with Z-scores greater than a threshold (e.g., 3) are flagged as outliers.

Using Z score method,we can find out how many standard deviations value away from the mean.




% of values expected to lie in the symmetric interval (−zσ, zσ)

| | |
|---|---|
| 1σ | 68.2689492% |
| 2σ | 95.4499736% |
| 3σ | 99.7300204% |
| 4σ | 99.993666% |
| 5σ | 99.9999426697% |

---

Figure in the left shows area under normal curve and how much area that standard deviation covers.

* 68% of the data points lie between + or - 1 standard deviation.
* 95% of the data points lie between + or - 2 standard deviation
* 99.7% of the data points lie between + or - 3 standard deviation

---

Z-score formula

$$Z\,score = \frac{X - Mean}{Standard\,Deviation}$$

---

If the z score of a data point is more than 3 (because it cover 99.7% of area), it indicates that the data value is quite different from the other values. It is taken as outliers.

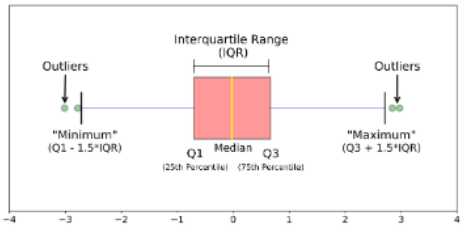
### Z-Score

[19]:
```python
import numpy as np

# Sample data
x = np.array([12, 13, 14, 19, 21, 23])
y = np.array([12, 13, 14, 19, 21, 23, 45000])

# Z-Score threshold for detecting outliers
threshold = 2

def z_score_outliers(data):
    mean_data = np.mean(data)
    std_dev = np.std(data)
    z_scores = np.abs((data - mean_data) / std_dev)  # Calculate Z-Scores
    print("Z-Scores:", z_scores)

    # Find outliers based on the threshold
    outliers = data[z_scores > threshold]
    if len(outliers) > 0:
        print(f"Outliers detected: {outliers}")
    else:
        print("No outliers detected")
    return outliers

# Run Z-Score outlier detection on the sample data
print("Checking for outliers in dataset x:")
z_score_outliers(x)
print("\nChecking for outliers in dataset y:")
z_score_outliers(y)
```

```
Checking for outliers in dataset x:
Z-Scores: [1.18957738 0.9516619  0.71374643 0.47583095 0.9516619  1.42749285]
No outliers detected

Checking for outliers in dataset y:
Z-Scores: [0.40856592 0.4085024  0.40843887 0.40812122 0.40799416 0.4078671
 2.44948967]
Outliers detected: [45000]
```

[19]:
```
array([45000])
```

- **IQR (Interquartile Range)**: Defines an acceptable range around the median, with points lying outside 1.5 times the IQR considered outliers.




* Q1 represents the 1st quartile/25th percentile of the data.
* Q2 represents the 2nd quartile/median/50th percentile of the data.
* Q3 represents the 3rd quartile/75th percentile of the data.
* (Q1–1.5*IQR) represent the smallest value in the data set and (Q3+1.5*IQR) represnt the largest value in the data set.

## IQR

```
•[22]: import numpy as np

       # Sample data
       x = np.array([12, 13, 14, 19, 21, 23])
       y = np.array([12, 13, 14, 19, 21, 23, 100, 45000])

       def iqr_outliers(data):
           # Calculate Q1, Q3, and IQR
           q1 = np.percentile(data, 25)
           q3 = np.percentile(data, 75)
           iqr = q3 - q1

           # Define outlier bounds
           lower_bound = q1 - 1.5 * iqr
           upper_bound = q3 + 1.5 * iqr

           # Identify outliers
           outliers = data[(data < lower_bound) | (data > upper_bound)]
           print("IQR Outliers:", outliers if len(outliers) > 0 else "No outliers detected")
           return outliers

       # Run IQR outlier detection on the sample data
       print("Checking for outliers in dataset x:")
       iqr_outliers(x)
       print("\nChecking for outliers in dataset y:")
       iqr_outliers(y)
```

```
Checking for outliers in dataset x:
IQR Outliers: No outliers detected

Checking for outliers in dataset y:
IQR Outliers: [  100 45000]
[22]: array([  100, 45000])
```

- **Hypothesis Testing (Grubbs' Test)**: This statistical test is specifically designed to identify outliers in a dataset. Grubbs' test evaluates whether the highest or lowest value in the dataset is an outlier by calculating a test statistic and comparing it to a critical value. This test assumes normal distribution and is particularly useful for univariate datasets.

Grubbs' test is defined for the hypothesis:

Ho: There are no outliers in the data set

$H_1$ : There is exactly one outlier in the data set

The Grubbs' test statistic is defined as:

$$G_{calculated} = \frac{\max|X_i - \overline{X}|}{SD}$$

with $\overline{X}$ and $SD$ denoting the sample mean and standard deviation, respectively.

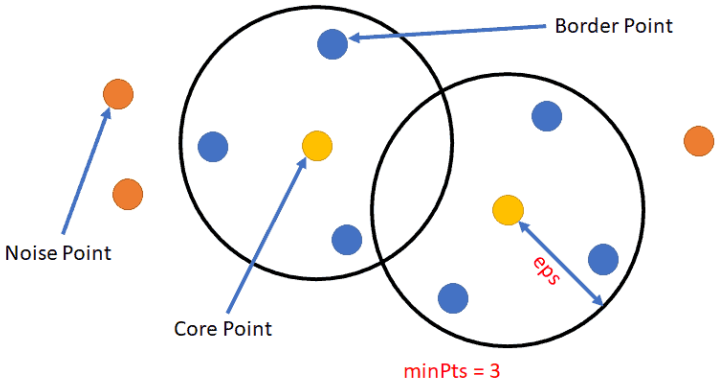
$$G_{critical} = \frac{(N-1)}{\sqrt{N}} \sqrt{\frac{\left(t_{\alpha/(2N),N-2}\right)^2}{N-2+\left(t_{\alpha/(2N),N-2}\right)^2}}$$

## Hypothesis Testing(grubbs test)

```
[3]: import numpy as np
import scipy.stats as stats
x = np.array([12,13,14,19,21,23])
y = np.array([12,13,14,19,21,23,45])
def grubbs_test(x):
    n = len(x)
    mean_x = np.mean(x)
    sd_x = np.std(x)
    numerator = max(abs(x-mean_x))
    g_calculated = numerator/sd_x
    print("Grubbs Calculated Value:",g_calculated)
    t_value = stats.t.ppf(1 - 0.05 / (2 * n), n - 2)
    g_critical = ((n - 1) * np.sqrt(np.square(t_value))) / (np.sqrt(n) * np.sqrt(n - 2 + np.square(t_value)))
    print("Grubbs Critical Value:",g_critical)
    if g_critical > g_calculated:
        print("From grubbs_test we observe that calculated value is lesser than critical value, Accept null hypothesis and conclude that there is no outlie
    else:
        print("From grubbs_test we observe that calculated value is greater than critical value, Reject null hypothesis and conclude that there is an outli
grubbs_test(x)
grubbs_test(y)
```

```
Grubbs Calculated Value: 1.4274928542926593
Grubbs Critical Value: 1.8871451177787137
From grubbs_test we observe that calculated value is lesser than critical value, Accept null hypothesis and conclude that there is no outliers

Grubbs Calculated Value: 2.2765147221587774
Grubbs Critical Value: 2.019968507680656
From grubbs_test we observe that calculated value is greater than critical value, Reject null hypothesis and conclude that there is an outliers
```

2. **Density-Based Methods**
   - **DBSCAN (Density-Based Spatial Clustering)**: Identifies clusters of data points based on density, with isolated points treated as outliers.




## DBSCAN

```
[25]: import numpy as np
from sklearn.cluster import DBSCAN

# Sample data
x = np.array([[12], [13], [14], [19], [21], [23]])  # Reshaping data for DBSCAN
y = np.array([[12], [13], [14], [19], [21], [23], [100], [45000]])

def dbscan_outliers(data, eps=3, min_samples=2):
    # Initialize DBSCAN model
    db = DBSCAN(eps=eps, min_samples=min_samples)

    # Fit the model and get labels (-1 represents an outlier)
    labels = db.fit_predict(data)

    # Identify outliers
    outliers = data[labels == -1]
    print("DBSCAN Outliers:", outliers.flatten() if len(outliers) > 0 else "No outliers detected")
    return outliers

# Run DBSCAN outlier detection on the sample data
print("Checking for outliers in dataset x:")
dbscan_outliers(x)
print("\nChecking for outliers in dataset y:")
dbscan_outliers(y)
```

```
Checking for outliers in dataset x:
DBSCAN Outliers: No outliers detected

Checking for outliers in dataset y:
DBSCAN Outliers: [  100 45000]
```

```
[25]: array([[  100],
       [45000]])
```

- **Local Outlier Factor (LOF)**: Measures the local density deviation of a point compared to its neighbors, flagging points in lower-density regions as outliers.

For a given Data set

$$D_n = \{(x_i, y_i) | x_i \in R^2, y_i \in \{X, Y, Z\}\}$$

Local Outlier Factor for each data point is given by

$$LOF(x_i) = \frac{\sum_{x_j \in N(x_i)} lrd(x_j)}{|N(x_i)|} \times \frac{1}{lrd(x_i)}$$

$|N(x_i)|$ : Number of elements in the neighborhood of $x_i$

$lrd(x_i)$ : Local Reachability Density of $x_i$

## Local Outlier Factor

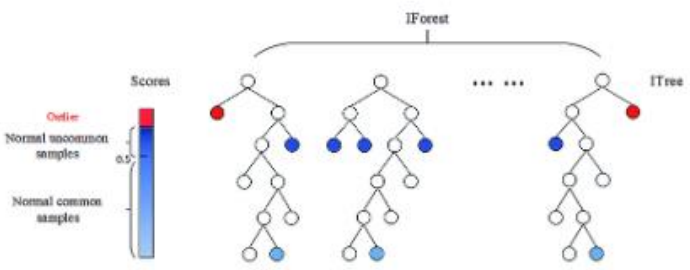
[28]:
```
import numpy as np
from sklearn.neighbors import LocalOutlierFactor

# Sample data
x = np.array([[12], [13], [14], [19], [21], [23]])  # Reshaping data for LOF
y = np.array([[12], [13], [14], [19], [21], [23], [100], [45000]])

def lof_outliers(data, n_neighbors=2):
    # Initialize LOF model
    lof = LocalOutlierFactor(n_neighbors=n_neighbors, contamination="auto")

    # Fit the model and predict outliers (-1 indicates an outlier)
    labels = lof.fit_predict(data)

    # Identify outliers
    outliers = data[labels == -1]
    print("LOF Outliers:", outliers.flatten() if len(outliers) > 0 else "No outliers detected")
    return outliers

# Run LOF outlier detection on the sample data
print("Checking for outliers in dataset x:")
lof_outliers(x)
print("\nChecking for outliers in dataset y:")
lof_outliers(y)
```

```
Checking for outliers in dataset x:
LOF Outliers: No outliers detected

Checking for outliers in dataset y:
LOF Outliers: [  100 45000]
```

[28]:
```
array([[  100],
       [45000]])
```

3. **Machine Learning Models**
   - **Isolation Forest**: An ensemble method that isolates data points by randomly selecting features and splitting them. Outliers are isolated quickly due to their rarity.

It is an clustering algorithm that belongs to the ensemble decision trees family and is similar in principle to Random Forest.



## Isolation Forest

[32]:
```
import numpy as np
from sklearn.ensemble import IsolationForest

# Sample data
x = np.array([[12], [13], [14], [19], [21], [23]])  # Reshaping data for Isolation Forest
y = np.array([[12], [13], [14], [19], [21], [23], [100], [45000]])

def isolation_forest_outliers(data, contamination=0.1):
    # Initialize Isolation Forest model
    iso_forest = IsolationForest(contamination=contamination, random_state=42)

    # Fit the model and predict outliers (-1 represents an outlier)
    labels = iso_forest.fit_predict(data)

    # Identify outliers
    outliers = data[labels == -1]
    print("Isolation Forest Outliers:", outliers.flatten() if len(outliers) > 0 else "No outliers detected")
    return outliers

# Run Isolation Forest outlier detection on the sample data
print("Checking for outliers in dataset x:")
isolation_forest_outliers(x)
print("\nChecking for outliers in dataset y:")
isolation_forest_outliers(y)
```

```
Checking for outliers in dataset x:
Isolation Forest Outliers: [23]

Checking for outliers in dataset y:
Isolation Forest Outliers: [45000]
```

[32]:
```
array([[45000]])
```

4. **Clustering Methods**
   - **Gaussian Mixture Models (GMM)**: Uses probabilities to assess the likelihood of a point belonging to a certain distribution, with low-probability points flagged as outliers.

## Gaussian Mixture Models (GMM)

•[39]:
```
import numpy as np
from sklearn.mixture import GaussianMixture
import warnings

# Sample data
x = np.array([[12], [13], [14], [19], [21], [23]])  # Reshaping data for GMM
y = np.array([[12], [13], [14], [19], [21], [23], [100]])

# Suppress specific warnings
with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=UserWarning)

    def gmm_outliers(data, n_components=2, threshold=0.01):
        # Fit GMM to the data
        gmm = GaussianMixture(n_components=n_components)
        gmm.fit(data)

        # Calculate log-likelihood of each point under the model
        scores = gmm.score_samples(data)

        # Detect outliers based on the threshold
        outliers = data[scores < np.log(threshold)]
        print("GMM Outliers:", outliers.flatten() if len(outliers) > 0 else "No outliers detected")
        return outliers

    # Run GMM outlier detection on the sample data
    print("Checking for outliers in dataset x:")
    gmm_outliers(x)
    print("\nChecking for outliers in dataset y:")
    gmm_outliers(y)
```

```
Checking for outliers in dataset x:
GMM Outliers: No outliers detected

Checking for outliers in dataset y:
GMM Outliers: [100]
```

**Managing Outliers**

1. **Removal**: Simply remove outliers if they are due to errors or are not representative of the population.
2. **Transformation**: Apply transformations (e.g., log, square root) to reduce the effect of extreme values.
3. **Imputation**: Replace outliers with more typical values (e.g., median or mean of the feature).
4. **Capping or Flooring**: Limit outliers to a specified range, often based on IQR or standard deviation bounds.

**Choosing the Right Outlier Detection Technique**

- **Z-Score & IQR**: Suitable for univariate, normally distributed data with fewer features.
- **Density-Based**: Ideal for complex datasets with clusters and varying densities.
- **Isolation Forest**: Well-suited for high-dimensional data or cases with clear definitions of normality.

Outlier analysis is essential for refining datasets, ensuring accurate results, and improving model performance. By understanding and selecting the appropriate detection technique, you can manage outliers effectively and build robust machine learning models.