Detecting And Treating Outliers In Python — Part 1

Hands-On Tutorial On Univariate Outliers

Alicia Nov 22, 2020 · 7 min read

Image by Will Myers on Unsplash

An Explorative Data Analysis (EDA) is crucial when working on data science projects. Knowing your data inside and out can simplify decision making concerning the

from the other data points in a random sample of a population.

But why can outliers cause problems?

Because in data science, we often want to make assumptions about a specific population. Extreme values, however, can have a significant impact on conclusions drawn from data or machine learning models. With outlier detection and treatment, anomalous observations are viewed as part of different populations to ensure stable findings for the population of interest.

When identified, outliers may reveal unexpected knowledge about a population, which also justifies their special handling during EDA.

Moreover, inaccuracies in data collection and processing can create so-called error-outliers. These measurements often do not belong to the population we are interested in and therefore need treatment.

Different types of outliers

One must distinguish between **univariate** and **multivariate outliers**. Univariate outliers are extreme values in the distribution of a specific variable, whereas multivariate outliers are a combination of values in an observation that is unlikely. For example, a univariate outlier could be a human age measurement of 120 years or a temperature measurement in Antarctica of 50 degrees Celsius.

A multivariate outlier could be an observation of a human with a height measurement of 2 meters (in the 95th percentile) and a weight measurement of 50kg (in the 5th percentile). Both types of outliers can affect the outcome of an analysis but are detected and treated differently.

Tutorial on univariate outliers using Python

This first post will deal with the detection of univariate outliers, followed by a second article on multivariate outliers. In a third article, I will write about how outliers of both types can be treated.

Outliers can be discovered in various ways, including statistical methods, proximity-based methods, or supervised outlier detection. In this article series, I will solely focus on commonly used *statistical methods*.

sklearn library.

```
#Load libraries
from sklearn.datasets import load_boston
import numpy as np
import pandas as pd
import seaborn as sns
from scipy import stats

#Load data
X, y = load_boston(return_X_y=True)

#Create data frame
boston = load_boston()
columns = boston.feature_names
df = pd.DataFrame(X, columns = columns)
```

univariateoutliers_intro hosted with ❤ by **GitHub** **view raw**

Visualizing outliers

A first and useful step in detecting univariate outliers is the visualization of a variables' distribution. Typically, when conducting an EDA, this needs to be done for all interesting variables of a data set individually. An easy way to visually summarize the distribution of a variable is the **box plot**.

In a box plot, introduced by John Tukey in 1970, the data is divided into quartiles. It usually shows a rectangular box representing 25%-75% of a sample's observations, extended by so-called whiskers that reach the minimum and maximum data entry. Observations shown outside of the whiskers are outliers (explained in more detail below).

Let's see an example. The plot below shows the majority of variables included in the Boston housing dataset. To receive a quick overview of all variables' distributions, you can use a group plot. Be aware that variables can differ in scale, and adding all variables into one grid may lead to some hard to read charts. I ran df.describe() first to get an idea of each variable's scale and then created three group plots for three different variable groups. Here is an example of medium scaled variables:

```
#df.describe()

```

```
6
7   ax = sns.boxplot(data=df_2, orient="h", palette="Set2")
```

boxplot_group hosted with ❤ by **GitHub** **view raw**

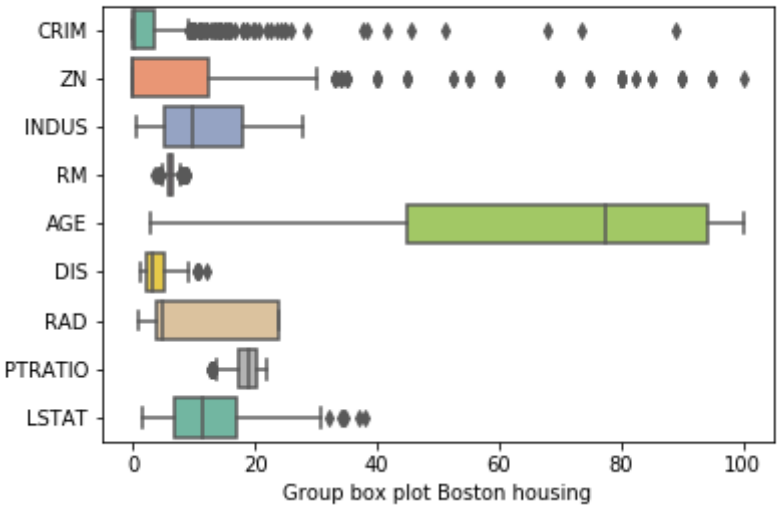


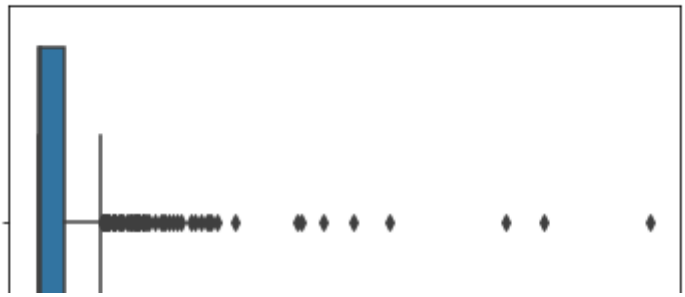
Image by author

It appears there are three variables, precisely AGE, INDUS, and RAD, with no univariate outlier observations. The remaining variables all have data points beyond their whiskers.

Let's look closer into the variable 'CRIM', which encodes the crime rate per capita by town. The individual box plot below shows that the crime rate in most towns is below 5%.

```
1   ax = sns.boxplot(x=df["CRIM"])
2   ax.set_xlabel('Crime rate per capita')
```

boxplot_CRIM hosted with ❤ by **GitHub** **view raw**



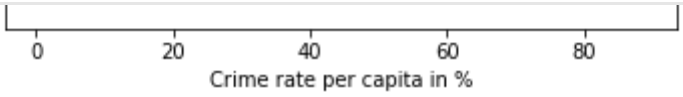


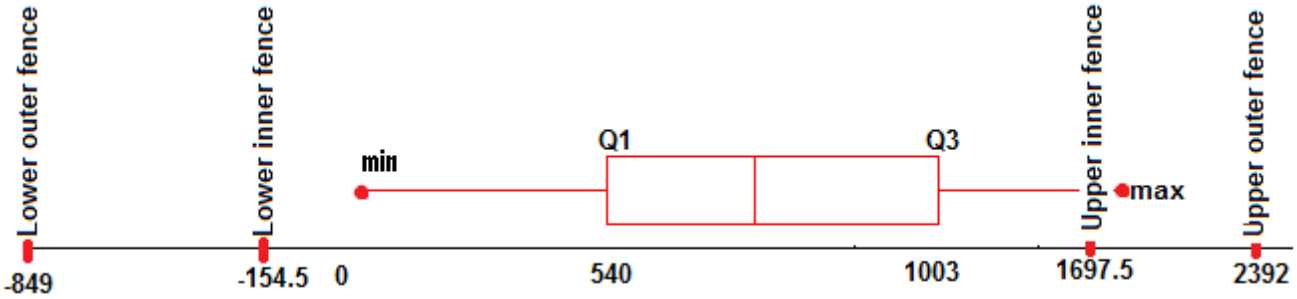
Image by author

Box plots are great to summarize and visualize the distribution of variables easily and quickly. However, they do not identify the actual indexes of the outlying observations. In the following, I will discuss three quantitative methods commonly used in statistics for the detection of univariate outliers:

- Tukey's box plot method
- Internally studentized residuals (AKA z-score method)
- Median Absolute Deviation method

Tukey's box plot method

Next to its visual benefits, the box plot provides useful statistics to identify individual observations as outliers. Tukey distinguishes between **possible** and **probable outliers**. A possible outlier is located between the **inner** and the **outer fence**, whereas a probable outlier is located outside the outer fence.



Example of a box plot including the inner and outer fences and minimum and maximum observations (known as whiskers). Image by Stephanie Glen on statisticsHowTo.com

While the inner (often confused with the whiskers) and outer fence are usually not shown on the actual box plot, they can be calculated using the **interquartile range** (IQR) like this:

IQR =Q3 - Q1, whereas q3 := 75th quartile and q1 := 25th quartile

*Inner fence = [Q1-1.5*IQR, Q3+1.5*IQR]*

The distribution's inner fence is defined as 1.5 x IQR below Q1, and 1.5 x IQR above Q3. The outer fence is defined as 3 x IQR below Q1, and 3 x IQR above Q3. Following Tukey, only the probable outliers are treated, which lie outside the outer fence. For the underlying example, this means:

```
#Tukey's method
def tukeys_method(df, variable):
    #Takes two parameters: dataframe & variable of interest as string
    q1 = df[variable].quantile(0.25)
    q3 = df[variable].quantile(0.75)
    iqr = q3-q1
    inner_fence = 1.5*iqr
    outer_fence = 3*iqr

    #inner fence lower and upper end
    inner_fence_le = q1-inner_fence
    inner_fence_ue = q3+inner_fence

    #outer fence lower and upper end
    outer_fence_le = q1-outer_fence
    outer_fence_ue = q3+outer_fence

    outliers_prob = []
    outliers_poss = []
    for index, x in enumerate(df[variable]):
        if x <= outer_fence_le or x >= outer_fence_ue:
            outliers_prob.append(index)
    for index, x in enumerate(df[variable]):
        if x <= inner_fence_le or x >= inner_fence_ue:
            outliers_poss.append(index)
    return outliers_prob, outliers_poss

probable_outliers_tm, possible_outliers_tm = tukeys_method(df, "CRIM")
print(probable_outliers_tm)
# [374, 375, 376, 378, 379, 380, 381, 384, 385, 386, 387, 398, 400, 403, 404, 405, 406,
# 410 412, 413, 414, 415, 417, 418, 425, 427, 437, 440, 468, 477]
print(possible_outliers_tm)
# [367, 371, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 384, 385, 386, 387, 388,
# 392, 394, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 409, 410, 411, 412, 413,
# 414, 415, 416, 417, 418, 419, 420, 422, 425, 426, 427, 429, 431, 434, 435, 436, 437,
# 438, 439, 440, 441, 443, 444, 445, 447, 448, 454, 468, 469, 477, 478, 479]
```

of all other variables in the data set by calling the function **tukeys_method** for each variable (line 28 above).

The great advantage of Tukey's box plot method is that the statistics (e.g. IQR, inner and outer fence) are robust to outliers, meaning to find one outlier is independent of all other outliers. Also, the statistics are easy to calculate. Furthermore, this method does not require a normal distribution of the data, which is often *not* guaranteed in real-life settings. If a distribution is highly skewed (usually found in real-life data), the Tukey method can be extended to the **log-IQ method**. Here, each value is transformed to its logarithm before calculating the inner and outer fences.

```
#Transform 'CRIM' to log
log_CRIM = np.log(df['CRIM'])
df['CRIM_man'] = df['CRIM']+1
log_CRIM = np.log(df['CRIM_man'])
df['CRIM_log'] = log_CRIM

#Plot
sns.distplot(df['CRIM_log'])

#Calculate probable and possible outliers using log-iq method
probable_outliers_logiq, possible_outliers_logiq = tukeys_method(df, 'CRIM_log')
print(probable_outliers_logiq)
print(possible_outliers_logiq)
```

outlier_tukeymethod_logiq hosted with ❤ by **GitHub** **view raw**

Internally studentized residuals AKA z-score method

Another commonly used method to detect univariate outliers is the **internally standardized residuals**, aka the **z-score method**. For each observation (Xn), it is measured how many standard deviations the data point is away from its mean ($\bar{X}$).

$$z_n = \frac{X_n - \bar{X}}{SD_X}$$

Following a common rule of thumb, if $z > C$, where C is usually set to 3, the observation is marked as an outlier. This rule stems from the fact that if a variable is normally distributed, 99.7% of all data points are located 3 standard deviations around the mean. Let's see on our example, which observations of 'CRIM' are detected to be outliers using the z-score:

```
#Internally studentized method (z-score)
def z_score_method(df, variable_name):
    #Takes two parameters: dataframe & variable of interest as string
    columns = df.columns
    z = np.abs(stats.zscore(df))
    threshold = 3
    outlier = []
    index=0
    for item in range(len(columns)):
        if columns[item] == variable_name:
            index = item
    for i, v in enumerate(z[:, index]):
        if v > threshold:
            outlier.append(i)
        else:
            continue
    return outlier

outlier_z = z_score_method(df, 'CRIM')
print(outlier_z)
# [380, 398, 404, 405, 410, 414, 418, 427]
```

outlier_zscore hosted with ❤ by **GitHub** **view raw**

When using the z-score method, 8 observations are marked as outliers. However, this method is highly limited as the distributions mean and standard deviation are sensitive to outliers. This means that finding one outlier is dependent on other outliers as every observation directly affects the mean.

Moreover, the z-score method assumes the variable of interest to be normally distributed. A more robust method that can be used instead is the externally studentized residuals. Here, the influence of the examined data point is removed from the calculation of the mean and standard deviation, like so:

$$i \in \mathbb{N}_{(n)}$$

$$SD_{X(n)} = \sqrt{\frac{1}{N-1} \sum_{i \in \mathbb{N}_{(n)}} (X_i - \overline{X}_{(n)})^2}$$

Image by author

Nevertheless, the externally studentized residuals have limitations as the mean and standard deviations are still sensitive to other outliers and still expect the variable of interest X to be normally distributed.

Median Absolute Deviation method

The **median absolute deviation method** (MAD) replaces the mean and standard deviation with more robust statistics, like the median and median absolute deviation. The median absolute deviation is defined as:

$$\text{MAD} = \text{median}(\left| X_i - \overline{X} \right|)$$

Image by author

The test statistic is calculated like the z-score using robust statistics. Also, to identify outlying observations, the same cut-off point of 3 is used. If the test statistic lies above 3, it is marked as an outlier. Compared to the internally (z-score) and externally studentized residuals, this method is more robust to outliers and does assume X to be parametrically distributed (Examples of discrete and continuous parametric distributions).

```python
#MAD method
def mad_method(df, variable_name):
    #Takes two parameters: dataframe & variable of interest as string
    columns = df.columns
    med = np.median(df, axis = 0)
    mad = np.abs(stats.median_absolute_deviation(df))
    threshold = 3
    outlier = []
    index=0
    for item in range(len(columns)):
        if columns[item] == variable_name:
            index == item
    for i, v in enumerate(df.loc[:,variable_name]):
        t = (v-med[index])/mad[index]
        if t > threshold:
            outlier.append(i)
        else:
            continue
    return outlier

outlier_mad = mad_method(df, 'CRIM')
print(outlier_mad)
#[20, 31, 32, 34, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 153, 154, 155,
# 156, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 171, 310, 356, 357,
# 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374,
# 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391,
# 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408,
# 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425,
# 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442,
# 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459,
# 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476,
# 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487]
```

outlier_madmethod hosted with ❤ by **GitHub** **view raw**

We can see that the MAD method detects 165 outliers for the crime rate per capita by town and with that the most outliers of all methods.

Wrapping up

There are different ways to detect univariate outliers, each one coming with advantages and disadvantages. The **z-score** needs to be applied critically due to its sensitivity to mean and standard deviation and its assumption of a normally distributed variable. The **MAD method** is often used instead and serves as a more

To decide on the right approach for your own data set, closely examine your variables' *distribution*, and use your *domain knowledge*.

In the next posting, I will address the detection of multivariate outliers.

Resources:

- https://link.springer.com/chapter/10.1007/978-3-319-43742-2_14
- https://www.rips-irsp.com/articles/10.5334/irsp.289/
- http://d-scholarship.pitt.edu/7948/1/Seo.pdf
- http://www.hermanaguinis.com/ORMoutliers.pdf
- https://www.statisticshowto.com/upper-and-lower-fences/
- https://wiki.analytica.com/index.php?title=Probability_Distributions