Data Science Essentials

Data Cleansing

Usually, data requires some cleansing before it can be useful in creating predictive models.

Missing and Repeated Values

It is common for data to have some values missing or to have some repeated rows of data. For example, consider the following data, which represents cupcake sales:

Date	**Chocolate Sales**	**Vanilla Sales**
01-01-2016	125	76
01-02-2016	122	0
	135	85
01-04-2016		76
01-05-2016	127	74
01-02-2016	122	81

Missing Values

In this dataset, there are some rows that contain empty columns. The values for these data points are missing, or unknown. In some cases, the missing values can be interpolated from the other available data. For example, the missing date falls within a sequence of incrementing dates, and it may be safe to assume that the missing value can be replaced with *01-03-2016*. The missing value for chocolate sales may be more difficult to discern, though you could conceivably use the mean value for the column; which given how little variance there seems to be in the data may be adequate. Alternatively, if you can't replace the missing value without compromising the validity of the model you are creating, it may be better to simply remove the entire row.

While missing values are often indicated by blank (or *null*) cells, they may be more difficult to spot. Some systems automatically replace missing values with *NA*, or sometimes missing numeric values are replaced with the text string *NaN* (not a number). Additionally, some systems insert a placeholder value such as *0* or *9999* to represent a missing value; which can be a difficult to find source of inaccuracies in a model. For example, the table above includes a vanilla sales figure of *0*. It is possible that no vanilla cupcakes were sold on that date, but based on the relatively consistent values for vanilla sales in the other rows, it's also possible that the actual sales figure is unknown and *0* has been inserted as a placeholder. Knowing how to determine whether a value is missing, and what to do about it depends on familiarity with data and the goals of your analysis.

You can detect and treat missing values with the **Clean Missing Data** module in Azure ML, or you can write custom R, Python, or SQL code. For more information about the **Clean Missing Data** module, see https://msdn.microsoft.com/en-us/library/azure/dn906028.aspx.

Duplicate Rows

The cupcake sales data described previously also includes two rows containing exactly the same values for all columns. Given that the dataset seems to represent daily sales totals, it seems likely that no two rows should share the same date; and since the other column values are identical, it is logical to assume that these rows are duplicates and one should be removed from the dataset.

However, handling duplicates may not always be this straightforward. In this example, we can assume that each row is uniquely identified by a key column (date), but what should be done about a case where two rows contain the same date but have different values in the other columns? You could opt to keep the first or last row that occurs, or you could try to merge the duplicate rows by assigning an average value for the numeric columns that share the same key.

Additionally, some datasets do not contain key columns that uniquely identify each row. In this case, you may find multiple rows that contain the same values for every column and assume they are duplicates – but be careful, it could also be that the rows represent different data points that happen to exhibit exactly the same features. As with handling missing values, how you identify and treat duplicates depends on your knowledge of the data and the use to which you intend to put it.

You can detect and remove duplicate rows with the **Remove Duplicate Rows** module in Azure ML, or you can write custom R, Python, or SQL code. For more information about the **Remove Duplicate Rows** module, see https://msdn.microsoft.com/en-us/library/azure/dn905805.aspx.

Feature Engineering

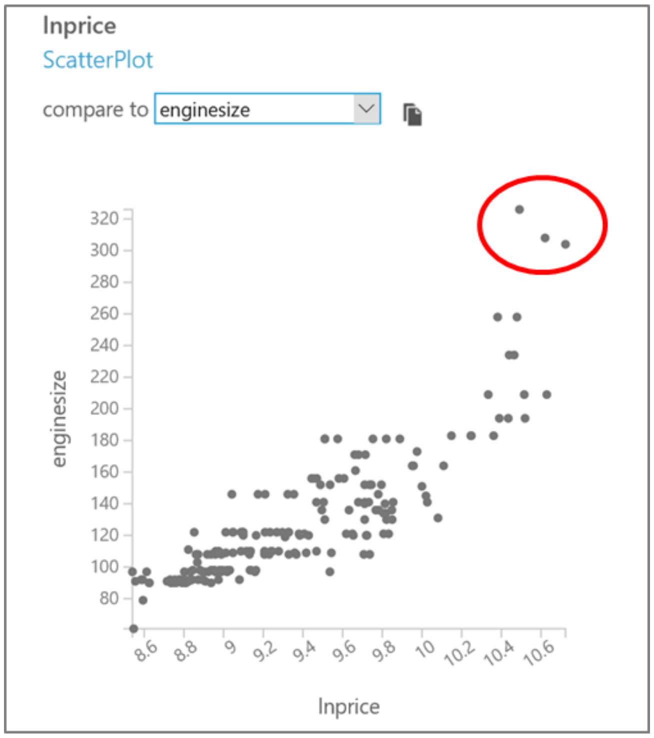
Sometimes you may need to create calculated values in your dataset that don't exist in the source data. This may be to combine multiple dimensions in the data into a single composite feature, or it may be to apply a logarithmic or other transformation to a numeric value in order to create a more distinct fit between features and the labels that you want to use them to predict.

The generation of custom calculated columns is generally referred to as *feature engineering*, and is a common technique in data modeling. You can use the **Apply Math Operation** Azure Machine Learning module to calculate a new column or you can use a custom R, Python, and SQL script. For more information about the **Apply Math Operation** module, see https://msdn.microsoft.com/en-us/library/azure/dn905975.aspx.

Outliers and Errors

Large datasets often include values that are errors or outliers, which will skew the relationships in the model. Finding outliers involves comprehensive exploration and visualization of the data, and you must be careful to ensure that the outliers you identify are genuine outliers that should be treated, and not indicators that there are some important subsets within the data that should be taken into account in the model.

The first step to identifying outliers is often to visualize the relationships between important features and labels as a scatterplot, and looking for plot points that fall outside of the apparent pattern in the data, as shown here:

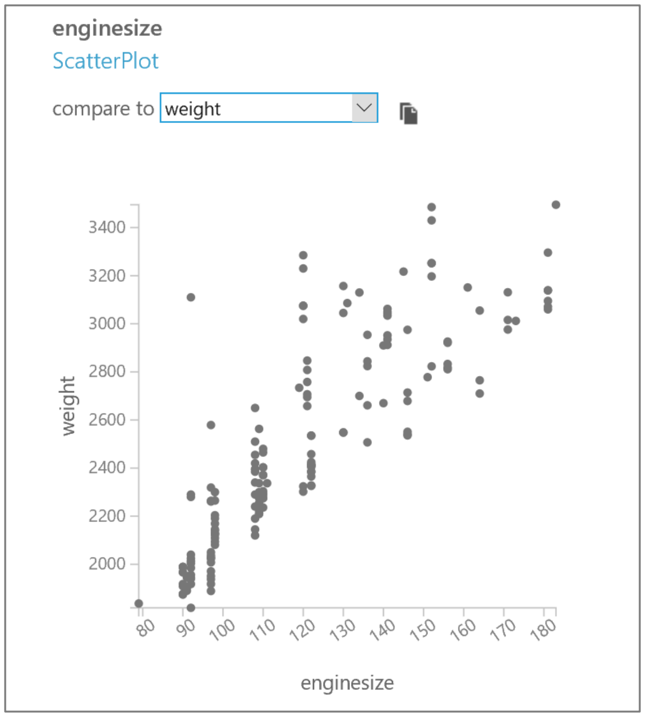


You should typically spend time comparing various plots in order to confirm that outliers are genuine, and then deciding how to treat them. Treatments include deleting rows containing outliers, or replacing outlier values with the maximum or minimum values (depending if they are above or below the normal ramge) or the mean value of the column.

You can treat outliers with the **Clip Values** module, or you can write custom R, Python, or SQL scripts to handle outliers. For more information about the **Clip Values** module, see https://msdn.microsoft.com/en-us/library/azure/dn905918.aspx.

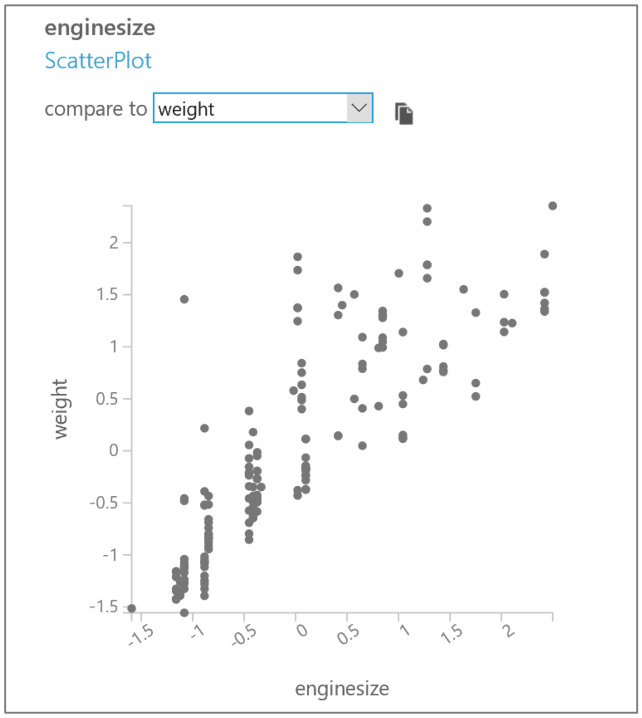
Scaling Data

Most datasets used in data science contain multiple numeric features, and these features are often expressed in different units of measurement. For example, the following scatterplot shows the relationship between the engine size and weight of automobiles.



When creating a model, the important thing is the relative relationship between the numeric features – not necessarily the absolute values. In this case, the absolute numeric values are on completely different scales; weights range between 1000 and 3500, and engine size ranges from 80 to 185. When you take into consideration that there may be many more numeric features, each on their own scale of values, it can become very difficult to compare multiple features in a model.

To address this problem, you can scale (or *normalize*) the numeric features in your dataset so that the data values are within a consistent scale, without losing the relative relationships between the features. For example, the following scatterplot shows the relationship between engine size and weight with the values normalized to a common scale.



Note that the numerical values for the features are now on the same scale, but the relative relationships between them are still apparent.

You can scale data using custom R or Python code, or you can use the **Normalize Data** module in Azure Machine Learning. For more information about the **Normalize Data** module, see https://msdn.microsoft.com/en-us/library/azure/dn905838.aspx.

Note: You should generally normalize data *after* you have removed rows containing missing values, duplicates, and outliers. This ensures that the scale is not skewed by extreme values that will not be used by the model.