Andy McDonald

Jun 10, 2021
·
8 min read
·
Listen

Using the missingno Python library to Identify and Visualise Missing Data Prior to Machine Learning

An example using petrophysical well log measurements

Photo by Tim Mossholder on Unsplash

Data exploration and pre-processing are important steps within any data science or machine learning workflow. When working on tutorial or training datasets it can be the case that they have been engineered in a way to make it easy to work with and allow the algorithm being discussed to be run successfully. However, in the real world, data is messy! It can have erroneous values, incorrect labels and parts of it can be missing.

Missing data is probably one of the most common issues when working with real datasets. Data can be missing for a multitude of reasons, including sensor failure, data vintage, improper data management, and even human error. Missing data can occur as single values, multiple values within one feature, or entire features may be missing.

It is important that missing data is identified and handled appropriately prior to further data analysis or machine learning. Many machine learning algorithms can't handle missing data and require entire rows, where a single missing value is present, to be deleted or replaced (imputed) with a new value.

Depending on the source of the data missing values may be represented in different ways. The most common is NaN (Not a Number), however, other variations can include "NA", "None", "-999", "0", " ", "-". If the missing data is represented by something other than NaN in your dataframe, then it should be converted to NaN using `np.NaN` as seen below.

```
df.replace('', np.NaN)
```

The following video accompanies this article:

The missingno Library

Missingno is an excellent and simple to use Python library that provides a series of visualisations to understand the presence and distribution of missing data within a pandas dataframe. This can be in the form of either a barplot, matrix plot, heatmap, or a dendrogram. The original publication for the library can be found here.

From these plots, we can identify where missing values occur, the extent of the missingness and whether any of the missing values are correlated with each other. Often, missing values may be seen as not contributing any information, but if analysed closely there may be an underlying story.

The missingno library can be installed using the pip command:

```
pip install missingno
```

The Dataset

For this tutorial, we will use a subset of the publicly available dataset from the Machine Learning competition run by Xeek and FORCE 2020. The objective of the competition was to predict lithology from existing labeled data. The dataset consists of 118 wells from the Norwegian Sea.

The data contains a series of electrical measurements that have been acquired by well logging tools. The measurements are used to characterise the subsurface geology and identify suitable hydrocarbon reservoirs.

The data and notebook for this article can be found on my GitHub repository at https://github.com/andymcdgeo/missingno_tutorial

Importing Libraries and Loading the Data

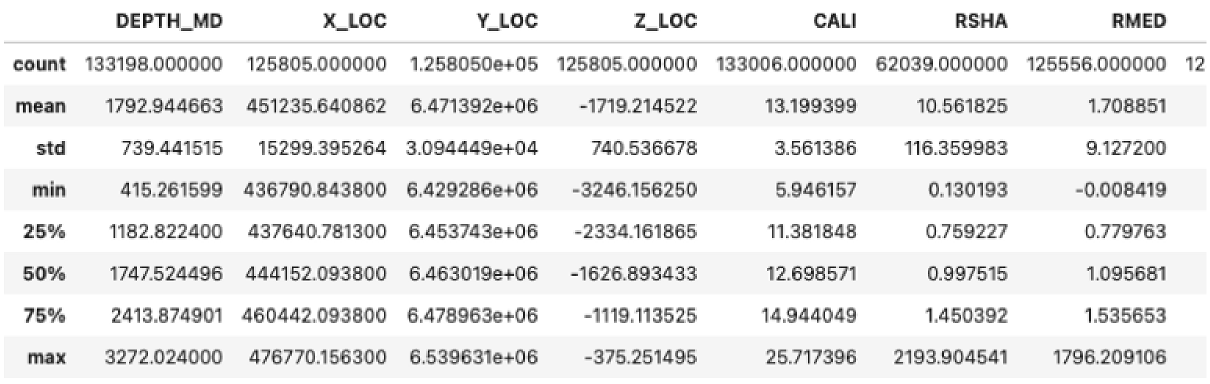
The first step in the process is to import the libraries. For this article, we will be working with pandas for load and storing our data and missingno for visualising data completeness.

```
import pandas as pd
import missingno as msnodf = pd.read_csv('xeek_train_subset.csv')
```

Quick Analysis with Pandas

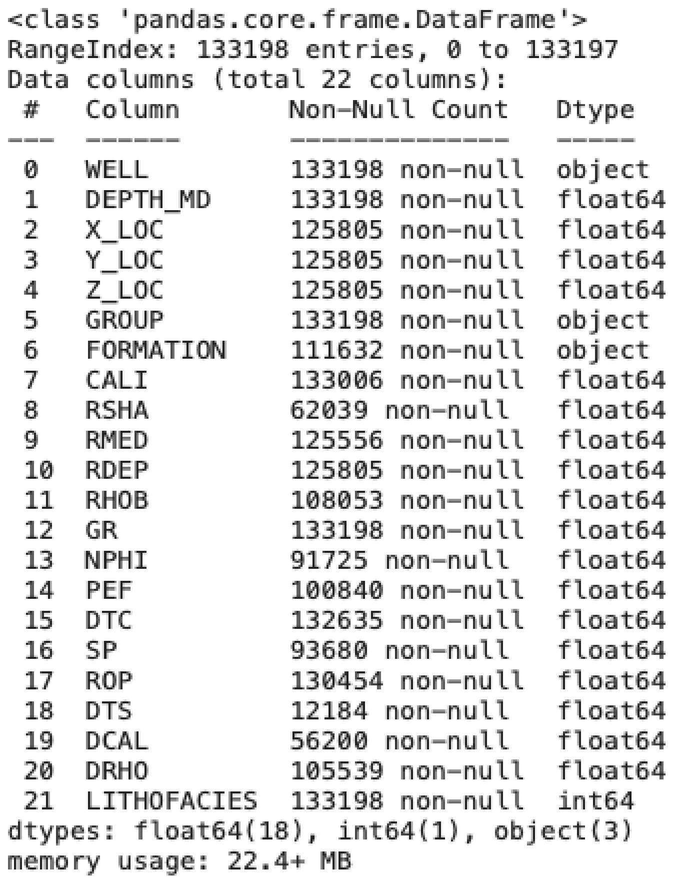
Before we use the missingno library, there are a few features within the pandas library that can give us an initial insight into how much missing data.

The first is using the `.describe()` method. This returns a table containing summary statistics about the dataframe such as the mean, maximum and minimum values. At the top of the table is a row called counts. In the example below, we can see that we having varying counts for each of the features within the dataframe. This provides an initial indication that not all values are present.



	DEPTH_MD	X_LOC	Y_LOC	Z_LOC	CALI	RSHA	RMED	
count	133198.000000	125805.000000	1.258050e+05	125805.000000	133006.000000	62039.000000	125556.000000	12
mean	1792.944663	451235.640862	6.471392e+06	-1719.214522	13.199399	10.561825	1.708851	
std	739.441515	15299.395264	3.094449e+04	740.536678	3.561386	116.359983	9.127200	
min	415.261599	436790.843800	6.429286e+06	-3246.156250	5.946157	0.130193	-0.008419	
25%	1182.822400	437640.781300	6.453743e+06	-2334.161865	11.381848	0.759227	0.779763	
50%	1747.524496	444152.093800	6.463019e+06	-1626.893433	12.698571	0.997515	1.095681	
75%	2413.874901	460442.093800	6.478963e+06	-1119.113525	14.944049	1.450392	1.535653	
max	3272.024000	476770.156300	6.539631e+06	-375.251495	25.717396	2193.904541	1796.209106	

We can take this one step further and use the `.info()` method. This will return back a summary of the dataframe as well as a count of the non-null values.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 133198 entries, 0 to 133197
Data columns (total 22 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   WELL         133198 non-null  object
 1   DEPTH_MD     133198 non-null  float64
 2   X_LOC        125805 non-null  float64
 3   Y_LOC        125805 non-null  float64
 4   Z_LOC        125805 non-null  float64
 5   GROUP        133198 non-null  object
 6   FORMATION    111632 non-null  object
 7   CALI         133006 non-null  float64
 8   RSHA         62039 non-null   float64
 9   RMED         125556 non-null  float64
 10  RDEP         125805 non-null  float64
 11  RHOB         108053 non-null  float64
 12  GR           133198 non-null  float64
 13  NPHI         91725 non-null   float64
 14  PEF          100840 non-null  float64
 15  DTC          132635 non-null  float64
 16  SP           93680 non-null   float64
 17  ROP          130454 non-null  float64
 18  DTS          12184 non-null   float64
 19  DCAL         56200 non-null   float64
 20  DRHO         105539 non-null  float64
 21  LITHOFACIES  133198 non-null  int64
dtypes: float64(18), int64(1), object(3)
memory usage: 22.4+ MB
```

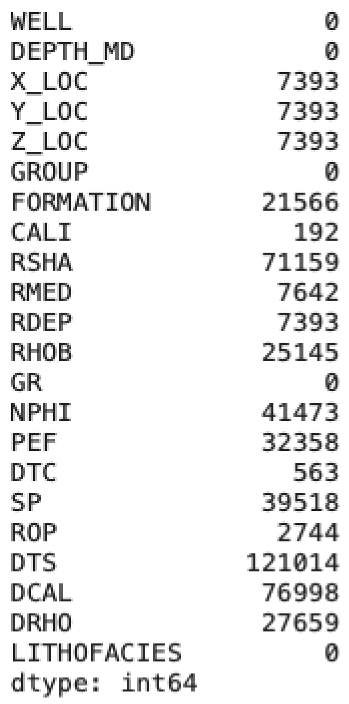
We can see from the example above that we have a more concise summary of the state of the data and the extent of data missingness.

Another quick method we can use is:

```
df.isna().sum()
```

This returns a summary of how many missing values are contained within the dataframe. The `isna()` part detects missing values within the dataframe and returns a Boolean value for each element in the dataframe. The `sum()` part sums up the number of True values.

The following information is returned by this line.



```
WELL                0
DEPTH_MD            0
X_LOC            7393
Y_LOC            7393
Z_LOC            7393
GROUP               0
FORMATION       21566
CALI              192
RSHA            71159
RMED             7642
RDEP             7393
RHOB            25145
GR                  0
NPHI            41473
PEF             32358
DTC               563
SP              39518
ROP              2744
DTS            121014
DCAL            76998
DRHO            27659
LITHOFACIES         0
dtype: int64
```

From this summary, we can see that a number of columns, namely WELL, DEPTH_MD, GROUP, GR, and LITHOFACIES have no null values. All others have a large and varying degree of missing values.

Using missingno to Identify Missing Data

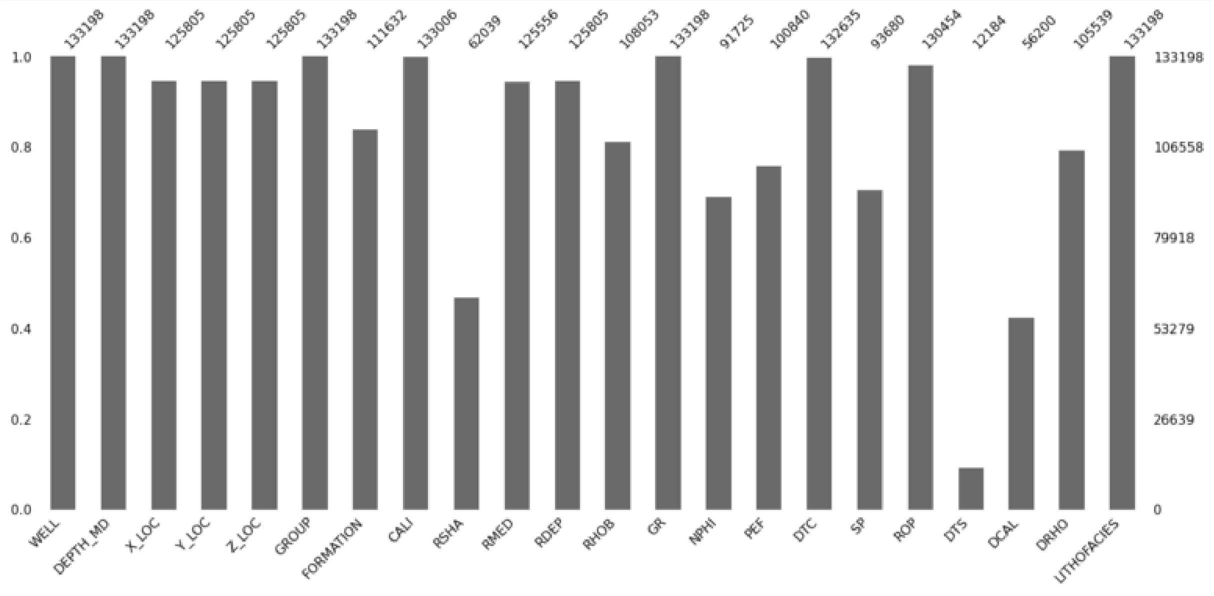
Within the missingno library, there are four types of plots for visualising data completeness: the barplot, the matrix plot, the heatmap, and the dendrogram plot. Each has its own advantages for identifying missing data.

Let’s take a look at each of these in turn.

Barplot

The barplot provides a simple plot where each bar represents a column within the dataframe. The height of the bar indicates how complete that column is, i.e, how many non-null values are present. It can be generated by calling upon:

```
msno.bar(df)
```



On the left side of the plot, the y-axis scale ranges from 0.0 to 1.0, where 1.0 represents 100% data completeness. If the bar is less than this, it indicates that we have missing values within that column.

On the right side of the plot, the scale is measured in index values. With the top right representing the maximum number of rows within the dataframe.

Along the top of the plot, there are a series of numbers that represent the total count of the non-null values within that column.

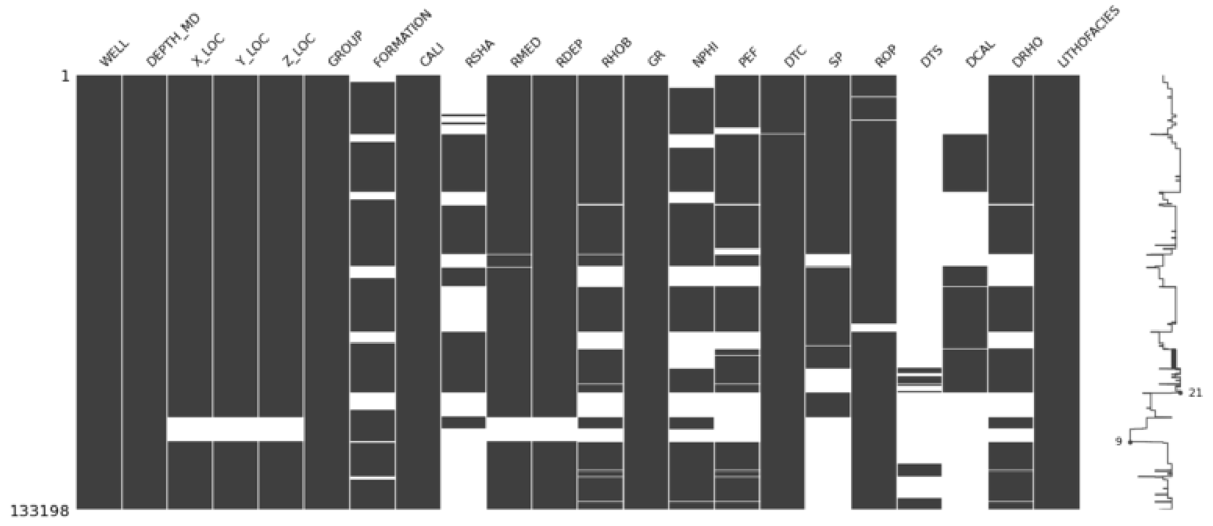
In this example we can see that a number of the columns (DTS, DCAL and RSHA) have a large amount of missing values. Other columns (e.g. WELL, DEPTH_MD and GR) are complete and have the maximum number of values.

Matrix Plot

The matrix plot is a great tool if you are working with depth-related data or time-series data. It provides a colour fill for each column. When data is present, the plot is shaded in grey (or your colour of choice), and when it is absent the plot is displayed in white.

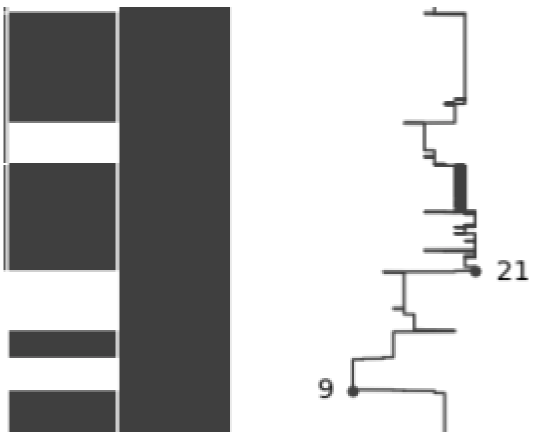
The matrix plot can be generated by calling upon:

```
msno.matrix(df)
```



missingno matrix plot showing data sparsity across all dataframe columns.

As seen in the resultant plot, the columns DTS, DCAL, and RSHA show large portions of missing data. This was identified in the bar plot, but the added benefit is you can view how that missing data is distributed in the dataframe.



Close-up view of the missingno sparkline. Image by the author.

On the right side of the plot is a sparkline that ranges from 0 on the left to the total number of columns in the dataframe on the right. A closeup can be seen above. When a row has a value in each column, the line will be at the maximum right position. As missing values start to increase within that row the line will move towards the left.

Heatmap

The heatmap is used to identify correlations of the nullity between each of the different columns. In other words, it can be used to identify if there is a relationship in the presence of null values between each of the columns.

Values close to positive 1 indicate that the presence of null values in one column is correlated with the presence of null values in another column.

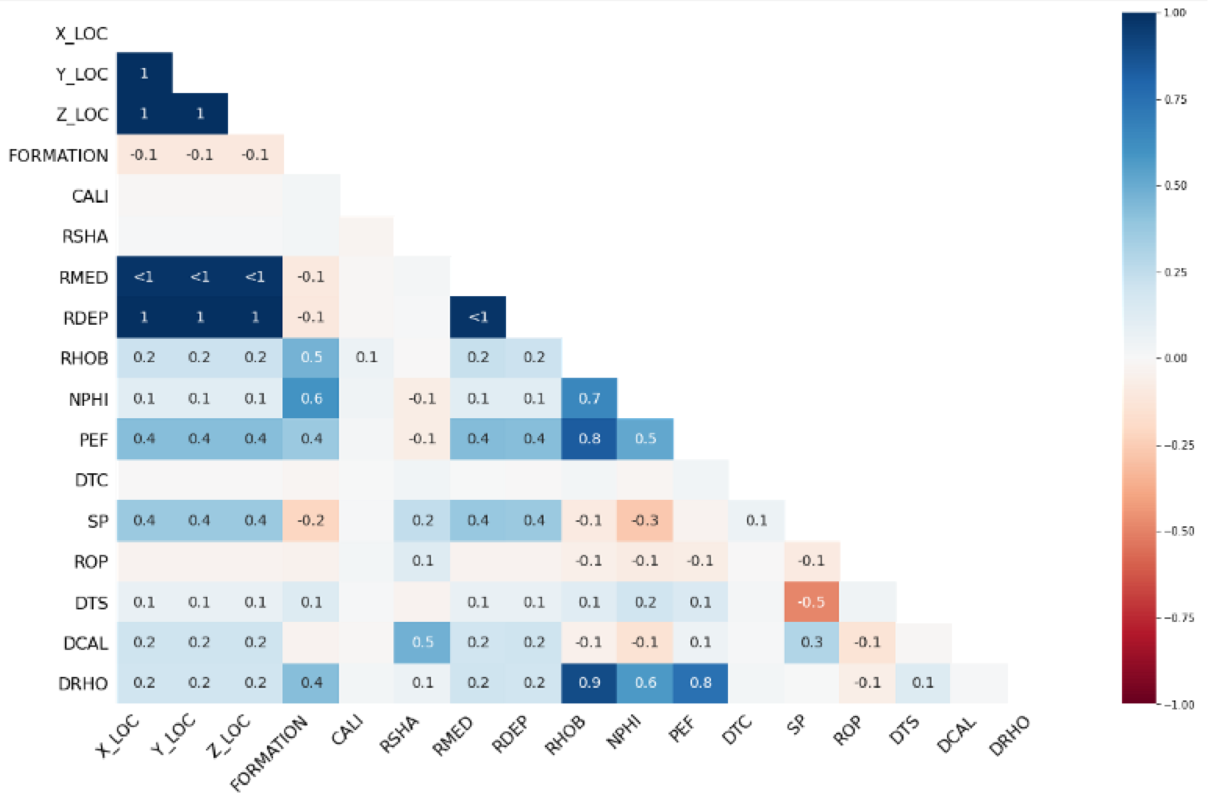
Values close to negative 1 indicate that the presence of null values in one column is anti-correlated with the presence of null values in another column. In other words, when null values are present in one column, there are data values present in the other column, and vice versa.

Values close to 0, indicate there is little to no relationship between the presence of null values in one column compared to another.

There are a number of values that show as <-1. This indicates that the correlation is very close to being 100% negative.

The heatmap can be generated by the following code:

```
msno.heatmap(df)
```



missingno heatmap plot illustrating the correlation in nullity between data columns. Image by the author.

Here we can see that the ROP column is slightly negatively correlated with the RHOB, NPHI and PEF columns, and slightly positively correlated with RSHA. If we take a look at DRHO, its absence is highly correlated with missing values in the RHOB, NPHI and PEF columns.

The heatmap approach is more suitable for smaller datasets.

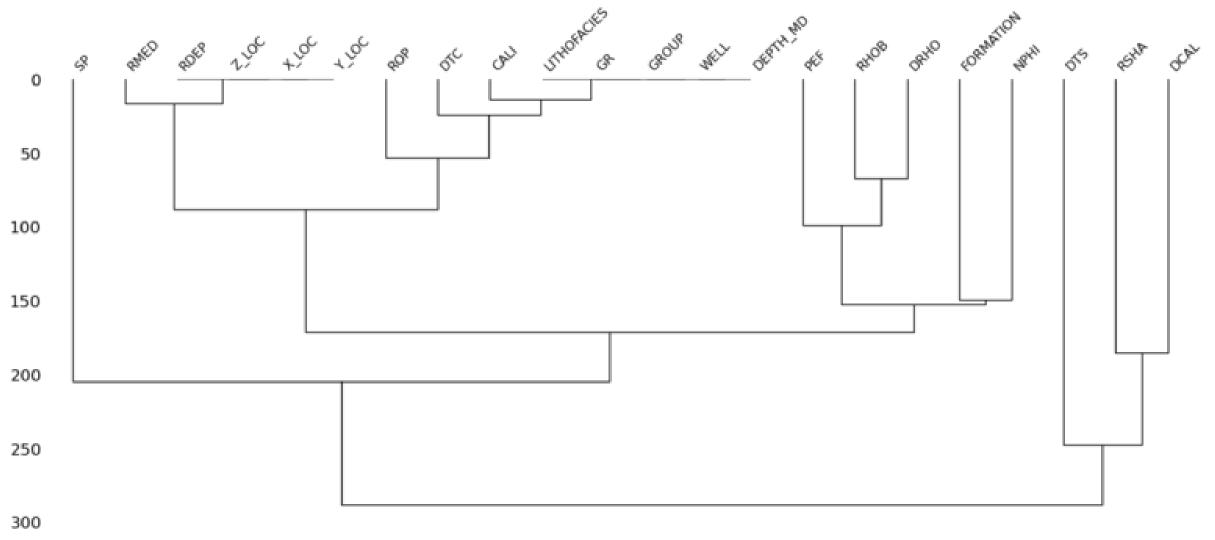
Dendrogram

The dendrogram plot provides a tree-like graph generated through hierarchical clustering and groups together columns that have strong correlations in nullity.

If a number of columns are grouped together at level zero, then the presence of nulls in one of those columns is directly related to the presence or absence of nulls in the others columns. The more separated the columns in the tree, the less likely the null values can be correlated between the columns.

The dendrogram can be generated by:

```
msno.dendrogram(df)
```



mssingno dendrogram illustrating the correlation in nullity between the well log measurements.

In the dendrogram plot above, we can see we have two distinct groups. The first is on the right side (DTS, RSHA, and DCAL) which all have a high degree of null values. The second is on the left, with the remainder of the columns which are more complete.

LITHOFACIES, GR, GROUP, WELL, and DEPTH_MD are all grouped together at zero indicating that they are complete.

RDEP, Z_LOC, X_LOC, and Y_LOC are grouped together close to zero. RMED is in the same larger branch suggesting that some of the missing values present within that column can be correlated with these four columns.

Summary

Identifying missing prior to applying machine learning is a key component of the data quality workflow. This can be achieved using the missingno library and a series of visualisations to understand how much missing data is present, where it occurs, and how the occurrence of missing values is related between the different data columns.

Thanks for reading!

If you have found this article useful, please feel free to check out my other articles looking at various aspects of Python and well log data. You can also find my code used in this article and others at GitHub.

If you want to get in touch you can find me on LinkedIn or at my website.

Interested in learning more about python and well log data or petrophysics? Follow me on Medium.

If you have enjoyed this article or any others and want to show your appreciation you are welcome to Buy Me a Coffee

References

Bilogur, (2018). Missingno: a missing data visualization suite. Journal of Open Source Software, 3(22), 547, https://doi.org/10.21105/joss.00547

Bormann, Peter, Aursand, Peder, Dilib, Fahad, Manral, Surrender, & Dischington, Peter. (2020). FORCE 2020 Well well log and lithofacies dataset for machine learning competition [Data set]. Zenodo. http://doi.org/10.5281/zenodo.4351156