<https://github.com/Deffro/Data-Science-Portfolio>

<https://towardsdatascience.com/outlier-detection-theory-visualizations-and-code-a4fd39de540c>

Outlier Detection — Theory, Visualizations, and Code

# What is Outlier Detection?

Outlier Detection is also known as anomaly detection, noise detection, deviation detection, or exception mining. There is no universally accepted definition. An early definition by (Grubbs, 1969) is: An outlying observation, or outlier, is one that appears to deviate markedly from other members of the sample in which it occurs. A more recent definition by (Barnett and Lewis, 1994) is:

*An observation which appears to be inconsistent with the remainder of that set of data.*

# Causes

Straight from [this](https://towardsdatascience.com/a-brief-overview-of-outlier-detection-techniques-1e0b2c19e561) excellent article, the most common causes of outliers are:

* **Human errors** — Data entry errors
* **Instrument errors** — Measurement errors
* **Experimental errors** — data extraction or experiment planning/executing errors
* **Intentional** — dummy outliers made to test detection methods
* **Data processing errors** — data manipulation or data set unintended mutations
* **Sampling errors** — extracting or mixing data from wrong or various sources
* **Natural** — not an error, novelties in data

# Applications

A list of applications that utilize outlier detection according to (Hodge, V.J. and Austin, J., 2014) is:

* **Fraud detection** — detecting fraudulent applications for credit cards,  
  state benefits or detecting fraudulent usage of credit cards or mobile phones.
* **Loan application processing** — to detect fraudulent applications or  
  potentially problematic customers.
* **Intrusion detection** — detecting unauthorized access in computer networks.
* **Activity monitoring** — detecting mobile phone fraud by monitoring  
  phone activity or suspicious trades in the equity markets.
* **Network performance** — monitoring the performance of computer  
  networks, for example, to detect network bottlenecks.
* **Fault diagnosis**— monitoring processes to detect faults in motors,  
  generators, pipelines, or space instruments on space shuttles for  
  example.
* **Structural defect detection** — monitoring manufacturing lines to  
  detect faulty production runs for example cracked beams.
* **Satellite image analysis** — identifying novel features or misclassified  
  features.
* **Detecting novelties in images**— for robot or surveillance  
  systems.
* **Motion segmentation** — detecting image features moving independently of the background.
* **Time-series monitoring** — monitoring safety-critical applications  
  such as drilling or high-speed milling.
* **Medical condition monitoring**— such as heart-rate monitors.
* **Pharmaceutical research** — identifying novel molecular structures.
* **Detecting novelty in the text**— to detect the onset of news stories, for  
  topic detection and tracking or for traders to pinpoint equity, commodities, FX trading stories, outperforming or underperforming  
  commodities.
* **Detecting unexpected entries** in databases — for data mining to  
  detect errors, frauds, or valid but unexpected entries.
* **Detecting mislabelled data** in a training data set.

# Approaches

There are 3 outlier detection approaches:

1. Determine the outliers with no prior knowledge of the data. This is analogous to **unsupervised**clustering.  
2. Model both normality and abnormality. This is analogous to **supervised**classification and need labeled data.  
3. Model only normality. This is called novelty detection and is analogous to **semi-supervised** recognition. It needs labeled data that belong to the normal class.

I will deal with the **first approach**. It is the most common case. Most datasets don’t have labeled data concerning outliers.

# Taxonomy

According to Ben-Gal I., (2005) outlier detection methods can be divided between **univariate**methods and **multivariate**methods. Another fundamental taxonomy of outlier detection methods is between **parametric**(statistical) methods, which assume a known underlying distribution of the observations, and **non-parametric** methods that are model-free like distance-based methods and clustering techniques.

# Dataset

I will use the [Pokemon](https://www.kaggle.com/abcsds/pokemon)dataset and perform Outlier Detection on 2 columns **[‘HP’, ‘Speed’]** It is a fun dataset to play with, it has few observations and the computations will be very fast and a lot of people are familiar with it. The choice of only two columns has been taken for **visualization**purposes only (two dimensions). To observe the results in a scatter plot. The methods can be scaled in **multiple dimensions**.

# Outlier Detection Algorithms

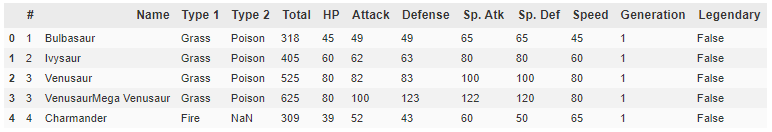
The algorithms that will follow are:

* **Isolation Forest**
* **Extended Isolation Forest**
* **Local Outlier Factor**
* **DBSCAN**
* **One Class SVM**
* **The ensemble**of the above

# Code and Visualizations

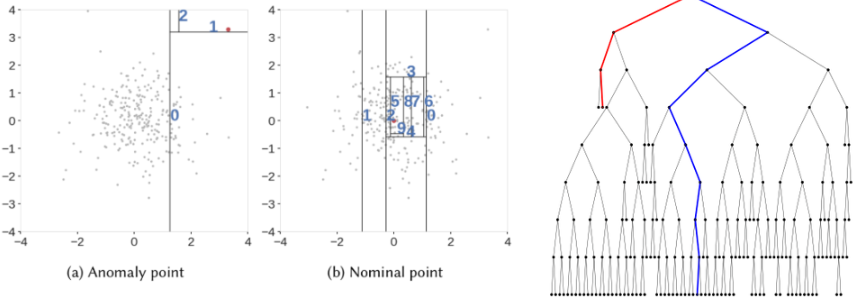
The code for this article is on my [GitHub](https://github.com/Deffro/Data-Science-Portfolio/blob/master/Notebooks/Outlier Detection/Outlier Detection - Theory%2C Visualizations and Code.ipynb). I will not include code in the article to keep it compact and shorter.

After reading the data, the first five rows are like this:



# Isolation Forest

* Isolation Forest, like any tree ensemble method, is built on the basis of decision trees. In these trees, partitions are created by first randomly selecting a feature and then selecting a random split value between the minimum and maximum value of the selected feature.
* In order to create a branch in the tree, first, a random feature is selected. Afterward, a random split value (between min and max value) is chosen for that feature. If the given observation has a lower value of this feature then the one selected it follows the left branch, otherwise the right one. This process is continued until a single point is isolated or specified maximum depth is reached.
* In principle, outliers are less frequent than regular observations and are different from them in terms of values (they lie further away from the regular observations in the feature space). That is why by using such random partitioning they should be identified closer to the root of the tree (shorter average path length, i.e., the number of edges an observation must pass in the tree going from the root to the terminal node), with fewer splits necessary.



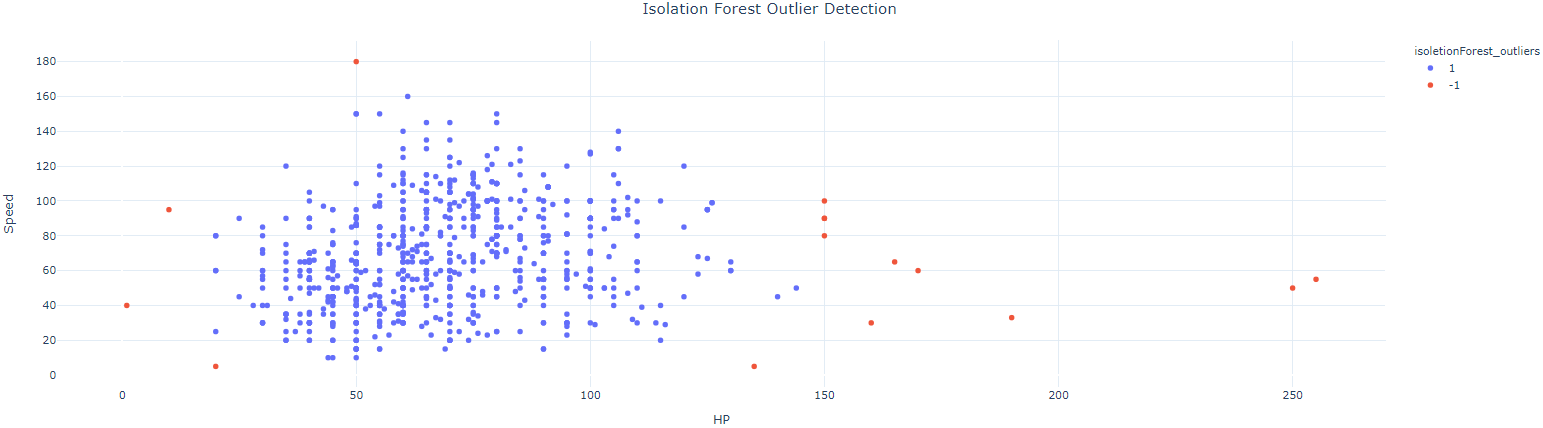
More on Isolation Forest:

* [Isolation Forest — Paper](https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/icdm08b.pdf?q=isolation-forest)
* [Outlier Detection with Isolation Forest](https://towardsdatascience.com/outlier-detection-with-isolation-forest-3d190448d45e)

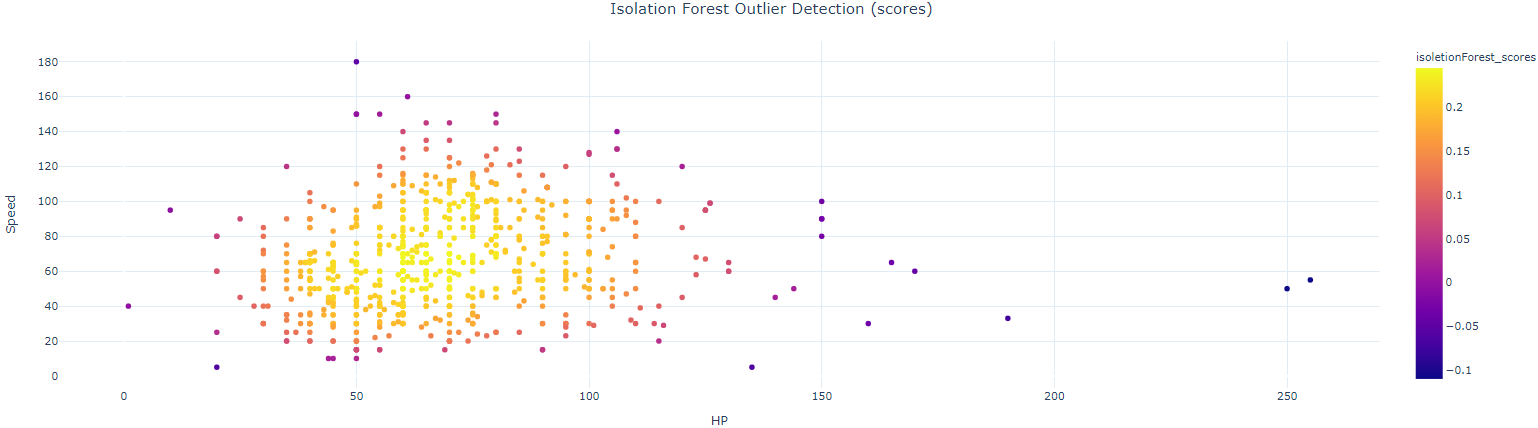
I will use IsolationForest from the sklearn library. When defining the algorithm there is an important parameter called contamination. It is the percentage of observations that the algorithm will expect as outliers. I set it equal to **2%**. We fit the X (2 features HP and Speed) to the algorithm and use fit\_predict to use it also on X. This produces plain outliers (-1 is outlier, 1 is inlier). We can also use the function decision\_function to get the scores Isolation Forest gave to each sample.

After running the algorithm it found 785 inliers and **15 outliers**.

Let’s plot the results.

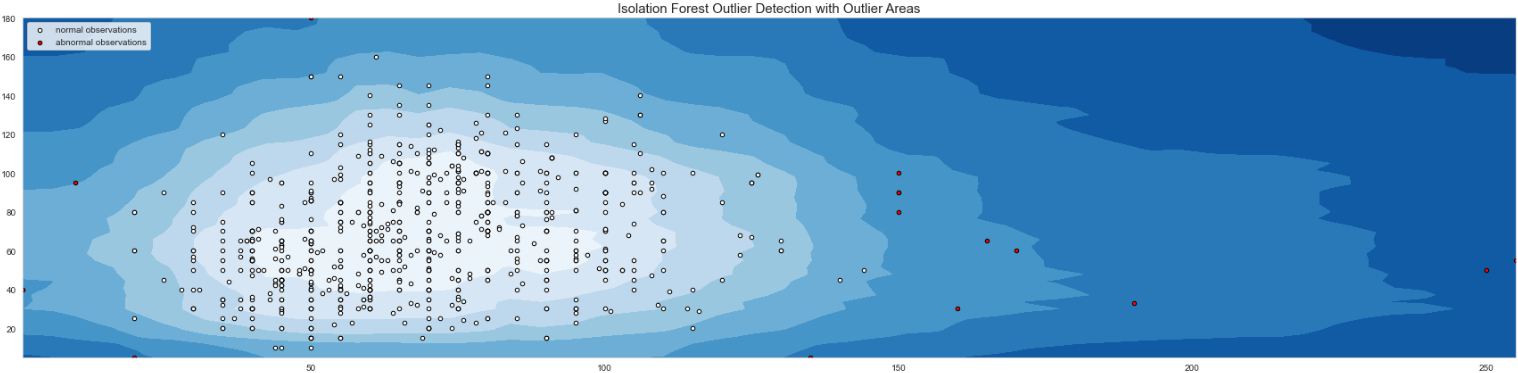


Or we can plot the pure scores rather than just outlier/inlier.



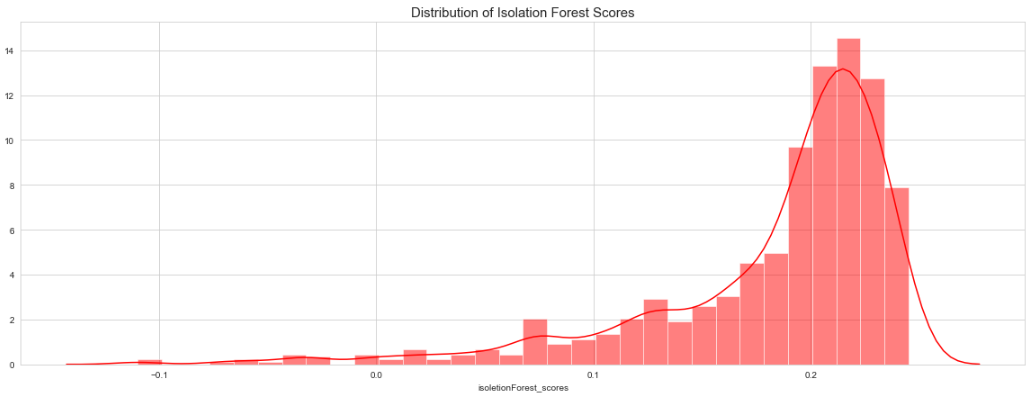
Visually the 15 outliers seem legit and outside of the main blob of data points.

We can make a more advanced visualization which except the inliers and outliers displays the decision boundaries of Isolation Forest.

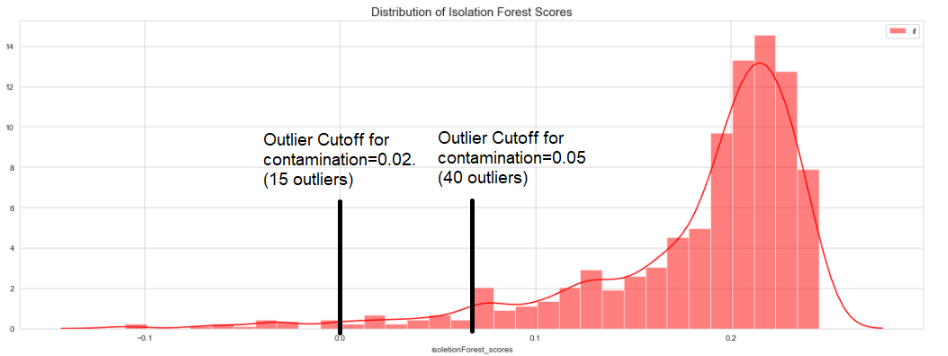


The darker the color, the more outlier is the region.

Finally, we can see the distribution of the scores.



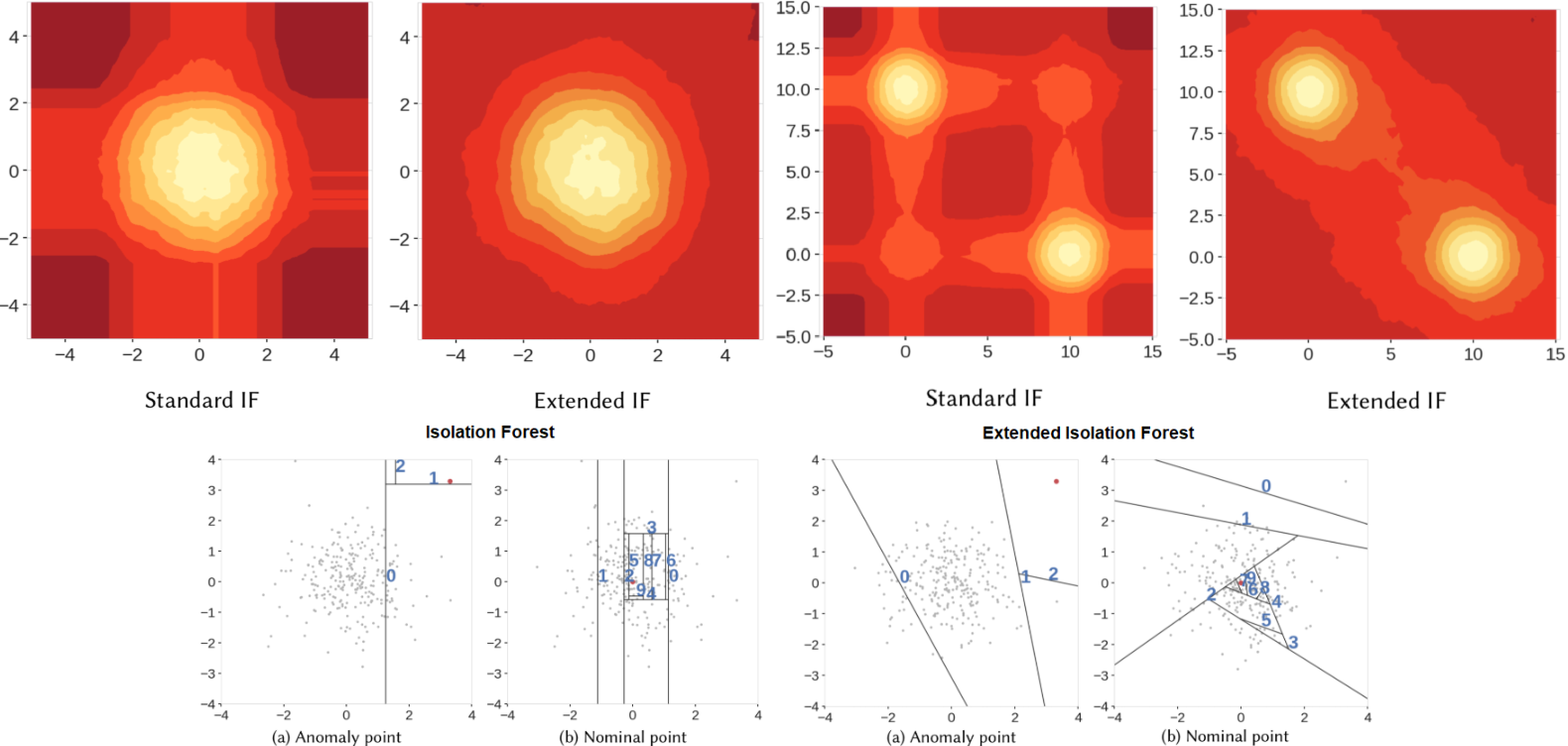
The distribution is important and helps us to better identify the correct contamination value for our case. If we change the contamination value, the isoletionForest\_scores will change, but the distribution will stay the same. The algorithm will adjust the cutoff for outliers in the distribution plot.



# Extended Isolation Forest

Isolation Forest has a drawback: Its decision boundaries are either vertical or horizontal. As the lines can only be parallel to the axes, there are regions that contain many branch cuts and only a few or single observations, which results in improper anomaly scores for some of the observations.

Extended Isolation Forest selects 1) a random slope for the branch cut and 2) a random intercept chosen from the range of available values from the training data. These terms are the linear regression line actually.



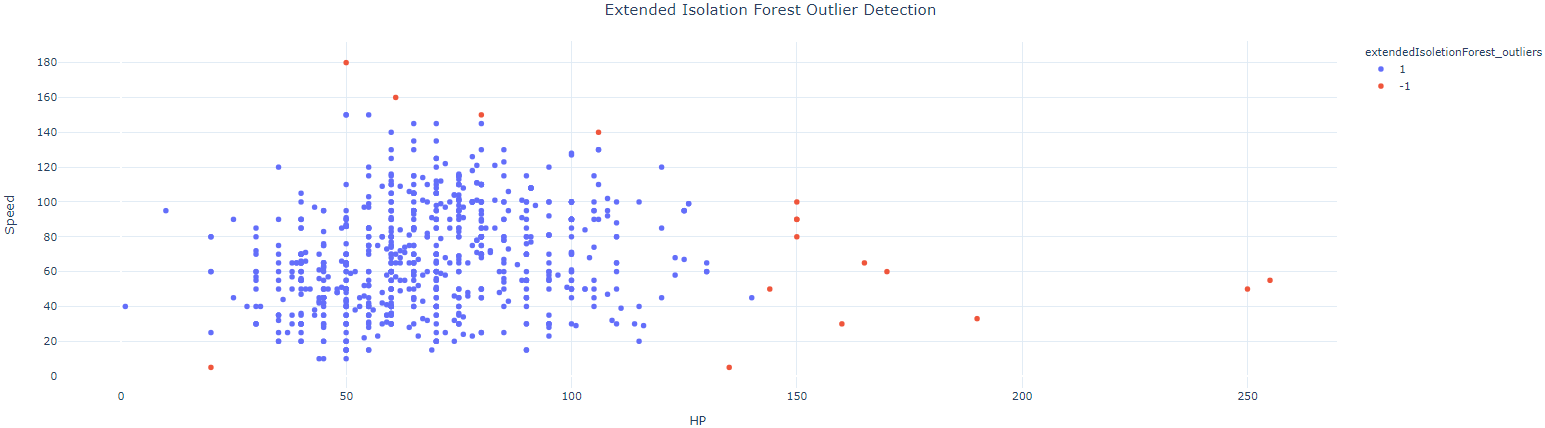
More on Extended Isolation Forest:

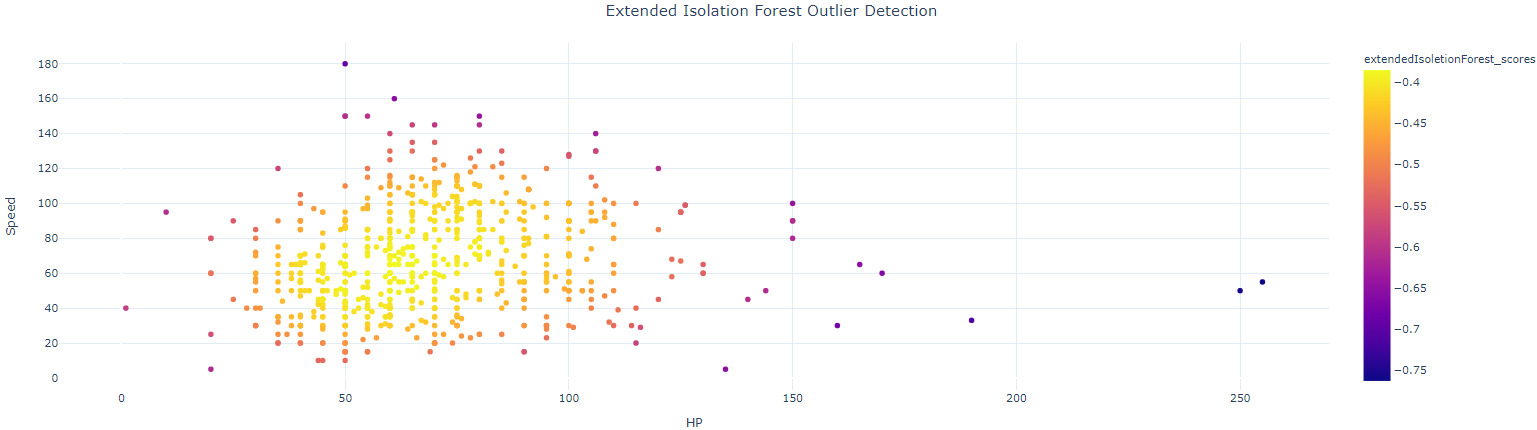
- [Extended Isolation Forest — Paper](https://arxiv.org/abs/1811.02141)  
- [Extended Isolation Forest — Github](https://github.com/sahandha/eif)  
- [Outlier Detection with Extended Isolation Forest](https://towardsdatascience.com/outlier-detection-with-extended-isolation-forest-1e248a3fe97b)

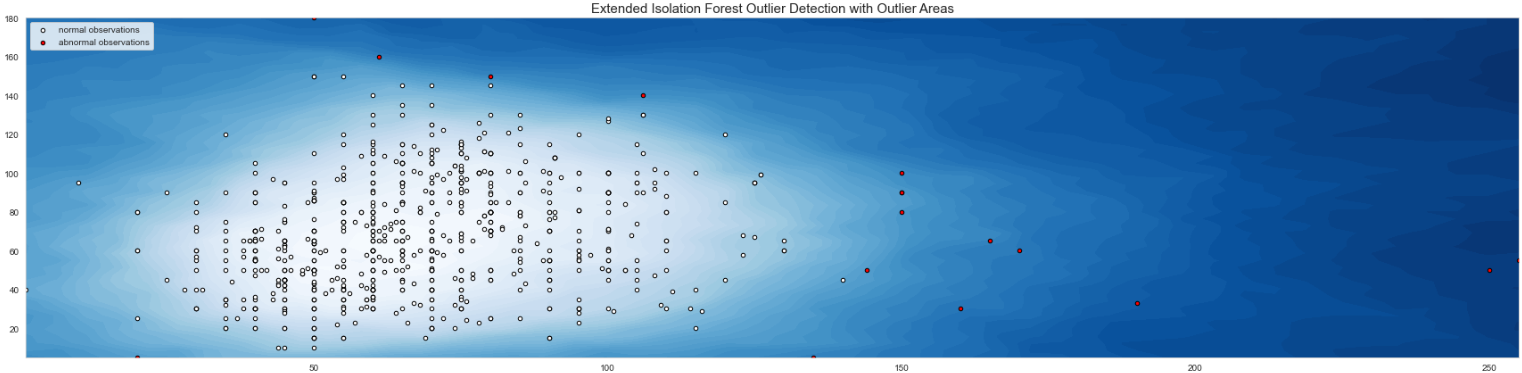
Extended Isolation Forest is not implemented in sklearn, but it is available on [Github](https://github.com/sahandha/eif)

We have to multiply its scores with -1 to be in the same form as the scores of the other algorithms.

Extended Isolation Forest does not provide plain outliers and inliers (as -1 and 1). We simply created them by taking the lowest 2% of the scores as outliers. The scores of this algorithm are different from the basic Isolation Forest. All scores here are negative.

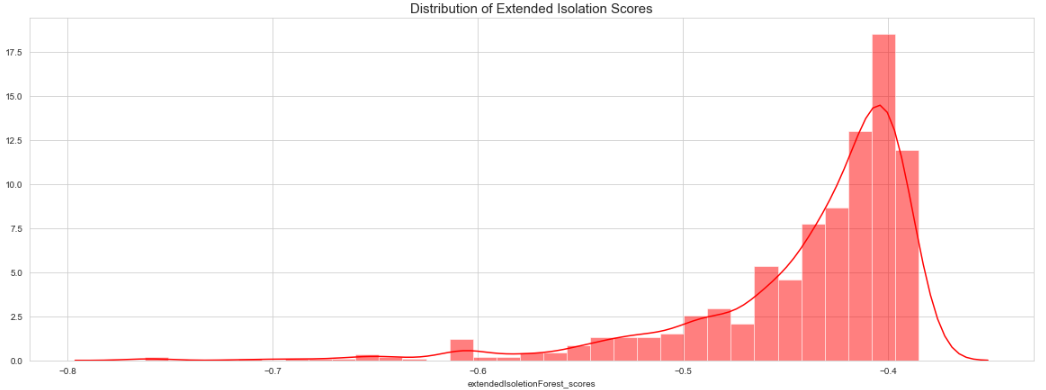






If you check the code you may notice that I used cmap=plt.cm.Blues in this plot instead of cmap=plt.cm.Blues\_r (reverse) of the previous. We can see how much smoother extended isolation forest is in the transitions between the different outlier regions.

The algorithm found 16 outliers.



# Local Outlier Factor

* The LOF is a calculation that looks at the neighbors of a certain point to find out its density and compare this to the density of other points later on.
* The LOF of a point tells the density of this point compared to the density of its neighbors. If the density of a point is much smaller than the densities of its neighbors (LOF ≫1), the point is far from dense areas and, hence, an outlier.
* This is useful because not all methods will not identify a point that’s an outlier relative to a nearby cluster of points (a local outlier) if that whole region is not an outlying region in the global space of data points.

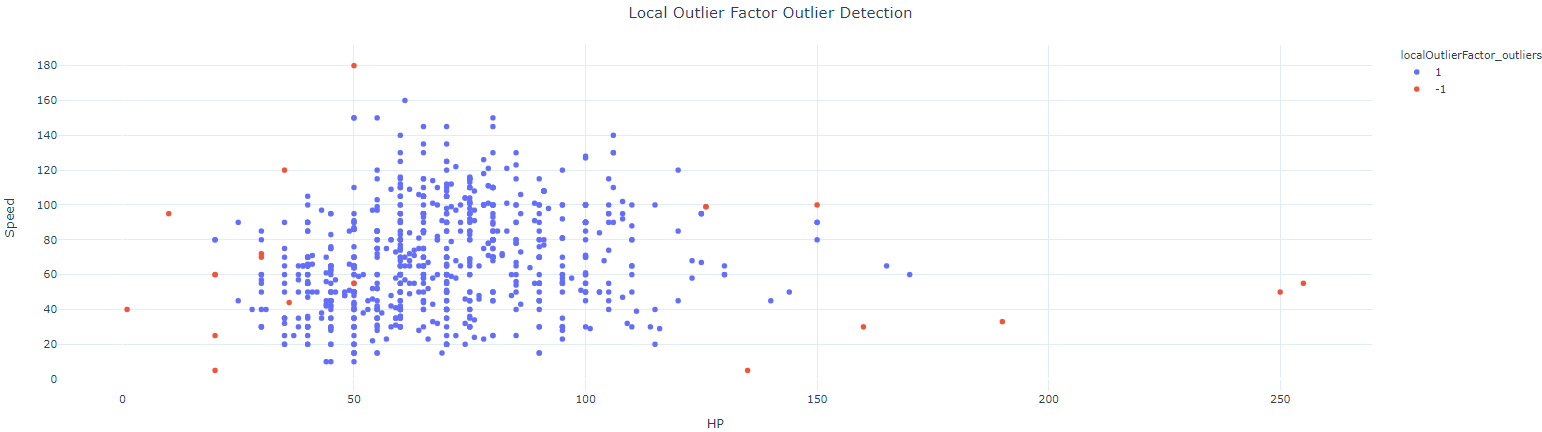
The LOF for a point P will have a:

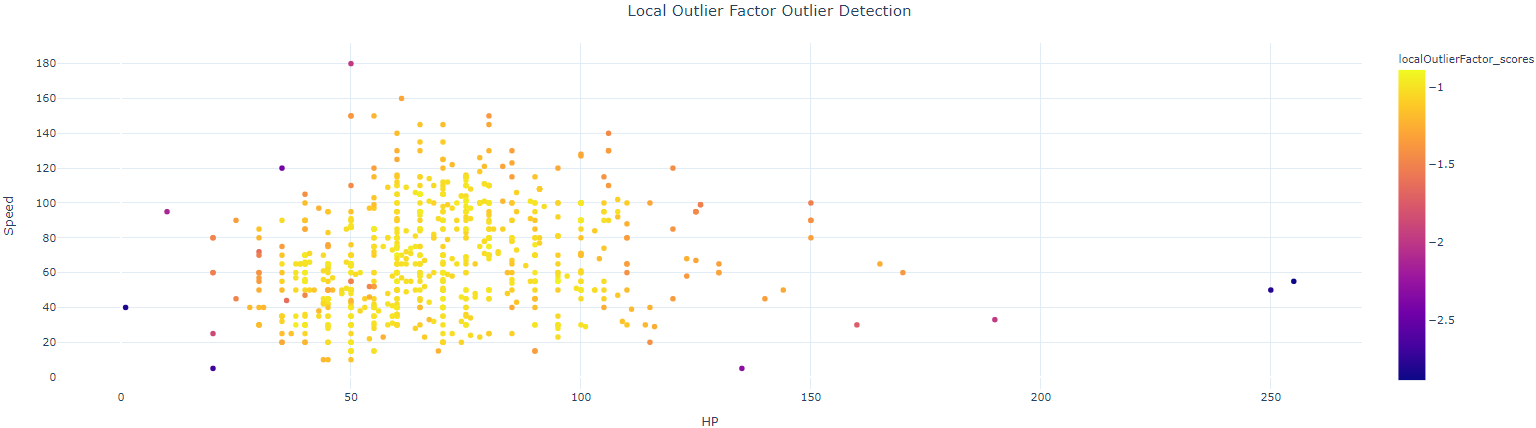
* High value if → P is far from its neighbors and its neighbors have high densities (are close to their neighbors) (LOF = (high distance sum) x (high density sum) = High value)
* Less high value if -> P is far from its neighbors, but its neighbors have low densities (LOF = (high sum) x (low sum) = middle value)
* Less high value if -> P is close to its neighbors and its neighbors have low densities (LOF = (low sum) x (low sum) = low value )

More on Local Outlier Factor:

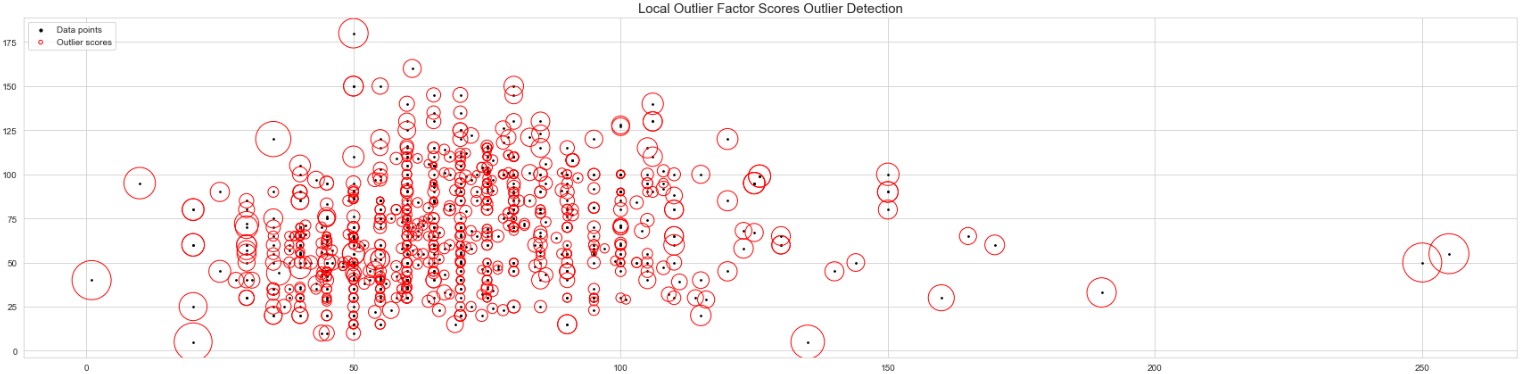
* [LoOP: Local Outlier Probabilities — Paper](https://dl.acm.org/doi/pdf/10.1145/1645953.1646195)
* [Local Outlier Factor for Anomaly Detection](https://towardsdatascience.com/local-outlier-factor-for-anomaly-detection-cc0c770d2ebe)

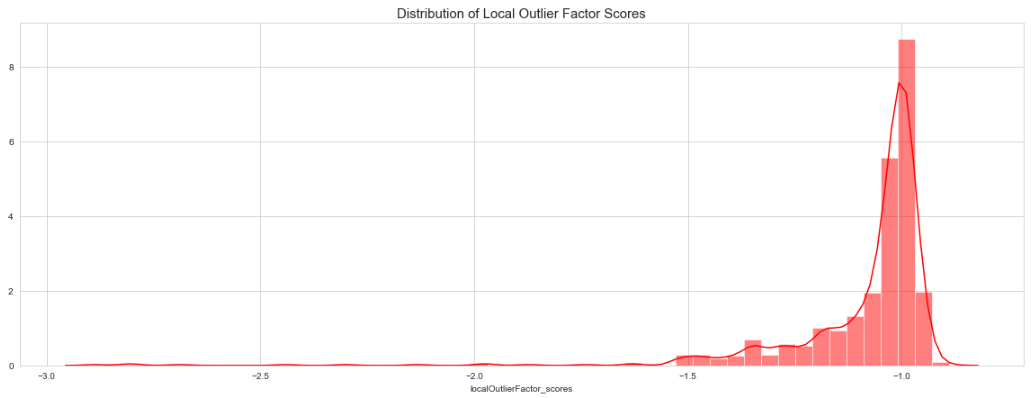
After running the code from the sklearn library, it determines 21 local outliers.





We can create another interesting plot, where the bigger the local outlier the bigger the circle around it.





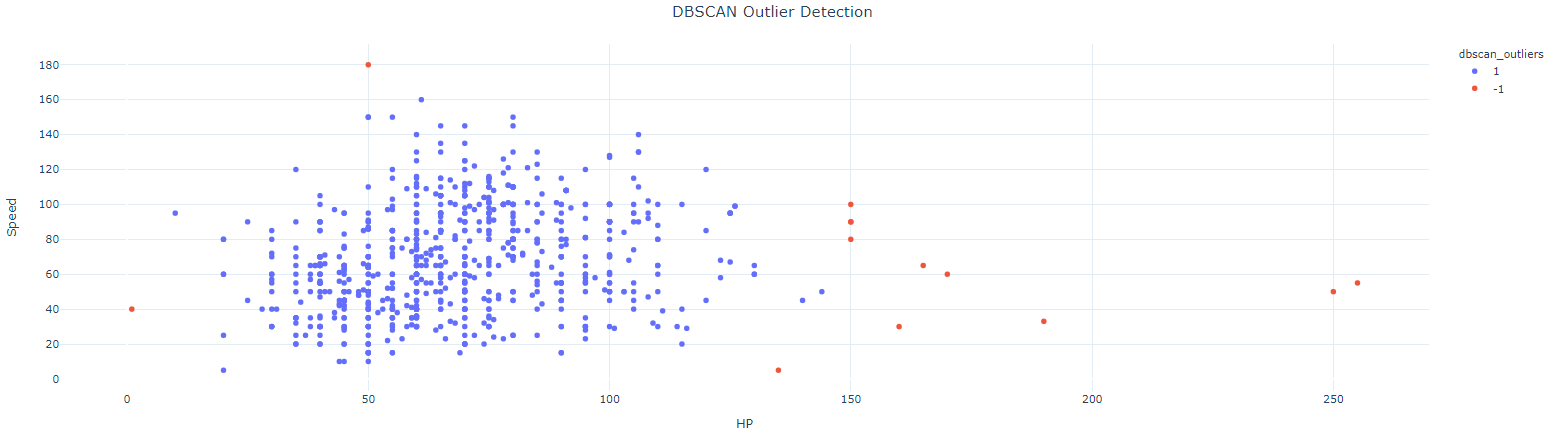
This algorithm is much different than the previous ones. It also finds outliers, but in a different manner. It finds local outliers. Did you notice that there are outliers inside the main mass of the plot?

# DBSCAN

A classic clustering algorithm that works as follows:

* Randomly select a point not already assigned to a cluster or designated as an outlier. Determine if it’s a core point by seeing if there are at least min\_samples points around it within epsilon distance.
* Create a cluster of this core point and all points within epsilon distance of it (all directly reachable points).
* Find all points that are within epsilon distance of each point in the cluster and add them to the cluster. Find all points that are within epsilon distance of all newly added points and add these to the cluster. Rinse and repeat. (i.e. perform “neighborhood jumps” to find all density-reachable points and add them to the cluster).

For our sample, it found 13 outliers after tuning the epsilon parameter.



The algorithm does not provide scores for outlier strength.

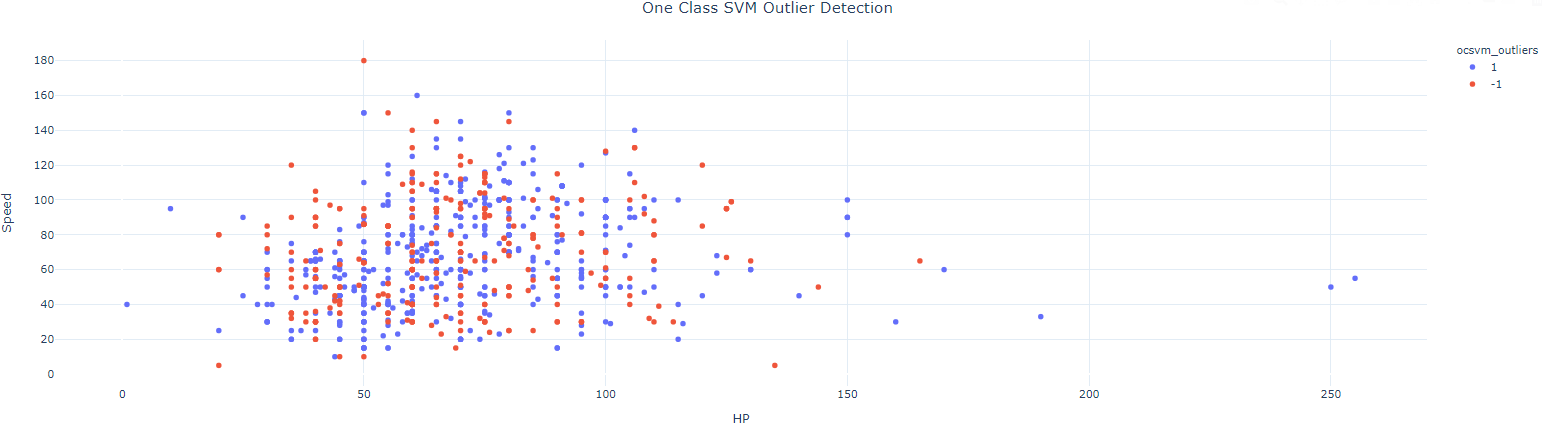
# One-Class SVM

* A one-class classifier is fit on a training dataset that only has examples from the normal class, but it can also be used for all data. Once prepared, the model is used to classify new examples as either normal or not-normal.
* The main difference from a standard SVM is that it is fit in an unsupervised manner and does not provide the normal hyperparameters for tuning the margin like C. Instead, it provides a hyperparameter “nu” that controls the sensitivity of the support vectors and should be tuned to the approximate ratio of outliers in the data.

More on One-Class SVM

* [Outlier Detection with One-Class SVMs](https://towardsdatascience.com/outlier-detection-with-one-class-svms-5403a1a1878c)
* [One-Class Classification Algorithms for Imbalanced Datasets](https://machinelearningmastery.com/one-class-classification-algorithms/)

After running the algorithm I get the following scatter plot



It doesn’t seem to work in this data. I couldn't find a better nu. For other nu values, the outliers were more than the inliers. If someone has any idea please share and I will update!

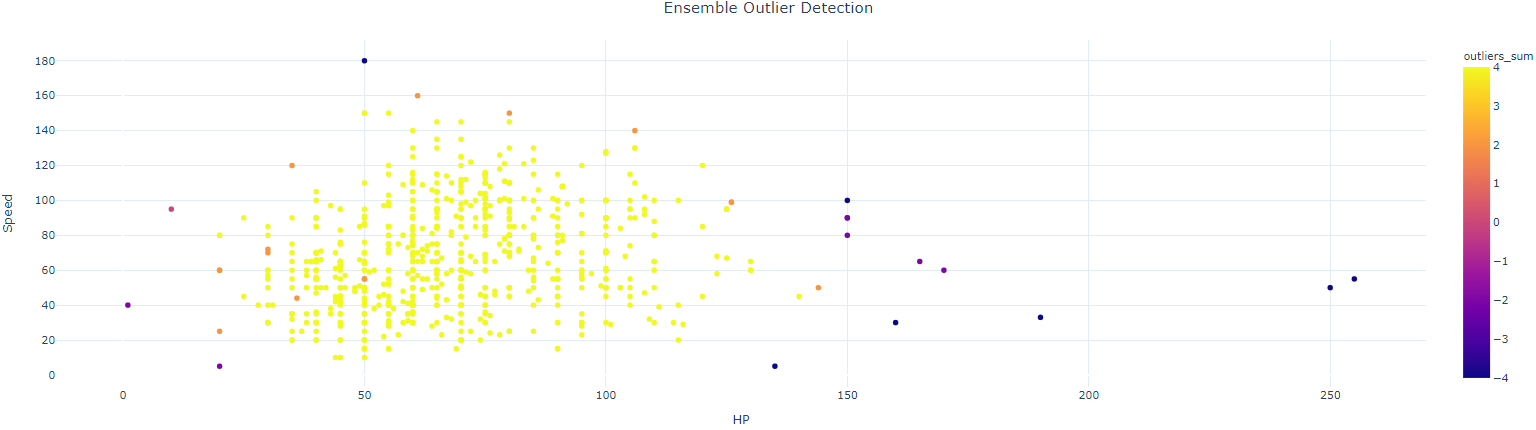
# Ensemble

Finally, let’s combine the 5 algorithms to make a robust one. I will simply add the outlier columns which are either -1 for outlier and 1 for inlier.

I will not use One-Class SVM.

After adding together the results we get:

data['outliers\_sum'].value\_counts()value count   
 4 770  
 2 15  
 -4 7  
 -2 7  
 0 1



Observations with outliers\_sum=4, mean than all 4 algorithms agreed that it is an inlier, while for complete outlier agreement the sum is -4.

Let’s first see for which 7 pokemon all algorithms agree for outliers. We can also keep as inliers the observations where sum=4 and the rest as outliers. It is up to us.

data.loc[data[‘outliers\_sum’]==-4][‘Name’]121 Chansey  
155 Snorlax  
217 Wobbuffet  
261 Blissey  
313 Slaking  
431 DeoxysSpeed Forme  
495 Munchlax

Individual images were taken from [pokemondb](https://pokemondb.net/).

These are our outliers on HP and Speed combined!

Thanks for reading!