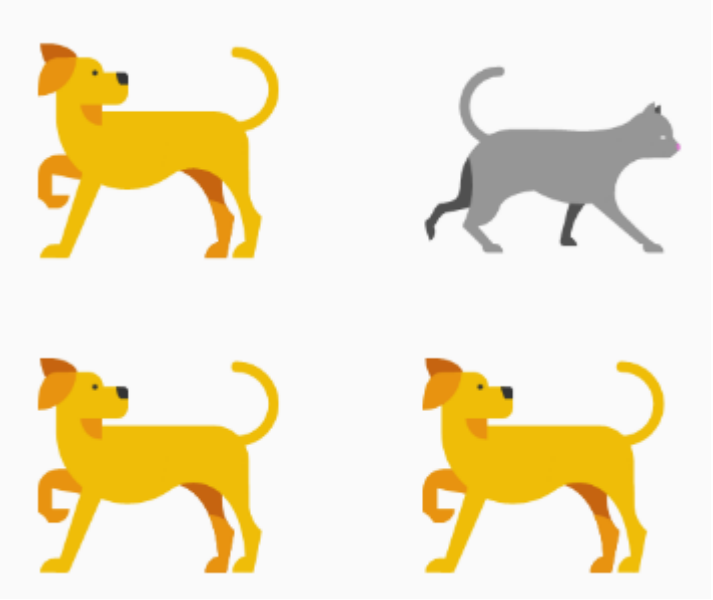


Anomaly detection is the process of detecting time-series data outliers; points on a given input time-series where the behavior isn't what was expected, or "weird".

Anomaly detection can be useful in lots of ways. For instance:

If you have a car, you might want to know: Is this oil gauge reading normal, or do I have a leak? If you're monitoring power consumption, you'd want to know: Is there an outage?

There are two types of time series anomalies that can be detected:

- **Spikes** indicate temporary bursts of anomalous behavior in the system.
- **Change points** indicate the beginning of persistent changes over time in the system.

In ML.NET, The IID Spike Detection or IID Change point Detection algorithms are suited for independent and identically distributed datasets . They assume that your input data is a sequence of data points that are independently sampled from one stationary distribution .

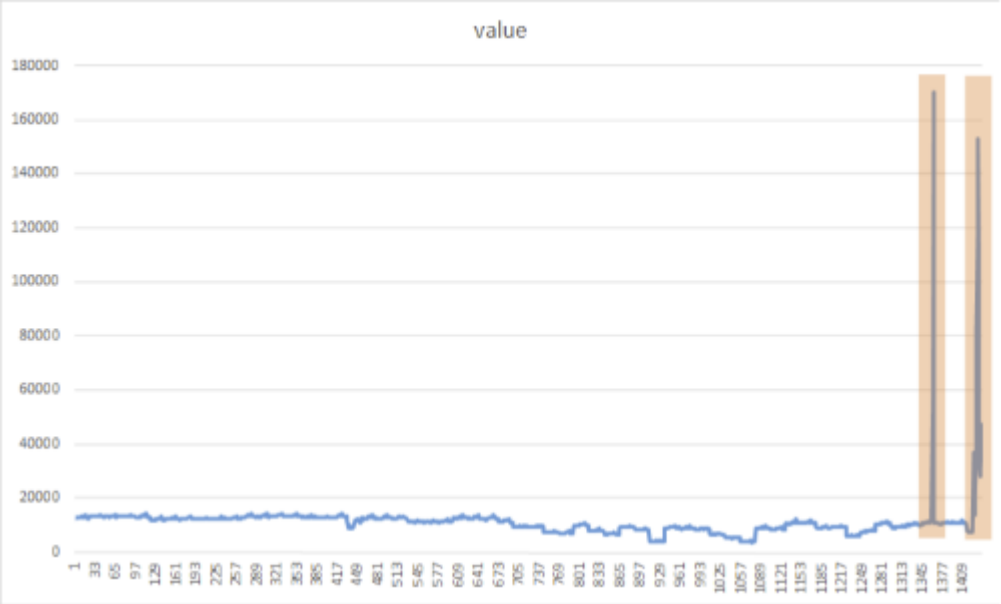
Unlike the models in the other tutorials, the time series anomaly detector transforms operate directly on input data. The `IEstimator.Fit()` method does not need training data to produce the transform. It does need the data schema though, which is provided by a data view generated from an empty list of `ProductSalesData`.

You'll analyze the same product sales data to detect spikes and change points. The building and training model process is the same for spike detection and change point detection; the main difference is the specific detection algorithm used.

Spike detection

The goal of spike detection is to identify sudden yet temporary bursts that significantly differ from the majority of the time series data values. It's important to detect these suspicious rare items, events, or observations in a timely manner to be minimized. The following approach can be used to detect a variety of anomalies such as: outages, cyber-attacks, or viral web content. The following

image is an example of spikes in a time series dataset:



Add the CreateEmptyDataView() method

Add the following method to `Program.cs`:

C#

```
IDataView CreateEmptyDataView(MLContext mlContext) {
    // Create empty DataView. We just need the schema to call Fit() for the time se-
ries transforms
    IEnumerable<ProductSalesData> enumerableData = new List<ProductSalesData>();
    return mlContext.Data.LoadFromEnumerable(enumerableData);
}
```

The `CreateEmptyDataView()` produces an empty data view object with the correct schema to be used as input to the `IEstimator.Fit()` method.

Create the DetectSpike() method

The `DetectSpike()` method:

- Creates the transform from the estimator.
- Detects spikes based on historical sales data.
- Displays the results.

1. Create the `DetectSpike()` method at the bottom of the **Program.cs** file using the following code:

   C#

   ```
   DetectSpike(MLContext mlContext, int docSize, IDataView productSales)
   {
   ```

```
}
```

2. Use the IidSpikeEstimator to train the model for spike detection. Add it to the `DetectSpike()` method with the following code:

```
C#
```

```
var iidSpikeEstimator = mlContext.Transforms.DetectIidSpike(outputColumnName:
nameof(ProductSalesPrediction.Prediction), inputColumnName:
nameof(ProductSalesData.numSales), confidence: 95d, pvalueHistoryLength: docSize
/ 4);
```

3. Create the spike detection transform by adding the following as the next line of code in the `DetectSpike()` method:

> **Tip**
>
> The `confidence` and `pvalueHistoryLength` parameters impact how spikes are detected. `confidence` determines how sensitive your model is to spikes. The lower the confidence, the more likely the algorithm is to detect "smaller" spikes. The `pvalueHistoryLength` parameter defines the number of data points in a sliding window. The value of this parameter is usually a percentage of the entire dataset. The lower the `pvalueHistoryLength`, the faster the model forgets previous large spikes.

```
C#
```

```
ITransformer iidSpikeTransform =
iidSpikeEstimator.Fit(CreateEmptyDataView(mlContext));
```

4. Add the following line of code to transform the `productSales` data as the next line in the `DetectSpike()` method:

```
C#
```

```
IDataView transformedData = iidSpikeTransform.Transform(productSales);
```

The previous code uses the Transform() method to make predictions for multiple input rows of a dataset.

5. Convert your `transformedData` into a strongly typed `IEnumerable` for easier display using the CreateEnumerable() method with the following code:

```
C#
```

```
var predictions = mlContext.Data.CreateEnumerable<ProductSalesPrediction>
(transformedData, reuseRowObject: false);
```

6. Create a display header line using the following Console.WriteLine() code:

```
C#
```

```
Console.WriteLine("Alert\tScore\tP-Value");
```

You'll display the following information in your spike detection results:

- `Alert` indicates a spike alert for a given data point.
- `Score` is the `ProductSales` value for a given data point in the dataset.
- `P-Value` The "P" stands for probability. The closer the p-value is to 0, the more likely the data point is an anomaly.

7. Use the following code to iterate through the `predictions IEnumerable` and display the results:

```
C#
```

```
foreach (var p in predictions)
{
    if (p.Prediction is not null)
    {
        var results = $"{p.Prediction[0]}\t{p.Prediction[1]:f2}
\t{p.Prediction[2]:F2}";

        if (p.Prediction[0] == 1)
        {
            results += " <-- Spike detected";
        }

        Console.WriteLine(results);
    }
}
Console.WriteLine("");
```

8. Add the call to the `DetectSpike()` method below the call to the `LoadFromTextFile()` method:

```
C#
```

```
DetectSpike(mlContext, _docsize, dataView);
```

Spike detection results

Your results should be similar to the following. During processing, messages are displayed. You may see warnings, or processing messages. Some of the messages have been removed from the

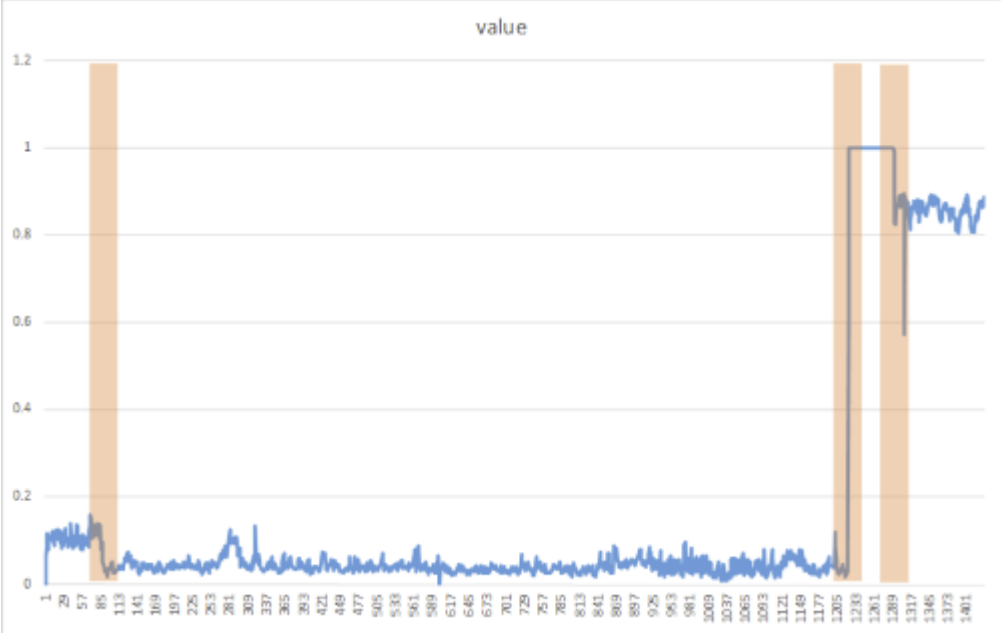
following results for clarity.

Console

```
Detect temporary changes in pattern
=============== Training the model ===============
=============== End of training process ===============
Alert   Score   P-Value
0       271.00  0.50
0       150.90  0.00
0       188.10  0.41
0       124.30  0.13
0       185.30  0.47
0       173.50  0.47
0       236.80  0.19
0       229.50  0.27
0       197.80  0.48
0       127.90  0.13
1       341.50  0.00 <-- Spike detected
0       190.90  0.48
0       199.30  0.48
0       154.50  0.24
0       215.10  0.42
0       278.30  0.19
0       196.40  0.43
0       292.00  0.17
0       231.00  0.45
0       308.60  0.18
0       294.90  0.19
1       426.60  0.00 <-- Spike detected
0       269.50  0.47
0       347.30  0.21
0       344.70  0.27
0       445.40  0.06
0       320.90  0.49
0       444.30  0.12
0       406.30  0.29
0       442.40  0.21
1       580.50  0.00 <-- Spike detected
0       412.60  0.45
1       687.00  0.01 <-- Spike detected
0       480.30  0.40
0       586.30  0.20
0       651.90  0.14
```

Change point detection

`Change points` are persistent changes in a time series event stream distribution of values, like level changes and trends. These persistent changes last much longer than `spikes` and could indicate catastrophic event(s). `Change points` are not usually visible to the naked eye, but can be detected in your data using approaches such as in the following method. The following image is an example of a change point detection:



Create the DetectChangepoint() method

The `DetectChangepoint()` method executes the following tasks:

- Creates the transform from the estimator.
- Detects change points based on historical sales data.
- Displays the results.

1. Create the `DetectChangepoint()` method, just after the `DetectSpike()` method declaration, using the following code:

```
C#
```

```
void DetectChangepoint(MLContext mlContext, int docSize, IDataView productSales)
{

}
```

2. Create the iidChangePointEstimator in the `DetectChangepoint()` method with the following code:

```
C#
```

```
var iidChangePointEstimator =
mlContext.Transforms.DetectIidChangePoint(outputColumnName:
nameof(ProductSalesPrediction.Prediction), inputColumnName:
nameof(ProductSalesData.numSales), confidence: 95d, changeHistoryLength: docSize
/ 4);
```

3. As you did previously, create the transform from the estimator by adding the following line of code in the `DetectChangePoint()` method: