Outlier Detection

Unfortunately, detecting is more of an art than a science. I'd like to offer the reader a framework to work with.

What is an outlier?
There are 2 popular approaches to defying an outlier:

- In the typical scenario, the distribution of the variable is Gaussian and thus outliers will lie outside the mean plus or minus 3 times the standard deviation (99%) of the variable.

- If the variable is not normally distributed, a general approach is then the interquartile range (IQR) * 1.5.

I personally favor Tukey's fences (developer of the Box Plot). He proposes as 1.5 as a "outlier" and 3 data is "far out". The upper fence is 0.75 + (IQR * 1.5).

How to calculate IQR?
Let's say you have the variables 1,2,3,4,5,6,7,8,9 and 10.

So IQR is 5 (7.5-2.5) which results in 7.5 + (5*1.5) = 15 as an outlier.

Let's say I have a column with numerical data 'Pre_Test_Score'. Now I want to find out with the help of Panda, what are the max() and min() values:

```
In [207]: # Max value
          data['Pre_Test_Score'].max()

Out[207]: 121.0

In [208]: # Min value
          data['Pre_Test_Score'].min()

Out[208]: 92.0
```

Let's calculate IQR * 1.5 for this column:

```
In [209]: # Calculating outlier values according to Tunkey's fence
          IQR_2 = data.Pre_Test_Score.quantile(0.75) - data.Pre_Test_Score.quantile(0.25)

          Lower_fence_2 = data.Pre_Test_Score.quantile(0.25) - (IQR_2 * 1.5)
          Upper_fence_2 = data.Pre_Test_Score.quantile(0.75) + (IQR_2 * 1.5)

In [210]: Upper_fence_2, Lower_fence_2, IQR_2

Out[210]: (108.5, 88.5, 5.0)
```

Using a conditional (>108.50) and a Lambda function, we want to count the number outliers we have.

```
In [211]: # How many outliers do we have?
          data[data['Pre_Test_Score'] > 108.50].apply(lambda x: x.count())

Out[211]: Name              1
          Age               1
          Gender            1
          Pre_Test_Score    1
          Post_Test_Score   1
          Country           1
          State             1
          dtype: int64
```

More than 5% of outliers?
As a rule of thumb, if outliers are <5% of the dataset, we might ignore them. If they are >5%, they are probably not outliers anymore but part of the nature of the dataset.

Typically, we have three strategies we can use to handle outliers:
1 – We drop them.
2 – We mark them as outliers and include it as a feature.
3 – We can np.log(X) transform the feature do dampen the effect of the outlier.

In our example above, we had only one outlier, but let's assume we had >5% of outliers. In this case, we could create a new column called 'outliers'. If a row had in the column 'Pre_Test_Score' had a value of >108.50, we mark this row as a 1. Otherwise 0.

In Python, this looks like that:

```
In [212]: data['outliers'] = np.where(data['Pre_Test_Score'] >= 108.50, 1,0)

In [213]: data.head(4)
```

Out[213]:

	Name	Age	Gender	Pre_Test_Score	Post_Test_Score	Country	State	outliers
0	Jason	42	Male	101.0	103	USA	CA	0
1	Molly	52	Female	NaN	191	USA	MI	0
2	Tina	35	Female	121.0	115	USA	NY	1
3	Jake	24	Male	92.0	112	USA	OR	0

You send that Python added at the end a new column 'outliers' where each row with a 'Pre_Test_Score' of >108.50 is labeled with a 1.

The purpose of this is that make it easier for a classification algorithm such as Logistic Regression to 'learn' patterns in the dataset. And the easier it is we make it for the algorithm, the better the algorithm will work. Better means that the algorithm will have an improved accuracy in classifying.

After our outlier detection work, we often need to normalize or standardize our data. One problem is, if we have significant outliers, normalizing our data negatively impact our data.

No matter if we use normalization or standardization, we squeeze the outliers into it and thus might make them more prominent.

In general, this should be of a less problem if we mark the outliers as stated above.

Last but not least, ideally we should favor standardization if our dataset has a normal distribution. If not, min-max scaling might be preferred.

Missing Values: How to deal with MCAR + MAR at the same time?

With missing numerical values, there's no straightforward answer to this question, and which method to use on which occasion is not set in stone.

Different methods make different assumptions and have different advantages and disadvantages.

But, as I'm strongly opinionated I suggest the follow practitioner approach:

Rule of Thumb #1:
If the missing values are <5% of the whole dataset, we just impute with the mean or median. The mean if the distribution is Gaussian and the median if the distribution is not a normal distribution.

Rule of Thumb #2:
If the missing values account for more than 5% of the dataset, we can apply a popular competition trick. But first, we need a quick statistics refresher:

MCAR (missing completely at random): entirely random. However, data are rarely MCAR.

MAR (missing at random): impossible to verify statistically. Occurs when the missingness is not random, but can be fully accounted for by variables where there is complete information. E.g. males are less likely to fill in a depression survey but it has nothing to do with their level of depression.

MNAR (missing not at random): e.g. men failed to fill a depression survey because of their level of depression.

Competition Trick:

If NAN is >5% of the dataset, interpolate NAN's with mean (if data distribution is Gaussian) or median (if non-normal distribution).

Additionally add a new column labeling NAN's with 1, else 0. Thus covering 2 angles:

(1) If the data was missing completely at random, this would be contemplated by the mean imputation, and

(2) if it wasn't this would be captured by the additional variable.

First, I replace NAN's with the mean of 100.71:

```
In [72]: # What's the mean of Pre_Test_Score?
         data['Pre_Test_Score'].mean()

Out[72]: 100.71375

In [74]: data['Pre_Test_Score'].fillna(value=100.71, inplace=True)
```

Second, I can create a new column "NAN_Labels" where we label 1 if the value is exact 100.71

```
In [75]: data['NAN_Labels'] = np.where(data['Pre_Test_Score'] == 100.71, 1,0)

In [76]: data.head()
```

Out[76]:

	Name	Age	Gender	Pre_Test_Score	Post_Test_Score	Country	State	Label	outliers	NAN_Labels
0	Jason	42	Male	101.00	103	USA	CA	0	0	0
1	Molly	52	Female	100.71	191	USA	MI	0	0	1
2	Tina	35	Female	121.00	115	USA	NY	1	1	0
3	Jake	24	Male	92.00	112	USA	OR	0	0	0
4	NaN	22	Male	98.00	134	USA	IL	0	0	0

You see that in row 1, we now have a NAN_Labels of 1. This is exactly what we wanted. Now we give the Machine Learning algorithm the chance of detecting patterns in NAN's.

In an academic environment, one might do Hypothesis test if the MCAR might be not random.