Missing Data

This lesson focuses on missing data by explaining some techniques to clean such data from our dataset.

WE'LL COVER THE FOLLOWING

What is missing data?

So, you have read in your data only to discover that some values are missing! What do you do?

First, you should try to understand why your data points are missing. Are they missing at random or not? Data that is missing at random could be removed with a large enough dataset. If your data is missing for a reason, perhaps missing means zero? Or is it a strong signal of a sensor malfunction? These non-random missing values should be fixed or leveraged. For example, set them to zero if they should be zero.

Second, you should consider how much data you have relative to how many rows have missing data. If you have 1 million data points with 10 points missing at random, you are probably okay to drop those rows. On the other hand, if you have 300 data points with 100 are missing, then you should probably be hesitant about removing 1/3 of your data. Assuming you can't find an underlying reason for the missing data, how do you handle them?

Filling missing data

There are a few ways to fill in missing values.

Using a statistical value

Fill in a column's missing values with a **statistical value** such as the mean, median, or mode. This has the advantage of being simple and easy to do but also introduces a smaller variance within your column than otherwise would not be present. High variance within a column is usually beneficial (assuming it is real) to a machine learning model. Also, it might not make a lot of sense. For example, imagine a data set of heights and weights with missing weights. You then fill all the missing weights with the average regardless of height. We know that height and weight are correlated, but we are not taking advantage of that knowledge.

Let's look at a simple example of our first idea of filling in missing values with a statistical value.

```
import numpy as np
import pandas as pd
pd_series = pd.Series([5, 10, np.nan, 15, 20, np.nan, 25, 50, np.nan])
print("Average of non-missing values: {0}".format(pd_series.mean()))
pd_series = pd_series.fillna(pd_series.mean())
print(pd_series)
```

Above, we created a Pandas series, which is just a single column of a Pandas dataframe. Within our series, we added 3 missing values with `np.nan`.

Pandas allows you to easily fill in missing values with the `fillna()` function. We pass this function the value we want to use. In this example, we pass the mean of the values that are not missing.

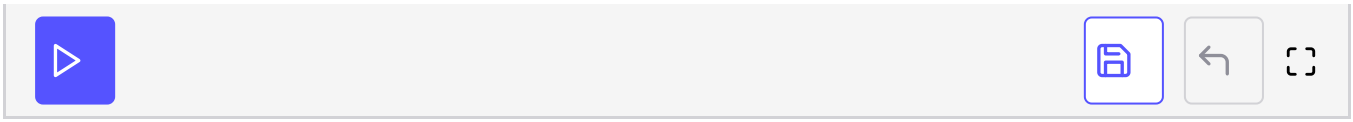
Using a model

Use a model to learn values that seem reasonable for missing values. Basically, leverage the data that is not missing to learn what the missing variable might be. In our height and weight example, you could use height to predict weight when not missing and then apply that model to heights with missing weights to fill in your missing data. A popular algorithm for this is K-Nearest Neighbors. For some more ideas on how to deal with missing data, take a look at this article.

Tracking and dropping missing data

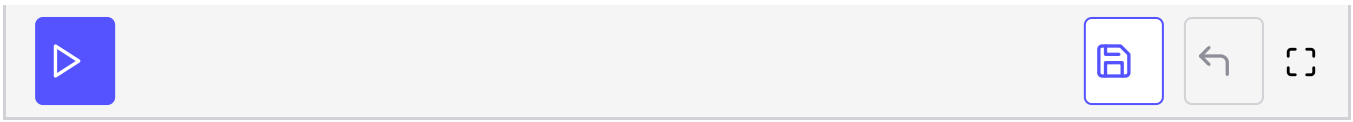
Pandas also makes it easy to **drop** missing values using the `dropna()` function:

```
import numpy as np
import pandas as pd

# Create a series with missing data
pd_series = pd.Series([5, 10, np.nan, 15, 20, np.nan, 25, 50, np.nan])
# Drop rows with missing data
pd_series = pd_series.dropna()
print(pd_series)
```



Or you can find which values are missing with the `isnull()` function.

```
import numpy as np
import pandas as pd

# Create a series with missing values
pd_series = pd.Series([5, 10, np.nan, 15, 20, np.nan, 25, 50, np.nan])
# Show which rows are missing
print(pd_series.isnull())
```



You are returned a new series for which a **True** value means the actual value is missing and a **False** value means there is an actual value.

Now that you're familiar with missing data, in the next lesson, you'll be studying how to handle data that varies significantly from other observations.