Flag missing values

One strategy for accounting for missing values is to simply ignore them. This is usually not a good idea because you often have little insight into how the missing data influenced the results.

Many machine learning models require a feature matrix that does not have missing values. At a minimum, you may need to convert missing elements to a flag that can be ingested by the model. For example, when dealing with missing values in a column of categorical data, an option is to treat “missing” as one of the categories for the model to train on. In such a scenario, it is important to follow the standard practice of “dummifying” the categorical variable, such as with Pandas’ get_dummies method.

Converting a *numerical* missing value to a flag to denote that it is missing is generally not good practice, because the choice of the value of the flag will have implications on how your model treats such data. It is usually best to use an imputation technique, such as those discussed below. The fact that a value was missing may be useful to track. You can do so even when you have imputed a replacement value, by adding a new column to the dataset that flags whether values from the newly complete column were originally missing.

Complete case analysis

The opposite strategy from tracking all your missing values is to delete either rows (observations) or columns (features), from your training data. In the case of deleting rows, this is called complete case analysis, and is quite common.

Complete case analysis can lead to undesirable results, but the degree to which it does depends on the *category of missingness*. The categories of missingness are detailed in the following sections.

If you plan to use a predictive model on the data that you have, you will still need a plan to account for missing values. Complete case models in training can yield additional problems in the future. Sometimes complete-case analysis is used as a first pass through the workflow.

Deleting rows by filtering out rows of Pandas DataFrames that have missing data is straightforward:

```
import pandas as pd
from numpy import nan

df = pd.DataFrame({'name':['apple','banana','orange'],
                   'price':[1.95, 3.00, nan], 'inventory':[nan, 12, 23]})

print(df)

     name  price  inventory
0   apple   1.95        NaN
1  banana   3.00       12.0
2  orange    NaN       23.0

print(df.dropna())

     name  price  inventory
1  banana    3.0       12.0

```

The default behavior of the .dropna method is to return a new DataFrame containing only the complete rows.

This method naturally has additional functionality, such as dropping columns rather than rows:

```
print(df.dropna(axis = 'columns'))

     name
0   apple
1  banana
2  orange

```

Note that dealing with missing values from a column-wise perspective really falls within Feature Engineering (since there are a number of reasons why one might exclude a column/feature from a model, beyond just whether that feature has missing values). This topic is covered in more detail in the next module.