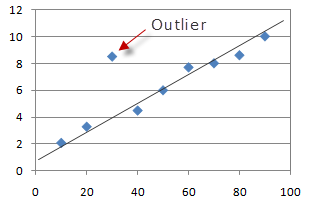
**Outlier Detection and Treatment: A Beginner's Guide**



One of the most important steps in data pre-processing is outlier detection and treatment. Machine learning algorithms are very sensitive to the range and distribution of data points. Data outliers can deceive the training process resulting in longer training times and less accurate models. Outliers are defined as samples that are significantly different from the remaining data. Those are points that lie outside the overall pattern of the distribution. Statistical measures such as mean, variance, and correlation are very susceptible to outliers.

A simple example of an outlier is here, a point that deviates from the overall pattern.



**Nature of Outliers:**

Outliers can occur in the dataset due to one of the following reasons,

1. Genuine extreme high and low values in the dataset
2. Introduced due to human or mechanical error
3. Introduced by replacing missing values

In some cases, the presence of outliers are informative and will require further study. For example, outliers are important in use-cases related to transaction management, where an outlier might be used to identify potentially fraudulent transactions.

In this article, I will discuss the following ways to identify outliers in your dataset and treat them.

**Outlier Detection**

* Extreme Value Analysis
* Z-score method
* K Means clustering-based approach
* Visualizing the data

**Outlier Treatment**

* Mean/Median or random Imputation
* Trimming
* Top, Bottom, and Zero Coding
* Discretization

However, none of these methods will deliver the objective truth about which of the observations are outliers. There is no rigid mathematical definition of what constitutes an outlier; determining whether or not an observation is an outlier is ultimately a subjective exercise that depends heavily on the business problem. So the methods discussed in this article can be a starting point to identify points in your data that should be treated as outliers.

**Methods to Detect Outliers:**

There are multiple methods to identify outliers in the dataset. I will discuss the following types in this article.

1. Extreme Value Analysis
2. Z-score method
3. K Means clustering-based approach
4. Visualizing the data

It is important to reiterate that these methods should not be used mechanically. They should be used to explore the data. They let you know which points might be worth a closer look.

**Dataset:**

I will be using the [Lending Club Loan Dataset](https://www.kaggle.com/wendykan/lending-club-loan-data) from Kaggle to demonstrate examples in this article.

**Importing Libraries**

import pandas as pd  
import numpy as np   
import seaborn as sns  
import matplotlib.pyplot as plt  
%matplotlib inline

**Importing dataset**

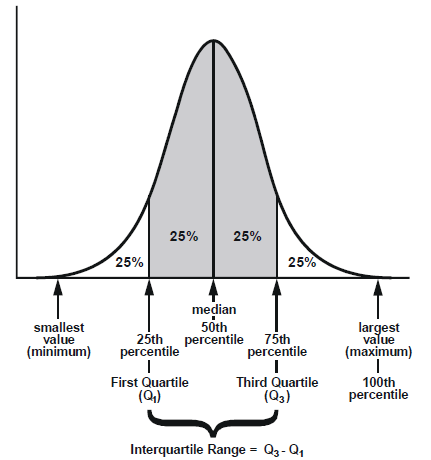
Now, let’s import the*Annual Income (annual\_inc)* column from the CSV file and identify the outliers.

use\_cols = ['annual\_inc']  
data = pd.read\_csv('loan.csv', usecols=use\_cols, nrows = 30000)

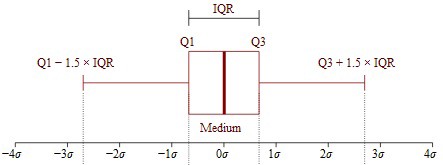
**Extreme Value Analysis:**

The most basic form of outlier detection is Extreme Value analysis. The key of this method is to determine the statistical tails of the underlying distribution of the variable and find the values at the extreme end of the tails.

In the case of a Gaussian Distribution, the outliers will lie outside the mean plus or minus 3 times the standard deviation of the variable.



If the variable is not normally distributed (not a Gaussian distribution), a general approach is to calculate the quantiles and then the inter-quartile range.



IQR (Inter quantiles range)= 75th quantile — 25th quantile

An outlier will be in the following upper and lower boundaries:

Upper Boundary = 75th quantile +(IQR \* 1.5)Lower Boundary = 25th quantile — (IQR \* 1.5)

Or for extreme cases:

Upper Boundary = 75th quantile +(IQR \* 3)Lower Boundary = 25th quantile — (IQR \* 3)

If the data point is above the upper boundary or below the lower boundary, it can be considered as an outlier.

**Code:**

First, let's calculate the Inter Quantile Range for our dataset,

IQR = data.annual\_inc.quantile(0.75) - data.annual\_inc.quantile(0.25)

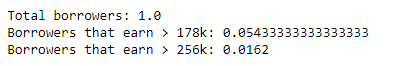
Using the IQR, we calculate the upper boundary using the formulas mentioned above,

upper\_limit = data.annual\_inc.quantile(0.75) + (IQR \* 1.5)  
upper\_limit\_extreme = data.annual\_inc.quantile(0.75) + (IQR \* 3)upper\_limit, upper\_limit\_extreme

https://miro.medium.com/max/209/1*t5tLk9O79vByt_CfIWeyxg.png

Now, let’s see the ratio of data points above the upper limit & extreme upper limit. i.e., the outliers.

total = np.float(data.shape[0])  
print('Total borrowers: {}'.format(data.annual\_inc.shape[0]/total))  
print('Borrowers that earn > 178k: {}'.format(data[data.annual\_inc>178000].shape[0]/total))  
print('Borrowers that earn > 256k: {}'.format(data[data.annual\_inc>256000].shape[0]/total))



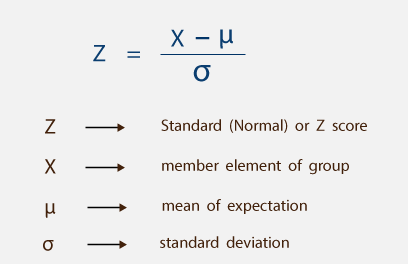
We can see that about 5% of the data is above the upper limit, and 1% of the data above the extreme upper limit.

**Standard Score (Z Score):**

A Z-score (or standard score) represents how many standard deviations a given measurement deviates from the mean. In other words, it merely re-scales or standardizes your data. A Z-score serves to specify the precise location of each observation within a distribution. The sign of the Z-score (+ or — ) indicates whether the score is above (+) or below ( — ) the mean.

The goal of taking Z-scores is to remove the effects of the location and scale of the data, allowing different datasets to be compared directly. The intuition behind the Z-score method of outlier detection is that, once we’ve centered and rescaled the data, anything that is too far from zero (the threshold is usually a Z-score of 3 or -3) should be considered an outlier.

The formula to calculate Z score is,



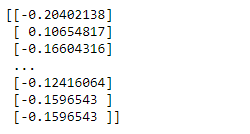
**Code:**

Importing libraries

from scipy import stats

Calculating Z score

z = stats.zscore(data)  
print(z)



Threshold > 3

threshold = 3  
print(np.where(z > 3))

https://miro.medium.com/max/538/1*UreImxd5-4Vc7nCuTx9cFg.png

In the above output, the first array contains the list of row numbers and the second array respective column numbers.

**Clustering Method:**

Clustering is a popular technique used to group similar data points or objects in groups or clusters. It can also be used as an important tool for outlier analysis. In this approach, we start by grouping similar kinds of objects.

We are going to use K-Means clustering, which will help us cluster the data points (annual income values in our case). The implementation that we are going to be using for KMeans uses Euclidean distance to group similar objects. Let’s get started.

**Code:**

**Importing Libraries**

We will now import the kmeans module from scipy.cluster.vq.

SciPystands for *Sci*entific Python and provides a variety of convenient utilities for performing scientific experiments.

from scipy.cluster.vq import kmeans  
from scipy.cluster.vq import vq

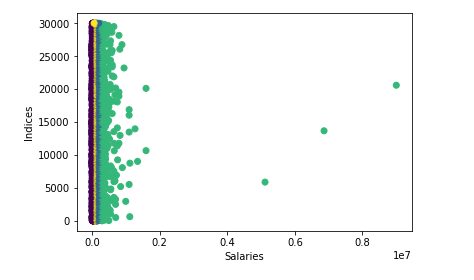
Now, let's convert the data into a numpy array and apply the K-Means function. We have to give two inputs — data and the number of clusters to be formed.

data\_raw = data['disbursed\_amount'].value  
centroids, avg\_distance = kmeans(data\_raw, 4)  
groups, cdist = vq(data\_raw, centroids)

Centroids are the center of the clusters generated by means(), and avg\_distance is the averaged Euclidean distance between the data points and the centroids generated by kmeans(). The next step is to call the vq() method. It returns the groups (clusters) of the data points and the distances between the data points and its nearest groups.

Let’s now plot the groups we have got.

y = np.arange(0,30000)  
plt.scatter(data\_raw, y , c=groups)  
plt.xlabel('Salaries')  
plt.ylabel('Indices')  
plt.show()



I am sure you are able to identify the outliers from the above graph.

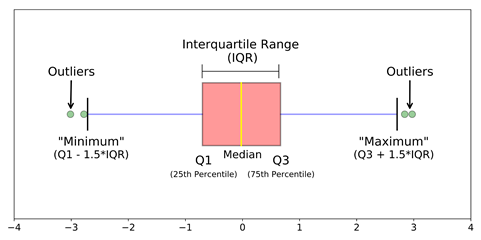
**Graphical Approach:**

As I mentioned in my [previous article](https://towardsdatascience.com/exploratory-data-analysis-in-python-ebdf643a33f6), Box plots, histograms, and Scatter plots are majorly used to identify outliers in the dataset.

**Box Plots**

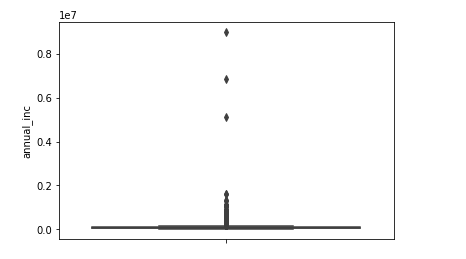
Box plot diagram, also termed as Whisker’s plot, is a graphical method typically depicted by quartiles and inter quartiles that helps in defining the upper limit and lower limit beyond which any data lying will be considered as outliers.

In brief, quantiles are points in a distribution that relates to the rank order of values in that distribution. For a given sample, you can find any quantile by sorting the sample. The middle value of the sorted sample is the middle quantile or the 50th percentile (also known as the median of the sample).



The very purpose of box plots is to identify outliers in the data series before making any further analysis so that the conclusion made from the study gives more accurate results not influenced by any extremes or abnormal values.

sns.boxplot(y='annual\_inc', data = data)

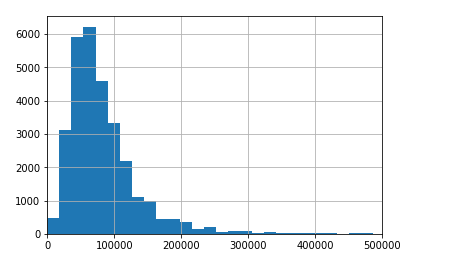


Here, outliers are observations that are numerically distant from the rest of the data. When reviewing a boxplot, an outlier is a data point that is located outside the fences (“whiskers”) of the boxplot.

**Histograms**

Histograms are one of the most common graphs used to display numeric data and finding the distribution of the dataset. An outlier is an observation that lies outside the overall pattern of distribution.

fig = data.annual\_inc.hist(bins=500)  
fig.set\_xlim(0,500000)

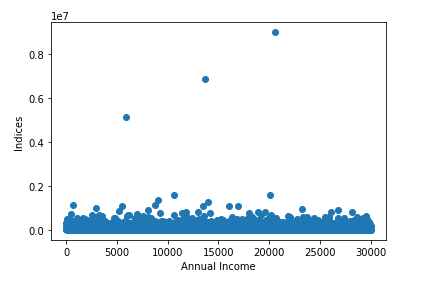


Here, the data points at the far right end of the x-axis can be considered outliers.

**Scatter Plots**

Scatter plots are used to find the association between two variables, and that association often has a pattern. We call a data point an outlier if it doesn’t fit the pattern.

data\_raw **= d**ata['annual\_inc'].values  
y = np.arange(0,30000)  
plt.scatter(data\_raw, y)  
plt.xlabel('Annual Income')  
plt.ylabel('Indices')  
plt.show()



**Methods to Pre-Process Outliers:**

1. Mean/Median or random Imputation
2. Trimming
3. Top, Bottom, and Zero Coding
4. Discretization

**Mean / Median / Random Sampling:**

Suppose we have reasons to believe that outliers are due to mechanical errors or problems during measurement. That means, the outliers are in nature similar to missing data, then any method used for missing data imputation can we used to replace outliers. The number are outliers are small (otherwise, they won't be called outliers), and it's reasonable to use mean/median/random imputation to replace them.

I will discuss the imputation of missing values in a separate article dedicated to Missing Values. In the meantime, if you need any sources for the same, [check this out](https://towardsdatascience.com/6-different-ways-to-compensate-for-missing-values-data-imputation-with-examples-6022d9ca0779).

**Trimming:**

In this method, we discard the outliers completely. That is, eliminate the data points that are considered as outliers. In situations where you won’t be removing a large number of values from the dataset, trimming is a good and fast approach.

index = data[(data['annual\_inc'] >= 256000)].index  
data.drop(index, inplace=True)

Here we use the pandas' drop method to remove all the records that are more than the upper limit value we found using extreme value analysis.

**Top / bottom / zero Coding:**

**Top Coding** means capping the maximum of the distribution at an arbitrary set value. A top coded variable is one for which data points above an upper bound are censored. By implementing top coding, the outlier is capped at a certain maximum value and looks like many other observations.

**Bottom coding** is analogous but on the left side of the distribution. That is, all values below a certain threshold are capped to that threshold. If the threshold is zero, then it is known as **zero-coding**. For example, for variables like “age” or “earnings”, it is not possible to have negative values. Thus it’s reasonable to cap the lowest value to zero.

**Code:**

print('Annual Income > 256000: {}'.format(data[data.annual\_inc>256000].shape[0]))  
print('Percentage of outliers: {}'.format(data[data.annual\_inc>256000].shape[0]/np.float(data.shape[0])))

https://miro.medium.com/max/304/1*D5daurhwOhcQJQagdnSSuA.png

In this step, we are capping the data points with values greater than 256000 to 256000.

data.loc[data.annual\_inc>256000,'annual\_inc'] = 256000  
data.annual\_inc.max()

Now, the maximum value will be displayed as 256000.

**Discretization:**

Discretization is the process of transforming continuous variables into discrete variables by creating a set of contiguous intervals that spans the range of the variable’s values. Thus, these outlier observations no longer differ from the rest of the values at the tails of the distribution, as they are now all together in the same interval/bucket.

There are several approaches to transform continuous variables into discrete ones. This process is also known as **binning**, with each bin being each interval.

**Discretization methods**

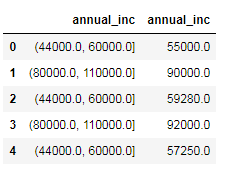
* Equal width binning
* Equal frequency binning

**Equal frequency discretization**

Equal frequency binning divides the possible values of the variable into N bins, where each bin carries the same amount of observations. This is particularly useful for skewed variables as it spreads the observations over the different bins equally. Typically, we find the interval boundaries by determining the quantiles. This would help in minimal loss of information and produces better results.

Here we are creating 5 bins using the pandas qcut function ( Quantile-based discretization function)

income\_discretised, intervals = pd.qcut(data.annual\_inc, 5, labels=None, retbins=True, precision=3, duplicates='raise')  
pd.concat([income\_discretised, data.annual\_inc], axis=1).head(5)



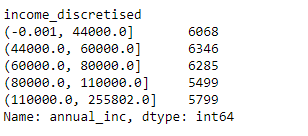
And the intervals are,

intervals

https://miro.medium.com/max/516/1*oAlR40abRgmTl5DmT0eZBg.png

Below we can see that there is almost an equal number of observations in each interval.

temp = pd.concat([income\_discretised, data.annual\_inc], axis=1)  
temp.columns = ['income\_discretised', 'annual\_inc']  
temp.groupby('income\_discretised')['annual\_inc'].count()



**Equal width discretization**

Equal width binning divides the scope of possible values into N bins of the same width. The width is determined by the range of values in the variable and the number of bins we wish to use to divide the variable.

width = (max value — min value) / N

For example if the values of the variable vary between 0 and 100, we create 5 bins like this: width = (100–0) / 5 = 20. The first and final bins (0–20 and 80–100) can be expanded to accommodate outliers (that is, values under 0 or greater than 100 would be placed in those bins as well).

There is no rule of thumb to define N. It depends on the use case.

Code:

income\_range = data.annual\_inc.max() - data.annual\_inc.min()  
min\_value = int(np.floor(data.annual\_inc.min()))  
max\_value = int(np.ceil(data.annual\_inc.max()))  
   
# let's round the bin width  
inter\_value = int(np.round(income\_range/5))  
   
min\_value, max\_value, inter\_value

https://miro.medium.com/max/184/1*lQw9QMV29mgVNc7xUwO1vw.png

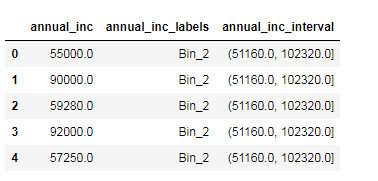
Now we are calculating the intervals,

intervals = [i for i in range(min\_value, max\_value+inter\_value, inter\_value)]  
labels = ['Bin\_'+str(i) for i in range(1,len(intervals))]  
print(intervals)  
print(labels)

https://miro.medium.com/max/468/1*kBMSVN66x-6_n1HUSZz3UQ.png

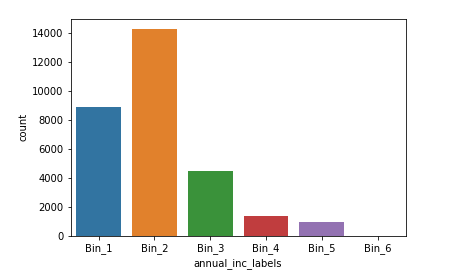
Finally, we use the pandas cut function to segment and sort data values into bins.

data['annual\_inc\_labels'] = pd.cut(x = data.annual\_inc, bins=intervals, labels=labels, include\_lowest=True)  
data['annual\_inc\_interval'] = pd.cut(x = data.annual\_inc, bins=intervals, include\_lowest=True)  
data.head(5)



We can count of data in each bin using a count plot, as shown below. We can see that the majority the people in the given sample dataset have their annual income under 10000

sns.countplot(data.annual\_inc\_labels)



I hope you found this article useful. Feel free to leave your thoughts!

References: