Anomaly/Outlier Detection

Using Python to Detect Anomalies/Outliers

Aniket Adhikari

2023-11-23

On this page

Anomaly detection, or outlier detection, is a machine learning process of detecting abnormal or rare observations in data that stick out from the rest of the data. Typically, outliers are data points that appear far away from other observations, meaning they differ greatly from the majority of the data

Importance of Anomalies

Improves Model Accuracy

Outliers can adversely effect model predictions if they aren't handled. Models might overfit to outliers and lead to reduced generalization on unseen data. Removing outliers can help to improve model accuracy and prevent misleading models.

Reduce Risks of Bad Decisions

Outlier driven misleading models can result in incorrect data interpretations and decisions. Furthermore, outliers can influence decision-making processes in various domains, from finance to healthcare. Properly handling outliers ensures that decisions are based on accurate and representative information.

Identify Cost-saving Opportunities

Analyzing positive outliers can reveal best practices and improvement opportunities leading to greater efficiencies. Likewise, negative outliers may show problems needing addressing.

Anomaly/Outlier Detection in Practice?

Fraud Detection in Finance → Identifying transactions, claims, or activity that is suspicious and does not conform to regular patterns

- Track credit card fraud
- Detect money laundering
- Detect unusual trading patterns in stock markets

Health Care → Monitoring patient data for anomalies

- Disease detection based on electrocardiogram (ECG) readings
- Uncover irregularities in medical imaging for cancer detection

Cybersecurity → Detecting unsual activity or malicious traffic on computer networks

- Detect network intrusion
- Prevent data exfiltration attempts
- Enhance cybersecurity measures

Manufacturing and Quality Control → Finding defects in manufacturing

- Prevention of defective products from reaching consumers
- Reduce waste
- Maintain product quality

Energy and Utilities Consumption → Finding anomalies in energy consumption patterns

- Identify power grid failures
- Identify abnormal usage
- Determine equipment malfunctions
- Ensure reliable service delivery

Types of Outliers

Visualization

- Use of scatter plots, box plots, histogram distributions
- Allows for visual inspection to find anomalies

Standard Deviation

- Thresholds based on standard deviations from the mean
- Z-score calculates how far away observatoin are from the mean in terms of standard deviatoin

Percentiles

- Declare observations below X percentile or above Y percentile as outliers
- Less sensitive to outliers than standard deviation

Distance-Based

- Mahalanobis, Euclidean, Manhattan distances
- Mark points too far from general cluster

Density-Based

- Find low density regions using algorithms like Local Outlier Factor
- Isolation Forests isolate anomalies

Classification-Based

- Train classification models to label outliers
- Supervised models so need tagged historical data

Time Series Specific

- Treat long term trends, seasonality before applying outlier detection
- Use statistical process control charts

Example of Anomaly Detection in Python

in this example, I'll be using the Pokemon dataset and try to find outliers in total attack and total defense. This can be useful for identifying Pokemon that are exceptionally weak or strong and then trying to trade for them in Pokemon GO. Unlike my prior posts, I would like to show the data that supports my visualizations. This means I'll show how outliers are found when looking at data as well as the visuals.

We start off by importing the necessary libraries, `pandas`, `matplotlib.pyplot`, and `seaborn`.

`pandas` will be used for creating the necessary DataFrames of total attack and total defense. `matplotlib.pyplot` will be used to create the subplots. `seaborn` will be used to specifically create boxplots. Boxplots are a type of diagram that is really helpful in showing how data is distributed.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

I can bring in the data from the Pokemon dataset and create columns that show total attack and total defense.

Total attack will be the sum of attack and special attack. Total defense will be the sum of defense and special defense.

```
data = pd.read_csv('../../datasets/pokemon.csv')
data['Total Attack'] = data['attack'] + data['sp_attack']
data['Total Defense'] = data['defense'] + data['sp_defense']
```

Here I created a shorthand for refering to total attack and total defense.

```
total_attack = data['Total Attack']
total_defense = data['Total Defense']
```

A good way to see how outliers can be found is by looking at the statistics associated with the data, such as:

- the number of data points (`count`)
- the average of the data points (`mean`)
- the standard deviation for the data (`std`)
- the minimum data point (`min`)
- the 25th percentile (`25%`)

- the 50th percentile (`50%` or `median`)
- the 75th percentile (`75%`)
- the maximum data point (`max`)

First off, we can look at the statistics for total attack.

```
1 total_attack.describe()
```

```
/Users/aniketadhikari/anaconda3/envs/homl3/lib/python3.10/site-packages/IPython/core/formatte
  return method()
```

	Total Attack
count	801.000000
mean	149.163546
std	53.357844
min	20.000000
25%	110.000000
50%	145.000000
75%	180.000000
max	360.000000

We can then look at the statistics for total defense

```
1 total_defense.describe()
```

```
/Users/aniketadhikari/anaconda3/envs/homl3/lib/python3.10/site-packages/IPython/core/formatte
  return method()
```

	Total Defense
count	801.000000
mean	143.920100
std	51.308855
min	35.000000
25%	104.000000
50%	140.000000
75%	178.000000
max	460.000000

Below is how we determine outliers in boxplots.

We start off by getting the upper and lower level quartiles, which are the 75th and 25th percentiles of the data, respectively.

`attack_q1`, or the 25th percentile, is 110.0 and `attack_q3`, or the 75th percentile, is 180.0.

We also have to get the median and interquartile range. interquartile range, or IQR, is determined by subtracting the 25th percentile from the 75th percentile. The difference would be $180 - 110 = 70$.

After that, IQR is used to form the "whiskers" of the box plot. The lower whisker is calculated by subtracting IQR $*$ 1.5 from the lower quartile. This is $110 - 70 * 1.5 = 5$. The upper whisker is calculated by adding IQR $*$ 1.5 to the upper quartiler. This is $180 + 70 * 1.5 = 285$

```
attack_q1 = total_attack.quantile(.25)
attack_q3 = total_attack.quantile(.75)

attack_median = total_attack.median()

attack_iqr = attack_q3 - attack_q1

attack_min_whisker = attack_q1 - 1.5 * attack_iqr
attack_max_whisker = attack_q3 + 1.5 * attack_iqr
```

After getting the calculations, we can create a DataFrame that is based on some conditions.

We filter the data so that we get any points where `total_attack` is greater than the upper whisker. In addition, we want to get any data points where `total_attack` is less than the lower whisker.

As a result of these conditional filters, we get 12 different Pokemon which satisfy these conditions. This means 12 Pokemon are identified as anomalies.

```
attack_outliers = data[(total_attack > attack_max_whisker) | (total_attack < a
attack_outliers[['name', 'Total Attack']]
```

```
/Users/aniketadhikari/anaconda3/envs/homl3/lib/python3.10/site-packages/IPython/core/formatt
  return method()
```

	name	Total Attack
256	Blaziken	290
380	Latios	290
444	Garchomp	290
645	Kyurem	290
657	Greninja	298
680	Aegislash	300
718	Diancie	320
381	Kyogre	330
382	Groudon	330
719	Hoopa	330
149	Mewtwo	344
383	Rayquaza	360

We can do the same thing for total defense to get outliers. From this, we get 10 Pokemon that are identified as anomalies.

```
defense_q1 = total_defense.quantile(.25)
defense_q3 = total_defense.quantile(.75)

defense_median = total_defense.median()

defense_iqr = defense_q3 - defense_q1

defense_min_whisker = defense_q1 - 1.5 * defense_iqr
defense_max_whisker = defense_q3 + 1.5 * defense_iqr

defense_outliers = data[(total_defense > defense_max_whisker) | (total_defense < defense_min
defense_outliers[['name', 'Total Defense']]
```

```
/Users/aniketadhikari/anaconda3/envs/homl3/lib/python3.10/site-packages/IPython/core/formatte
  return method()
```

	name	Total Defense
747	Toxapex	294
475	Probopass	295
376	Regirock	300
377	Regice	300
378	Registeel	300
702	Carbink	300
410	Bastiodon	306
305	Aggron	310
207	Steelix	325
212	Shuckle	460

Finally, I wanted to visualize both these sets of data. The top box diagram represents the total attack. The outliers are shown and appear as if there are only 7 outliers when there are actually 12. This is simply because there is some overlap between some data points.

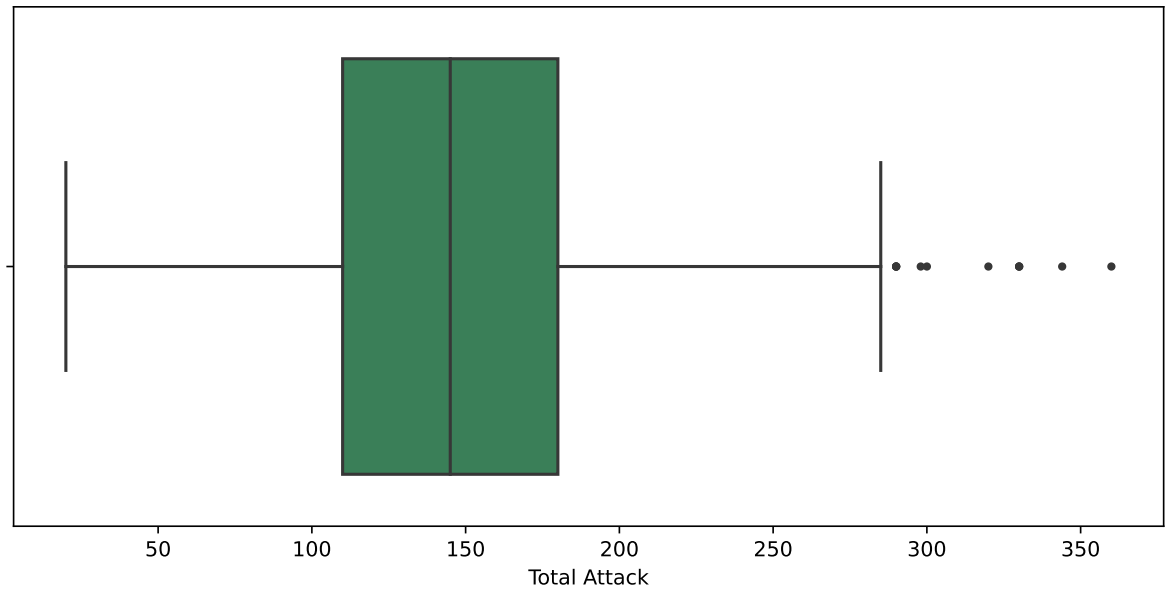
The bottom box diagram represents the total defense and has the same issue with the data points overlapping.

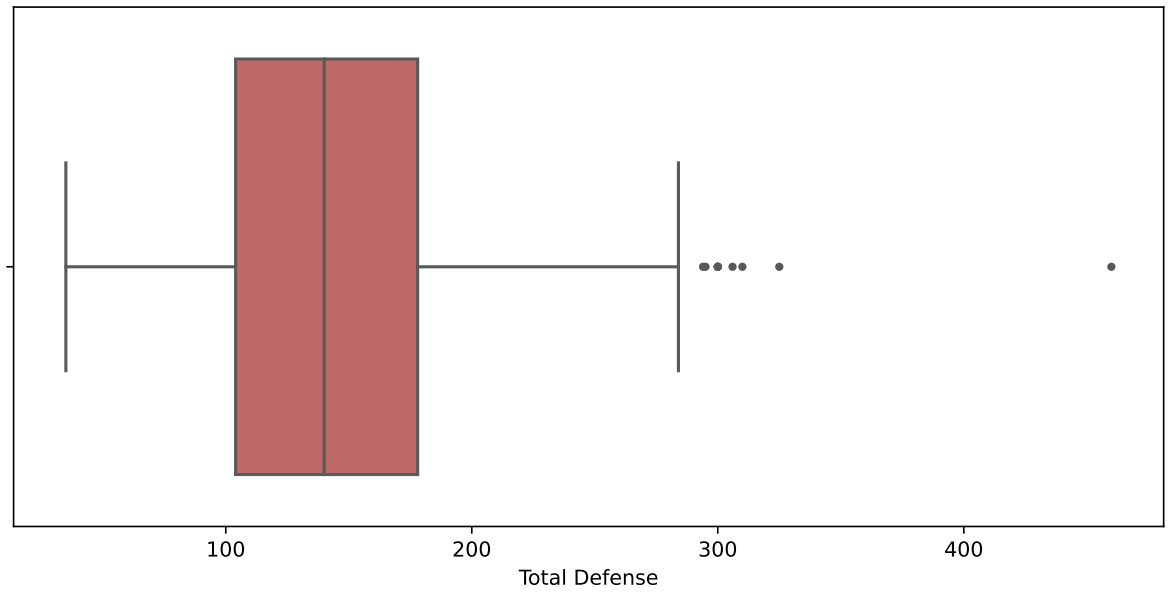
This is an important lesson in not depending solely on visualizations and supplementing it with the actual data.

```
fig, axs = plt.subplots(2, 1, figsize=(10, 10))

sns.boxplot(x=total_attack, ax=axs[0], fliersize=3, flierprops={"marker": "o"}, color="seagre
sns.boxplot(x=total_defense, ax=axs[1], fliersize=3, flierprops={"marker": "o"}, color="india
```

```
<AxesSubplot:xlabel='Total Defense'>
```


50
100
150
200
250
300
350
Total Attack


100
200
300
400
Total Defense