CHURN

How to increase customer loyalty with Tableau and Machine Learning.

Franco Arda

What does a Gaussian look like?

To start with, we want to know what a Gaussian (normal) distribution looks like. With randn() from NumPy, we can generate a sample of random numbers drawn from a Gaussian distribution. Then randn() function will genera

The randn() function will generate a specific number of random numbers such as 10,000 drawn from a Gaussian distribution with a mean of zero (or 100 as I defined it) and a standard deviation of 1.

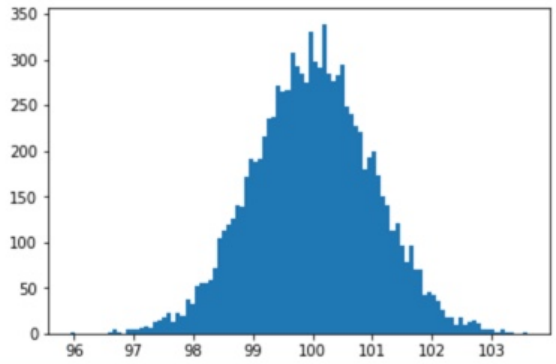
```
In [12]: # Generate a sample of random Gaussians
         from numpy.random import randn
         import matplotlib.pyplot as plt

In [33]: # Generate univariate obervations
         data = randn(10000) + 100
```

Visualizing the distribution with Matplotlib would generate a perfect Gaussian, but the numbers were randomly chosen as a I limited the example to 10,000 observations.

```
In [34]: # Historgram
         plt.hist(data, bins=100)
         plt.show

Out[34]: <function matplotlib.pyplot.show>
```



We can see something important: while the distribution with only 10,000 observations has already a nicely Bell-shaped distribution around the mean of 100, we can always expect some noise or limitations around our data sample.

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.