How to Remove Outliers for Machine Learning

by Jason Brownlee on April 25, 2018 in Data Preparation

Tweet Tweet Share Share

Last Updated on August 18, 2020

When modeling, it is important to clean the data sample to ensure that the observations best represent the problem.

Sometimes a dataset can contain extreme values that are outside the range of what is expected and unlike the other data. These are called outliers and often machine learning modeling and model skill in general can be improved by understanding and even removing these outlier values.

In this tutorial, you will discover outliers and how to identify and remove them from your machine learning dataset.

After completing this tutorial, you will know:

- That an outlier is an unlikely observation in a dataset and may have one of many causes.
- How to use simple univariate statistics like standard deviation and interquartile range to identify and remove outliers from a data sample.
- How to use an outlier detection model to identify and remove rows from a training dataset in order to lift predictive modeling performance.

Kick-start your project with my new book Data Preparation for Machine Learning, including *step-by-step tutorials* and the *Python source code* files for all examples.

Let's get started.

- **Update May/2018**: Fixed bug when filtering samples via outlier limits.
- **Update May/2020**: Updated to demonstrate on a real dataset.

How to Use Statistics to Identify Outliers in Data
Photo by Jeff Richardson, some rights reserved.

Tutorial Overview

This tutorial is divided into five parts; they are:

1. What are Outliers?
2. Test Dataset
3. Standard Deviation Method
4. Interquartile Range Method
5. Automatic Outlier Detection

Data Preparation for Machine Learning (7-Day Mini-Course)

How to Calculate Feature Importance With Python

Recursive Feature Elimination (RFE) for Feature Selection in Python

How to Remove Outliers for Machine Learning

What are Outliers?

An outlier is an observation that is unlike the other observations.

It is rare, or distinct, or does not fit in some way.

> *We will generally define outliers as samples that are exceptionally far from the mainstream of the data.*

— Page 33, Applied Predictive Modeling, 2013.

Outliers can have many causes, such as:

- Measurement or input error.
- Data corruption.
- True outlier observation (e.g. Michael Jordan in basketball).

There is no precise way to define and identify outliers in general because of the specifics of each dataset. Instead, you, or a domain expert, must interpret the raw observations and decide whether a value is an outlier or not.

> *Even with a thorough understanding of the data, outliers can be hard to define. […] Great care should be taken not to hastily remove or change values, especially if the sample size is small.*

— Page 33, Applied Predictive Modeling, 2013.

Nevertheless, we can use statistical methods to identify observations that appear to be rare or unlikely given the available data.

> *Identifying outliers and bad data in your dataset is probably one of the most difficult parts of data cleanup, and it takes time to get right. Even if you have a deep understanding of statistics and how outliers might affect your data, it's always a topic to explore cautiously.*

— Page 167, Data Wrangling with Python, 2016.

This does not mean that the values identified are outliers and should be removed. But, the tools described in this tutorial can be helpful in shedding light on rare events that may require a second look.

A good tip is to consider plotting the identified outlier values, perhaps in the context of non-outlier values to see if there are any systematic relationship or pattern to the outliers. If there is, perhaps they are not outliers and can be explained, or perhaps the outliers themselves can be identified more systematically.

Data Preparation for Machine Learning (7-Day Mini-Course)

How to Calculate Feature Importance With Python

Recursive Feature Elimination (RFE) for Feature Selection in Python

How to Remove Outliers for Machine Learning

Data Preparation for Machine Learning (7-Day Mini-Course)

How to Calculate Feature Importance With Python

Recursive Feature Elimination (RFE) for Feature Selection in Python

How to Remove Outliers for Machine Learning

Data Preparation for Machine Learning (7-Day Mini-Course)

How to Calculate Feature Importance With Python

Recursive Feature Elimination (RFE) for Feature Selection in Python

How to Remove Outliers for Machine Learning

Data Preparation for Machine Learning (7-Day Mini-Course)

How to Calculate Feature Importance With Python

Recursive Feature Elimination (RFE) for Feature Selection in Python

Test Dataset

Before we look at outlier identification methods, let's define a dataset we can use to test the methods.

We will generate a population 10,000 random numbers drawn from a Gaussian distribution with a mean of 50 and a standard deviation of 5.

Numbers drawn from a Gaussian distribution will have outliers. That is, by virtue of the distribution itself, there will be a few values that will be a long way from the mean, rare values that we can identify as outliers.

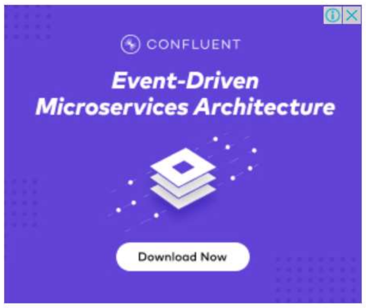
We will use the *randn()* function to generate random Gaussian values with a mean of 0 and a standard deviation of 1, then multiply the results by our own standard deviation and add the mean to shift the values into the preferred range.

The pseudorandom number generator is seeded to ensure that we get the same sample of numbers each time the code is run.

```
# generate gaussian data
from numpy.random import seed
from numpy.random import randn
from numpy import mean
from numpy import std
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(10000) + 50
# summarize
print('mean=%.3f stdv=%.3f' % (mean(data), std(data)))
```

Running the example generates the sample and then prints the mean and standard deviation. As expected, the values are very close to the expected values.

```
mean=50.049 stdv=4.994
```



Standard Deviation Method

If we know that the distribution of values in the sample is Gaussian or Gaussian-like, we can use the standard deviation of the sample as a cut-off for identifying outliers.

The Gaussian distribution has the property that the standard deviation from the mean can be used to reliably summarize the percentage of values in the sample.

For example, within one standard deviation of the mean will cover 68% of the data.

So, if the mean is 50 and the standard deviation is 5, as in the test dataset above, then all data in the sample between 45 and 55 will account for about 68% of the data sample. We can cover more of the data sample if we expand the range as follows:

- 1 Standard Deviation from the Mean: 68%
- 2 Standard Deviations from the Mean: 95%
- 3 Standard Deviations from the Mean: 99.7%

A value that falls outside of 3 standard deviations is part of the distribution, but it is an unlikely or rare event at approximately 1 in 370 samples.

Three standard deviations from the mean is a common cut-off in practice for identifying outliers in a Gaussian or Gaussian-like distribution. For smaller samples of data, perhaps a value of 2 standard deviations (95%) can be used, and for larger samples, perhaps a value of 4 standard deviations (99.9%) can be used.

> *Given mu and sigma, a simple way to identify outliers is to compute a z-score for every xi, which is defined as the number of standard deviations away xi is from the mean […] Data values that have a z-score sigma greater than a threshold, for example, of three, are declared to be outliers.*

— Page 19, Data Cleaning, 2019.

Let's make this concrete with a worked example.

Sometimes, the data is standardized first (e.g. to a Z-score with zero mean and unit variance) so that the

How to Remove Outliers for Machine Learning
Data Preparation for Machine Learning (7-Day Mini-Course)
How to Calculate Feature Importance With Python
Recursive Feature Elimination (RFE) for Feature Selection in Python
How to Remove Outliers for Machine Learning
Data Preparation for Machine Learning (7-Day Mini-Course)
How to Calculate Feature Importance With Python
Recursive Feature Elimination (RFE) for Feature Selection in Python
How to Remove Outliers for Machine Learning
Data Preparation for Machine Learning (7-Day Mini-Course)
How to Calculate Feature Importance With Python
Recursive Feature Elimination (RFE) for Feature Selection in Python
How to Remove Outliers for Machine Learning
Data Preparation for Machine Learning (7-Day Mini-Course)
How to Calculate Feature Importance With Python
Recursive Feature Elimination (RFE) for Feature Selection in Python
How to Remove Outliers for Machine Learning
Data Preparation for Machine Learning (7-Day Mini-Course)

Sometimes, the data is standardized first (e.g. to a Z-score with zero mean and unit variance) so that the outlier detection can be performed using standard Z-score cut-off values. This is a convenience and is not required in general, and we will perform the calculations in the original scale of the data here to make things clear.

We can calculate the mean and standard deviation of a given sample, then calculate the cut-off for identifying outliers as more than 3 standard deviations from the mean.

```
...
# calculate summary statistics
data_mean, data_std = mean(data), std(data)
# identify outliers
cut_off = data_std * 3
lower, upper = data_mean - cut_off, data_mean + cut_off
```

We can then identify outliers as those examples that fall outside of the defined lower and upper limits.

```
...
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
```

Alternately, we can filter out those values from the sample that are not within the defined limits.

```
...
# remove outliers
outliers_removed = [x for x in data if x > lower and x < upper]
```

We can put this all together with our sample dataset prepared in the previous section.

The complete example is listed below.

```
# identify outliers with standard deviation
from numpy.random import seed
from numpy.random import randn
from numpy import mean
from numpy import std
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(10000) + 50
# calculate summary statistics
data_mean, data_std = mean(data), std(data)
# identify outliers
cut_off = data_std * 3
lower, upper = data_mean - cut_off, data_mean + cut_off
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
print('Identified outliers: %d' % len(outliers))
# remove outliers
outliers_removed = [x for x in data if x >= lower and x <= upper]
print('Non-outlier observations: %d' % len(outliers_removed))
```

Running the example will first print the number of identified outliers and then the number of observations that are not outliers, demonstrating how to identify and filter out outliers respectively.

```
Identified outliers: 29
Non-outlier observations: 9971
```

So far we have only talked about univariate data with a Gaussian distribution, e.g. a single variable. You can use the same approach if you have multivariate data, e.g. data with multiple variables, each with a different Gaussian distribution.

You can imagine bounds in two dimensions that would define an ellipse if you have two variables. Observations that fall outside of the ellipse would be considered outliers. In three dimensions, this would be an ellipsoid, and so on into higher dimensions.

Alternately, if you knew more about the domain, perhaps an outlier may be identified by exceeding the limits on one or a subset of the data dimensions.

Interquartile Range Method

Not all data is normal or normal enough to treat it as being drawn from a Gaussian distribution.

A good statistic for summarizing a non-Gaussian distribution sample of data is the Interquartile Range, or IQR for short.

The IQR is calculated as the difference between the 75th and the 25th percentiles of the data and defines the box in a box and whisker plot.

Remember that percentiles can be calculated by sorting the observations and selecting values at specific

How to Calculate Feature Importance With Python
Recursive Feature Elimination (RFE) for Feature Selection in Python
How to Remove Outliers for Machine Learning
Data Preparation for Machine Learning (7-Day Mini-Course)
How to Calculate Feature Importance With Python
Recursive Feature Elimination (RFE) for Feature Selection in Python
How to Remove Outliers for Machine Learning
Data Preparation for Machine Learning (7-Day Mini-Course)
How to Calculate Feature Importance With Python
Recursive Feature Elimination (RFE) for Feature Selection in Python
How to Remove Outliers for Machine Learning
Data Preparation for Machine Learning (7-Day Mini-Course)
How to Calculate Feature Importance With Python
Recursive Feature Elimination (RFE) for Feature Selection in Python
How to Remove Outliers for Machine Learning
Data Preparation for Machine Learning (7-Day Mini-Course)
How to Calculate Feature Importance With Python
Recursive Feature Elimination (RFE) for Feature Selection in Python
How to Remove Outliers for Machine

Remember that percentiles can be calculated by sorting the observations and selecting values at specific indices. The 50th percentile is the middle value, or the average of the two middle values for an even number of examples. If we had 10,000 samples, then the 50th percentile would be the average of the 5000th and 5001st values.

We refer to the percentiles as quartiles ("*quart*" meaning 4) because the data is divided into four groups via the 25th, 50th and 75th values.

The IQR defines the middle 50% of the data, or the body of the data.

> *Statistics-based outlier detection techniques assume that the normal data points would appear in high probability regions of a stochastic model, while outliers would occur in the low probability regions of a stochastic model.*

— Page 12, Data Cleaning, 2019.

The IQR can be used to identify outliers by defining limits on the sample values that are a factor *k* of the IQR below the 25th percentile or above the 75th percentile. The common value for the factor *k* is the value 1.5. A factor k of 3 or more can be used to identify values that are extreme outliers or "*far outs*" when described in the context of box and whisker plots.

On a box and whisker plot, these limits are drawn as fences on the whiskers (or the lines) that are drawn from the box. Values that fall outside of these values are drawn as dots.

We can calculate the percentiles of a dataset using the *percentile()* NumPy function that takes the dataset and specification of the desired percentile. The IQR can then be calculated as the difference between the 75th and 25th percentiles.

```
...
# calculate interquartile range
q25, q75 = percentile(data, 25), percentile(data, 75)
iqr = q75 - q25
```

We can then calculate the cutoff for outliers as 1.5 times the IQR and subtract this cut-off from the 25th percentile and add it to the 75th percentile to give the actual limits on the data.

```
...
# calculate the outlier cutoff
cut_off = iqr * 1.5
lower, upper = q25 - cut_off, q75 + cut_off
```

We can then use these limits to identify the outlier values.

```
...
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
```

We can also use the limits to filter out the outliers from the dataset.

```
...
# remove outliers
outliers_removed = [x for x in data if x > lower and x < upper]
```

We can tie all of this together and demonstrate the procedure on the test dataset.

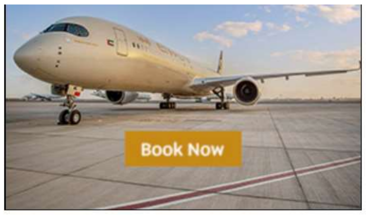
The complete example is listed below.

```
# identify outliers with interquartile range
from numpy.random import seed
from numpy.random import randn
from numpy import percentile
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(10000) + 50
# calculate interquartile range
q25, q75 = percentile(data, 25), percentile(data, 75)
iqr = q75 - q25
print('Percentiles: 25th=%.3f, 75th=%.3f, IQR=%.3f' % (q25, q75, iqr))
# calculate the outlier cutoff
cut_off = iqr * 1.5
lower, upper = q25 - cut_off, q75 + cut_off
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
print('Identified outliers: %d' % len(outliers))
# remove outliers
outliers_removed = [x for x in data if x >= lower and x <= upper]
print('Non-outlier observations: %d' % len(outliers_removed))
```

Running the example first prints the identified 25th and 75th percentiles and the calculated IQR. The number of outliers identified is printed followed by the number of non-outlier observations.

```
Percentiles: 25th=46.685, 75th=53.359, IQR=6.674
Identified outliers: 81
Non-outlier observations: 9919
```

The approach can be used for multivariate data by calculating the limits on each variable in the dataset in turn, and taking outliers as observations that fall outside of the rectangle or hyper-rectangle.



Automatic Outlier Detection

In machine learning, an approach to tackling the problem of outlier detection is one-class classification.

One-Class Classification, or OCC for short, involves fitting a model on the "*normal*" data and predicting whether new data is normal or an outlier/anomaly.

> *A one-class classifier aims at capturing characteristics of training instances, in order to be able to distinguish between them and potential outliers to appear.*

— Page 139, Learning from Imbalanced Data Sets, 2018.

A one-class classifier is fit on a training dataset that only has examples from the normal class. Once prepared, the model is used to classify new examples as either normal or not-normal, i.e. outliers or anomalies.

A simple approach to identifying outliers is to locate those examples that are far from the other examples in the feature space.

This can work well for feature spaces with low dimensionality (few features), although it can become less reliable as the number of features is increased, referred to as the curse of dimensionality.

The local outlier factor, or LOF for short, is a technique that attempts to harness the idea of nearest neighbors for outlier detection. Each example is assigned a scoring of how isolated or how likely it is to be outliers based on the size of its local neighborhood. Those examples with the largest score are more likely to be outliers.

> *We introduce a local outlier (LOF) for each object in the dataset, indicating its degree of outlier-ness.*

— LOF: Identifying Density-based Local Outliers, 2000.

The scikit-learn library provides an implementation of this approach in the LocalOutlierFactor class.

We can demonstrate the LocalOutlierFactor method on a predictive modelling dataset.

We will use the Boston housing regression problem that has 13 inputs and one numerical target and requires learning the relationship between suburb characteristics and house prices.

The dataset can be downloaded from here:

- Boston Housing Dataset (housing.csv)
- Boston Housing Dataset Details (housing.names)

Looking in the dataset, you should see that all variables are numeric.

```
0.00632,18.00,2.310,0,0.5380,6.5750,65.20,4.0900,1,296.0,15.30,396.90,4.98,24.00
0.02731,0.00,7.070,0,0.4690,6.4210,78.90,4.9671,2,242.0,17.80,396.90,9.14,21.60
0.02729,0.00,7.070,0,0.4690,7.1850,61.10,4.9671,2,242.0,17.80,392.83,4.03,34.70
0.03237,0.00,2.180,0,0.4580,6.9980,45.80,6.0622,3,222.0,18.70,394.63,2.94,33.40
0.06905,0.00,2.180,0,0.4580,7.1470,54.20,6.0622,3,222.0,18.70,396.90,5.33,36.20
...
```

No need to download the dataset, we will download it automatically.

First, we can load the dataset as a NumPy array, separate it into input and output variables and then split it into train and test datasets.

The complete example is listed below.

```python
# load and summarize the dataset
from pandas import read_csv
from sklearn.model_selection import train_test_split
# load the dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
df = read_csv(url, header=None)
# retrieve the array
data = df.values
# split into inpiut and output elements
X, y = data[:, :-1], data[:, -1]
# summarize the shape of the dataset
print(X.shape, y.shape)
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
# summarize the shape of the train and test sets
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

Running the example loads the dataset and first reports the total number of rows and columns in the dataset, then the data number of examples allocated to the train and test datasets.

```
(506, 13) (506,)
(339, 13) (167, 13) (339,) (167,)
```

It is a regression predictive modeling problem, meaning that we will be predicting a numeric value. All input variables are also numeric.

In this case, we will fit a linear regression algorithm and evaluate model performance by training the model on the test dataset and making a prediction on the test data and evaluate the predictions using the mean absolute error (MAE).

The complete example of evaluating a linear regression model on the dataset is listed below.

```
# evaluate model on the raw dataset
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error
# load the dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
df = read_csv(url, header=None)
# retrieve the array
data = df.values
# split into inpiut and output elements
X, y = data[:, :-1], data[:, -1]
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
# fit the model
model = LinearRegression()
model.fit(X_train, y_train)
# evaluate the model
yhat = model.predict(X_test)
# evaluate predictions
mae = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % mae)
```

Running the example fits and evaluates the model then reports the MAE.

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

In this case, we can see that the model achieved a MAE of about 3.417.

```
MAE: 3.417
```

Next, we can try removing outliers from the training dataset.

The expectation is that the outliers are causing the linear regression model to learn a bias or skewed understanding of the problem, and that removing these outliers from the training set will allow a more effective model to be learned.

We can achieve this by defining the LocalOutlierFactor model and using it to make a prediction on the training dataset, marking each row in the training dataset as normal (1) or an outlier (-1). We will use the default hyperparameters for the outlier detection model, although it is a good idea to tune the configuration to the specifics of your dataset.

```
...
# identify outliers in the training dataset
lof = LocalOutlierFactor()
yhat = lof.fit_predict(X_train)
```

We can then use these predictions to remove all outliers from the training dataset.

```
...
# select all rows that are not outliers
mask = yhat != -1
X_train, y_train = X_train[mask, :], y_train[mask]
```

We can then fit and evaluate the model as per normal.

The updated example of evaluating a linear regression model with outliers deleted from the training dataset is listed below.

```
# evaluate model on training dataset with outliers removed
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import LocalOutlierFactor
from sklearn.metrics import mean_absolute_error
# load the dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'
df = read_csv(url, header=None)
# retrieve the array
data = df.values
# split into inpiut and output elements
X, y = data[:, :-1], data[:, -1]
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
# summarize the shape of the training dataset
print(X_train.shape, y_train.shape)
# identify outliers in the training dataset
lof = LocalOutlierFactor()
yhat = lof.fit_predict(X_train)
# select all rows that are not outliers
mask = yhat != -1
X_train, y_train = X_train[mask, :], y_train[mask]
# summarize the shape of the updated training dataset
```

Data Preparation for Machine Learning (7-Day Mini-Course)

How to Calculate Feature Importance With Python

Recursive Feature Elimination (RFE) for Feature Selection in Python

How to Remove Outliers for Machine Learning

Data Preparation for Machine Learning (7-Day Mini-Course)

How to Calculate Feature Importance With Python

Recursive Feature Elimination (RFE) for Feature Selection in Python

How to Remove Outliers for Machine Learning

Data Preparation for Machine Learning (7-Day Mini-Course)

How to Calculate Feature Importance With Python

Recursive Feature Elimination (RFE) for Feature Selection in Python

How to Remove Outliers for Machine Learning

Data Preparation for Machine Learning (7-Day Mini-Course)

How to Calculate Feature Importance With Python

Recursive Feature Elimination (RFE) for

```
print(X_train.shape, y_train.shape)
# fit the model
model = LinearRegression()
model.fit(X_train, y_train)
# evaluate the model
yhat = model.predict(X_test)
# evaluate predictions
mae = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % mae)
```

Running the example fits and evaluates the linear regression model with outliers deleted from the training dataset.

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

Firstly, we can see that the number of examples in the training dataset has been reduced from 339 to 305, meaning 34 rows containing outliers were identified and deleted.

We can also see a reduction in MAE from about 3.417 by a model fit on the entire training dataset, to about 3.356 on a model fit on the dataset with outliers removed.

```
(339, 13) (339,)
(305, 13) (305,)
MAE: 3.356
```

The Scikit-Learn library provides other outlier detection algorithms that can be used in the same way such as the IsolationForest algorithm. For more examples of automatic outlier detection, see the tutorial:

- 4 Automatic Outlier Detection Algorithms in Python

Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- Develop your own Gaussian test dataset and plot the outliers and non-outlier values on a histogram.
- Test out the IQR based method on a univariate dataset generated with a non-Gaussian distribution.
- Choose one method and create a function that will filter out outliers for a given dataset with an arbitrary number of dimensions.

If you explore any of these extensions, I'd love to know.

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

Tutorials

- How to Identify Outliers in your Data
- 4 Automatic Outlier Detection Algorithms in Python

Books

- Applied Predictive Modeling, 2013.
- Data Cleaning, 2019.
- Data Wrangling with Python, 2016.

API

- seed() NumPy API
- randn() NumPy API
- mean() NumPy API
- std() NumPy API
- percentile() NumPy API

Feature Selection in Python
How to Remove Outliers for Machine Learning
Data Preparation for Machine Learning (7-Day Mini-Course)
How to Calculate Feature Importance With Python
Recursive Feature Elimination (RFE) for Feature Selection in Python
How to Remove Outliers for Machine Learning
Data Preparation for Machine Learning (7-Day Mini-Course)
How to Calculate Feature Importance With Python
Recursive Feature Elimination (RFE) for Feature Selection in Python
How to Remove Outliers for Machine Learning
Data Preparation for Machine Learning (7-Day Mini-Course)
How to Calculate Feature Importance With Python
Recursive Feature Elimination (RFE) for Feature Selection in Python
How to Remove Outliers for Machine Learning
Data Preparation for Machine Learning (7-Day Mini-Course)
How to Calculate Feature Importance With Python
Recursive Feature Elimination (RFE) for Feature Selection in Python
How to Remove Outliers for Machine Learning
Data Preparation for Machine Learning (7-