HANDLING MISSING DATA IN PANDAS: NAN VALUES EXPLAINED

In applied data science, you will usually have missing data. For example, an industrial application with sensors will have sensor data that is missing on certain days.

You have a couple of alternatives to work with missing data. You can:

- Drop the whole row
- Fill the row-column combination with some value

It would not make sense to drop the column as that would throw away that metric for all rows. So, let's look at how to handle these scenarios.

(This tutorial is part of our Pandas Guide. Use the right-hand menu to navigate.)

NaN means missing data

Missing data is labelled **NaN.**

Note that **np.nan** is not equal to Python **Non**e. Note also that np.nan is not even to np.nan as np.nan basically means **undefined**.

Here make a dataframe with 3 columns and 3 rows. The array np.arange(1,4) is copied into each row.

```
import pandas as pd
import numpy as np
df = pd.DataFrame(,index=,
columns=)
```

Results:

	X	Y	Z
a	1	2	3
b	1	2	3
c	1	2	3

Now reindex this array adding an index **d**. Since d has no value it is filled with **NaN**.

`df.reindex(index=)`

	X	Y	Z
a	1.0	2.0	3.0
b	1.0	2.0	3.0
c	1.0	2.0	3.0
d	NaN	NaN	NaN

isna

Now use **isna** to check for missing values.

`pd.isna(df)`

	X	Y	Z
a	False	False	False
b	False	False	False
c	False	False	False
d	True	True	True

notna

The opposite check—looking for actual values—is **notna()**.

```
pd.notna(df)
```

	X	Y	Z
a	True	True	True
b	True	True	True
c	True	True	True
d	False	False	False

nat

nat means a missing date.

```
df = pd.Timestamp('20211225')
df.loc = np.nan
```

	X	Y	Z	time
a	1.0	2.0	3.0	2021-12-25
b	1.0	2.0	3.0	2021-12-25
c	1.0	2.0	3.0	2021-12-25
d	NaN	NaN	NaN	NaT

fillna

Here we can fill NaN values with the integer 1 using **fillna(1)**. The date column is not changed since the integer 1 is not a date.

```
df=df.fillna(1)
```

	X	Y	Z	time
a	1.0	2.0	3.0	2021-12-25 00:00:00
b	1.0	2.0	3.0	2021-12-25 00:00:00
c	1.0	2.0	3.0	2021-12-25 00:00:00
d	1.0	1.0	1.0	1

To fix that, fill empty time values with:

```
df.fillna(pd.Timestamp('20221225'))
```

dropna()

dropna() means to drop rows or columns whose value is empty. Another way to say that is to show only rows or columns that are not empty.

Here we fill row c with **NaN**:

```
df = pd.DataFrame(,index=,
columns=)
df.loc=np.NaN
```

	X	Y	Z
a	1.0	2.0	3.0
b	1.0	2.0	3.0
c	NaN	NaN	NaN

Then run **dropna** over the row (axis=0) axis.

```
df.dropna()
```

You could also write:

```
df.dropna(axis=0)
```

All rows except c were dropped:

	X	Y	Z
a	1.0	2.0	3.0
b	1.0	2.0	3.0

To drop the column:

```
df = pd.DataFrame(,index=,
columns=)
df=np.NaN
```

	X	Y	Z	V
a	1	2	3	NaN
b	1	2	3	NaN
c	1	2	3	NaN

```
df.dropna(axis=1)
```

	X	Y	Z
a	1	2	3
b	1	2	3
c	1	2	3

interpolate

Another feature of Pandas is that it will fill in missing values using what is logical.

Consider a time series—let's say you're monitoring some machine and on certain days it fails to report. Below it reports on Christmas and every other day that week. Then we reindex the Pandas Series, creating gaps in our timeline.

```
import pandas as pd
```

```
import numpy as np
arr=np.array()
idx=np.array()
df = pd.DataFrame(arr,index=idx)
idx=
df=df.reindex(index=idx)
```

2021-12-25	1.0
2021-12-26	NaN
2021-12-27	2.0
2021-12-28	NaN
2021-12-29	3.0

We use the **interpolate()** function. Pandas fills them in nicely using the midpoints between the points. Of course, if this was curvilinear it would fit a function to that and find the average another way.

```
df=df.interpolate()
```

2021-12-25	1.0
2021-12-26	1.5
2021-12-27	2.0
2021-12-28	2.5
2021-12-29	3.0

That concludes this tutorial.

Related reading

- BMC Machine Learning & Big Data Blog
- Pandas Data Types
- Python Development Tools: Your Python Starter Kit
- Snowflake Guide, a series of tutorials
- Enabling the Citizen Data Scientists