**Detecting Outliers with Simple and Advanced Techniques**

**A tutorial on how to detect outliers using standard deviation, interquartile range, isolation forest, DBSCAN, and local outlier factor**



Photo by [Benjamin Voros](https://unsplash.com/@vorosbenisop?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral)

[Outliers](https://en.wikipedia.org/wiki/Outlier) are data points that are far away from the majority of the observations in the dataset. Outliers can appear for many reasons such as natural deviations in population behavior, fraudulent activities, and human or system errors. However, detecting and identifying outliers is essential before running any statistical analysis or preparing data for training machine learning models.

In this article, we will cover univariate and multivariate outliers, how they differ and how they can be identified using statistical methods and automated anomaly detection techniques. We will see the interquartile range and standard deviation methods to detect univariate outliers and isolation forest, DBSCAN — Density-Based Spatial Clustering of Applications with Noise, and LOF — Local Outlier Factor to detect multivariate outliers.

While following the article, I encourage you to check out the [Jupyter Notebook](https://github.com/Idilismiguzel/Machine-Learning/blob/master/Outlier%20Detection/Outlier_Detection.ipynb) on my GitHub for full analysis and code.

We have a lot to cover, let’s get started! 🚀

**Data**

In the article, we will use the [Glass Identification data set from UCI](https://www.kaggle.com/uciml/glass), which has 8 attributes (i.e. ‘Na’) related to glass content and also glass type.

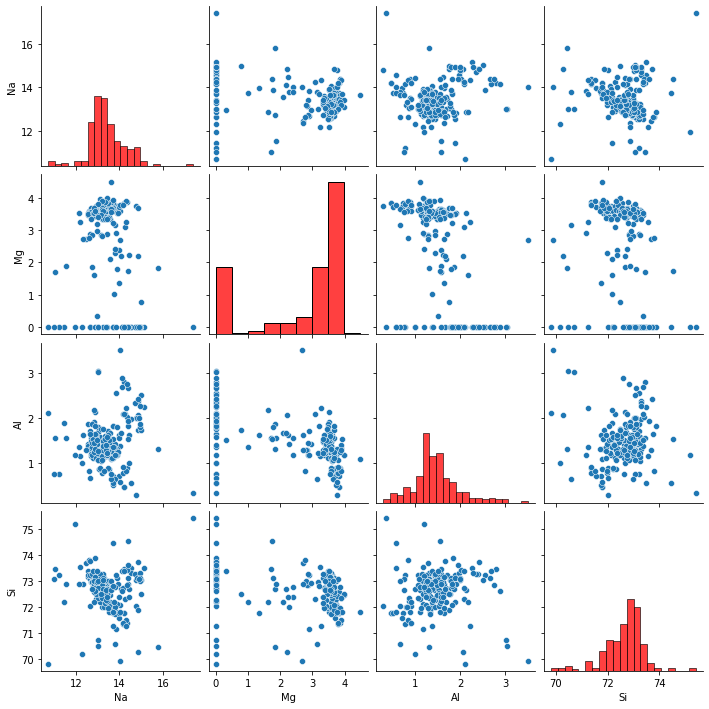
import pandas as pdglass = pd.read\_csv('glass.csv')

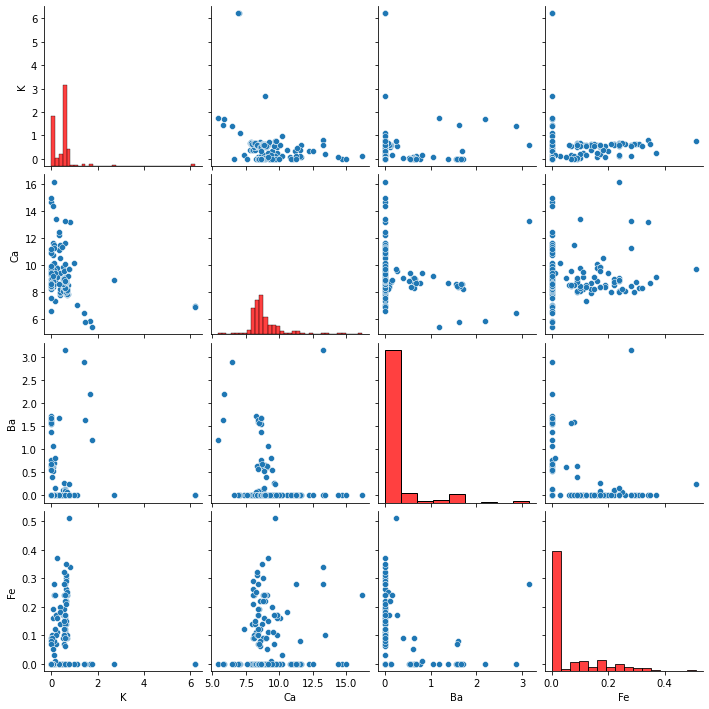


**Univariate and Multivariate Outliers**

By using seaborn’s pairplot we can plot the pairwise relationships between glass contents and through this visual, we can see what the distribution of our underlying data looks like.

import seaborn as snssns.pairplot(glass, diag\_kws={'color':'red'})

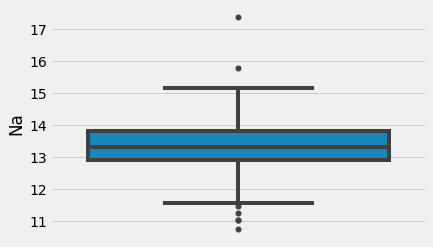




Pairwise relationship between variables

If we take a closer look at the plots above we see that glass attributes are in the x and y-axis. Along with the diagonal in red, we can see the histogram plots showing the distributions.

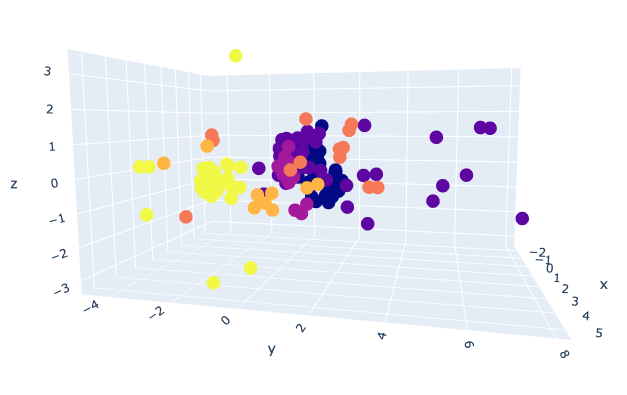
As you can see not all attributes have a normal distribution that follows a bell-shaped curve, but in fact, most of the attributes are skewed towards lower values (i.e. Ba, Fe) or higher values (i.e. Mg). To detect **univariate outliers** we should focus on the distribution of a single attribute and find data points that fall far away from the majority of the data of that attribute. For example, if we select “Na” and plot a boxplot we can find which are the data points outside of whiskers and can be marked as outliers.



Boxplot of Na — showing data points that are outside of whiskers.

In contrast, to detect **multivariate outliers** we should focus on the combination of at least two variables in n-dimensional space. For example, in the glass dataset, we can use all eight attributes of glass and plot them in n-dimensional space and find multivariate outliers by detecting which are the data points that fall far away.

Since plotting more than three dimensions is not possible, we should find a way to transform eight dimensions into a lower-dimensional space. PCA — Principal Component Analysis is a dimensionality reduction technique that performs linear mapping of the high-dimensional space into lower-dimensional space by maximizing the variance in the low-dimensional representation. We can transform eight dimensions into three-dimensional space by performing PCA with n\_components=3.



PCA visualization with 3 components (Colors represent “Type” of glass)

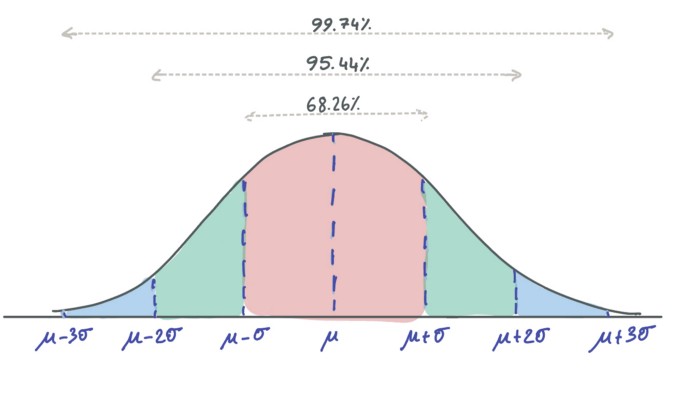
You can see in the plot, there are some data points close to each other (composing dense areas) and some far away that might be *multivariate* outliers.

We need to follow different procedures to detect univariate and multivariate outliers. Let’s start with univariate outliers and learn standard deviation and interquartile range methods to detect them.

**Univariate Outlier Detection**

**1. Standard Deviation Method**

Suppose a variable is (almost) normally distributed. In this case, its histogram should follow a bell-shaped curve, and 68.3% of the data values lie within one standard deviation from the mean, 95.4% of the data values lie within two standard deviations from the mean, and 99.7% of the data values lie within three standard deviations from the mean.



Therefore, we can detect outliers if there are data points that lie further away than three standard deviations from the mean.

By using the standard deviation technique we removed two records based on the distribution of the “Na” variable which is extreme. You can expand and cover all other attributes to remove univariate outliers.

Shape of original dataset: (213, 9)   
Shape of dataset after removing outliers in Na column: (211, 9)

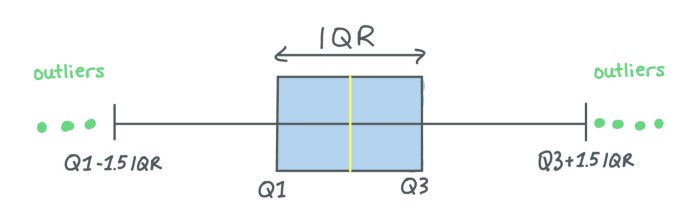
**2. Inter-Quartile Range Method**

The interquartile range method, best shown using a box plot, divides data into quartiles by defining three points:

Quartile 1 (Q1) represents 25th percentile  
Quartile 2 (Q2) represents 50th percentile  
Quartile 3 (Q3) represents 75th percentile

The box in the boxplot represents the IQR range which is defined as the range between Q1 and Q3; IQR = Q3 — Q1 and data points that fall below Q1 - 1.5\*IQR or above Q3 + 1.5\*IQR are defined as outliers.

In the boxplot Q1 - 1.5\*IQR and Q3 + 1.5\*IQRare represented by the whiskers and outliers are represented with dots above or below.



Shape of original dataset: (213, 9)   
Shape of dataset after removing outliers in Na column: (206, 9)

By using the IQR technique we removed seven records based on the distribution of the “Na” variable. As you can notice standard deviation method was only able to find 2 outliers which were really extreme points, but with the IQR method we were able to detect more (5 more records that were not so extreme). It is up to you and your use case to decide which method is better for the dataset and if you have room to drop more data points.

Let’s continue with multivariate outliers and learn about isolation forest, DBSCAN — Density-Based Spatial Clustering of Applications with Noise, and LOF — Local Outlier Factor.

**Multivariate Outlier Detection**

**1. Isolation Forest Method**

[Isolation forest](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html) is an unsupervised machine learning algorithm based on random forest. You might know, random forest is an [ensemble learning](https://medium.com/towards-data-science/practical-guide-to-ensemble-learning-d34c74e022a0) model that uses an ensemble of the base models (say 100 decision trees) and models that are outperforming given higher weight in the final decision.

If you need a refresher on Ensemble Learning you can have a look at this article.

**[Practical Guide to Ensemble Learning](https://towardsdatascience.com/practical-guide-to-ensemble-learning-d34c74e022a0" \t "_blank)**

**[Improve your model with voting, bagging, boosting and stacking](https://towardsdatascience.com/practical-guide-to-ensemble-learning-d34c74e022a0" \t "_blank)**

[towardsdatascience.com](https://towardsdatascience.com/practical-guide-to-ensemble-learning-d34c74e022a0" \t "_blank)

Isolation forest follows the methodology of random forest, but in contrast, it detects (or in other words isolates) anomalous data points. To do that, it makes two fundamental assumptions: outliers are a minority and they are unusual.

Isolation forest creates decision trees by randomly selecting a feature and then randomly selecting a split rule to separate the values of selected feature. This process continues until set hyperparameter values are reached. Then, isolation forest considers if the tree is shorter and has fewer partitions then the corresponding value is an outlier (minority and unusual).

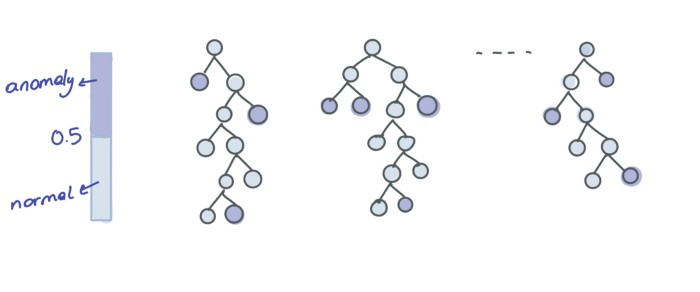


Illustration on isolation forest anomaly detection

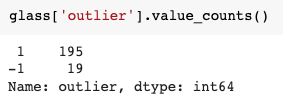
Let’s see the implementation using IsolationForest class from [scikit-learn](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html).

from sklearn.ensemble import IsolationForestIsolationForest(*n\_estimators=100*, *max\_samples='auto'*, *contamination='auto'*, *max\_features=1.0*, *bootstrap=False*, *n\_jobs=None*, *random\_state=None*, *verbose=0*, *warm\_start=False*)

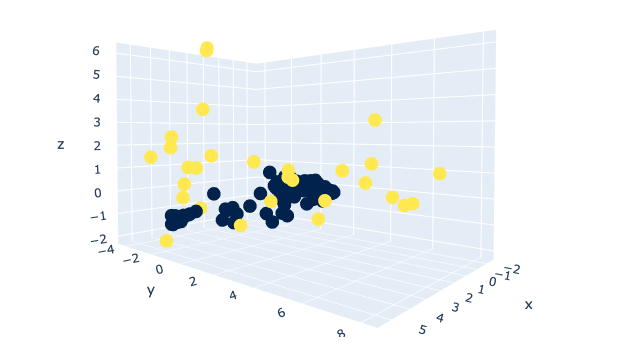
Isolation Forest algorithm has several hyperparameters such as **n\_estimators** for the number of base estimators to ensemble, **max\_samples** for the number of samples for training the model, **contamination** for defining the proportion of outliers in data, **max\_features** for the number of features drawn from data for training. You can look at the [doc](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html) for the rest.

We initiate isolation forest by setting the number of base models to 100, max features to the total number of features, and contamination to 'auto' that uses offset and contamination thresholds as determined in its original paper. If contamination is 0.1 then 10% of the dataset will be defined as outliers.

By calling glass['outlier'].value\_counts() we can see that there are 19 records tagged as -1 — outliers and the rest 195 records are tagged as 1 –not outliers.



We can visualize outliers by reducing the number of features to three components with PCA as we covered previously.



Outliers (in yellow) detected by Isolation Forest (n\_estimators=100, contamination=’auto’)

To keep the content focused, I will not show how to tune hyperparameters, but if you are interested you can check [this article](https://towardsdatascience.com/hyperparameter-tuning-with-grid-search-and-random-search-6e1b5e175144).

**2. Density-Based Spatial Clustering of Applications with Noise (DBSCAN) Method**

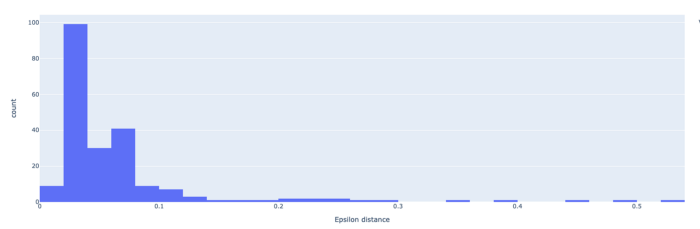
[DBSCAN](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html?highlight=dbscan#sklearn.cluster.DBSCAN) is a popular clustering algorithm that is often used as an alternative to K-means. It is *density-based*, which means it focuses on areas with high density where many data points locate. It performs *spatial clustering* by measuring feature space distance (i.e. Euclidean distance) between data to identify which can be clustered together. It allows applications with noise which means we can use DBSCAN on noisy data. But that’s not all, one of the biggest advantages of DBSCAN is we do not need to pre-define the number of clusters.

Let’s see the implementation using DBSCAN class from [scikit-learn](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html).

from sklearn.cluster import DBSCANDBSCAN(*eps=0.5*, *min\_samples=5*, *metric=’euclidean’*, *metric\_params=None*, *algorithm=’auto’*, *leaf\_size=30*, *p=None*, *n\_jobs=None*)

DBSCAN has several hyperparameters such as **eps (epsilon)** for the maximum distance between two data points to be considered in the same cluster, **min\_samples** for the number of close data points for a point to be considered a core point, **metric** for calculating the distance between points. You can look at the [doc](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html?highlight=dbscan#sklearn.cluster.DBSCAN) for the rest.

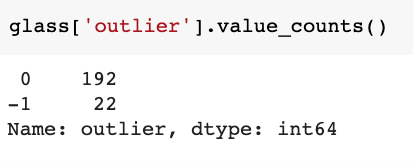
While initiating DBSCAN, it is very important to carefully select hyperparameters. For example, if the eps value is selected too small, then most of the data can be classified as outliers since the neighborhood area is defined as smaller. In contrast, if the eps value is selected too large, then most of the points can be clustered together since they are likely to be located in the same neighborhood. Here we selected eps as 0.4 using the [k-distance graph.](https://stackoverflow.com/questions/43160240/how-to-plot-a-k-distance-graph-in-python)



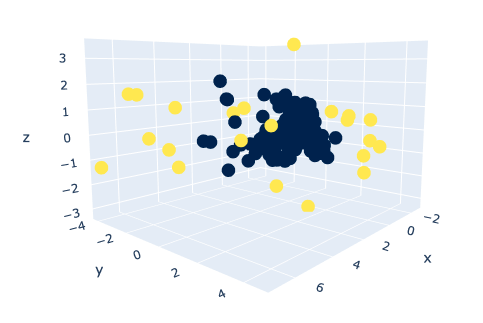
K-distance plot where most of the data lie within 0.4 units from their 10th nearest neighbor

Besides, min\_samples is an important hyperparameter that is usually equal to or greater than 3, and most cases selected as D+1 where D is the dimension of the dataset. In our example, we set min\_samples equal to 10.

Since DBSCAN identifies clusters by density, high-density areas are where clusters occur, and low-density areas are where outliers take place. By calling glass['outlier'].value\_counts() we can see that there are 22 records tagged as -1 — outliers and the rest 192 records are tagged as 1 –not outliers.



We can visualize outliers using PCA.



Outliers (in yellow) detected by DBSCAN (eps=0.4, min\_samples=10)

**3. Local Outlier Factor (LOF)**

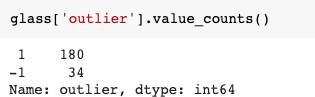
[LOF](https://scikit-learn.org/stable/auto_examples/neighbors/plot_lof_outlier_detection.html#sphx-glr-auto-examples-neighbors-plot-lof-outlier-detection-py) is a popular unsupervised anomaly detection algorithm that computes the local density deviation of data points with respect to their neighbors. After this computation, points that have lower densities are considered outliers.

Let’s see the implementation using LOF class from [scikit-learn](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html).

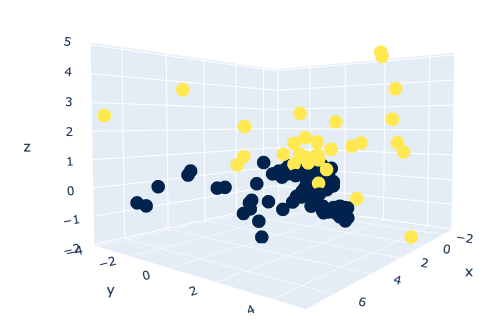
from sklearn.neighbors import LocalOutlierFactor*LocalOutlierFactor(n\_neighbors=20, algorithm='auto', leaf\_size=30, metric='minkowski', p=2, metric\_params=None, contamination='auto', novelty=False, n\_jobs=None)*

LOF has several hyperparameters such as **n\_neighbors** for selecting the number of neighbors by default equal to 20 and **contamination** for defining the proportion of outliers it can be equal to 'float'that is in the range (0, 0.5] or 'auto' that uses offset and contamination thresholds as determined in its original paper.

By calling glass['outlier'].value\_counts() we can see that there are 34 records tagged as -1 — outliers and the rest 180 records are tagged as 1 –not outliers.



Lastly, we can visualize these outliers using PCA.



Outliers (in yellow) detected by LOF (n\_neighbors=20, contamination=’auto’)

**Conclusion**

In this article, we explored different methods to detect outliers in our dataset. We started with univariate outlier detection techniques and covered standard deviation and interquartile range methods. We performed these methods on the “Na” column in the glass identification dataset. We then moved to multivariate outlier detection techniques and covered isolation forest, DBSCAN, and local outlier factor. With these methods, we learned how to detect outliers using all the dimensions in feature space. In addition to outlier detection, we also learned how to use PCA — a dimensionality reduction technique to visualize n-dimensional data.

While covering the methods, we didn’t spend too much time tuning the hyperparameters. However, hyperparameter tuning is an important step in ML model development. If you want to learn or refresh your knowledge, you can have a look at the article below.

**[Hyperparameter Tuning with Grid Search and Random Search](https://towardsdatascience.com/hyperparameter-tuning-with-grid-search-and-random-search-6e1b5e175144" \t "_blank)**

**[And a deep dive into how to combine them](https://towardsdatascience.com/hyperparameter-tuning-with-grid-search-and-random-search-6e1b5e175144" \t "_blank)**

[towardsdatascience.com](https://towardsdatascience.com/hyperparameter-tuning-with-grid-search-and-random-search-6e1b5e175144" \t "_blank)

I hope you enjoyed reading about outlier detection and find the article useful for your work!