Python missing data

Operating on Null Values

As we have seen, Pandas treats None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful methods for detecting, removing, and replacing null values in Pandas data structures.

They are:

isnull()

Generate a Boolean mask indicating missing values

notnull()

Opposite of isnull()

dropna()

Return a filtered version of the data

fillna()

Return a copy of the data with missing values filled or imputed

Detecting null values

Pandas data structures have two useful methods for detecting null data: **isnull() and notnull()** The isnull() and notnull() methods produce similar Boolean results for Data Frames.

```
import pandas as pd
df = pd.read_csv("/WEATHER_DATA.csv",parse_dates=['day'])
type(df.day[0])
df
```

```
         day  temperature  windspeed   event
0 2017-01-01         32.0        6.0    Rain
1 2017-01-04          NaN        9.0   Sunny
2 2017-01-05         28.0        NaN    Snow
3 2017-01-06          NaN        7.0     NaN
4 2017-01-07         32.0        NaN    Rain
5 2017-01-08          NaN        NaN   Sunny
6 2017-01-09          NaN        NaN     NaN
7 2017-01-10         34.0        8.0  Cloudy
8 2017-01-11         40.0       12.0   Sunny
```

```
df.set_index('day',inplace=True)
df
```

```
            temperature  windspeed   event
day
2017-01-01         32.0        6.0    Rain
2017-01-04          NaN        9.0   Sunny
2017-01-05         28.0        NaN    Snow
2017-01-06          NaN        7.0     NaN
2017-01-07         32.0        NaN    Rain
2017-01-08          NaN        NaN   Sunny
2017-01-09          NaN        NaN     NaN
2017-01-10         34.0        8.0  Cloudy
2017-01-11         40.0       12.0   Sunny
```

```
df.isnull()
```

```
            temperature  windspeed  event
day
2017-01-01        False      False  False
2017-01-04         True      False  False
2017-01-05        False       True  False
2017-01-06         True      False   True
2017-01-07        False       True  False
2017-01-08         True       True  False
2017-01-09         True       True   True
2017-01-10        False      False  False
2017-01-11        False      False  False
```

```
df.notnull()
```

```
            temperature  windspeed  event
day
2017-01-01         True       True   True
2017-01-04        False       True   True
2017-01-05         True      False   True
2017-01-06        False       True  False
2017-01-07         True      False   True
2017-01-08        False      False   True
2017-01-09        False      False  False
2017-01-10         True       True   True
2017-01-11         True       True   True
```

Dropping null values

dropna() will drop all rows in which any null value is present

```
df1=df.dropna()
df1
```

```
            temperature  windspeed   event
day
2017-01-01         32.0        6.0    Rain
2017-01-10         34.0        8.0  Cloudy
2017-01-11         40.0       12.0   Sunny
```

you can drop NA values along a different axis; axis=1 drops all columns containing a null value

```
df1=df.dropna(axis='columns')
df1
```

```
Empty DataFrame
Columns: []
Index: [2017-01-01 00:00:00, 2017-01-04 00:00:00, 2017-01-05 00:00:00,
2017-01-06 00:00:00, 2017-01-07 00:00:00, 2017-01-08 00:00:00, 2017-
01-09 00:00:00, 2017-01-10 00:00:00, 2017-01-11 00:00:00]
```

This drops some good data as well; you might rather be interested in dropping rows or columns with all NA values, or a majority of NA values. This can be specified through the **how or thresh parameters**, which allow fine control of the number of nulls to allow through.

The default is **how='any'**, such that any row or column (depending on the axis key-word) containing a null value will be dropped. You can also specify how='all', which will only drop rows/columns that are all null values:

```
df1=df.dropna( how='all')
df1
```

```
            temperature  windspeed   event
day
2017-01-01         32.0        6.0    Rain
2017-01-04          NaN        9.0   Sunny
2017-01-05         28.0        NaN    Snow
2017-01-06          NaN        7.0     NaN
2017-01-07         32.0        NaN    Rain
2017-01-08          NaN        NaN   Sunny
2017-01-10         34.0        8.0  Cloudy
2017-01-11         40.0       12.0   Sunny
```

```
df1=df.dropna(axis='rows', thresh=3)
df1
```

```
            temperature  windspeed   event
day
2017-01-01         32.0        6.0    Rain
2017-01-10         34.0        8.0  Cloudy
2017-01-11         40.0       12.0   Sunny
```

```
df1=df.dropna(axis='rows', thresh=2)
df1
```

```
            temperature  windspeed   event
day
2017-01-01         32.0        6.0    Rain
2017-01-04          NaN        9.0   Sunny
2017-01-05         28.0        NaN    Snow
```

```
2017-01-07         32.0        NaN    Rain
2017-01-10         34.0        8.0  Cloudy
2017-01-11         40.0       12.0   Sunny
```

Filling null values

fillna() method, which returns a copy of the array with the null values replaced.

Fill all NaN with one specific value

```
new_df = df.fillna(0)
new_df
```

```
            temperature  windspeed   event
day
2017-01-01         32.0        6.0    Rain
2017-01-04          0.0        9.0   Sunny
2017-01-05         28.0        0.0    Snow
2017-01-06          0.0        7.0       0
2017-01-07         32.0        0.0    Rain
2017-01-08          0.0        0.0   Sunny
2017-01-09          0.0        0.0       0
2017-01-10         34.0        8.0  Cloudy
2017-01-11         40.0       12.0   Sunny
```

we can specify a forward-fill to propagate the previous value forward:

```
new_df = df.fillna(method="ffill")
new_df
```

```
            temperature  windspeed   event
day
2017-01-01         32.0        6.0    Rain
2017-01-04         32.0        9.0   Sunny
2017-01-05         28.0        9.0    Snow
2017-01-06         28.0        7.0    Snow
2017-01-07         32.0        7.0    Rain
2017-01-08         32.0        7.0   Sunny
2017-01-09         32.0        7.0   Sunny
2017-01-10         34.0        8.0  Cloudy
2017-01-11         40.0       12.0   Sunny
```

we can specify a back-fill to propagate the next values backward:

```
new_df = df.fillna(method="bfill")
new_df
```

```
            temperature  windspeed   event
day
2017-01-01         32.0        6.0    Rain
2017-01-04         28.0        9.0   Sunny
2017-01-05         28.0        7.0    Snow
```

```
2017-01-06         32.0        7.0    Rain
2017-01-07         32.0        8.0    Rain
2017-01-08         34.0        8.0   Sunny
2017-01-09         34.0        8.0  Cloudy
2017-01-10         34.0        8.0  Cloudy
2017-01-11         40.0       12.0   Sunny
```

we can also specify an axis along which the fills take place: # axis is either "index" or "columns"

```
new_df = df.fillna(method="bfill", axis="columns")
new_df
```

```
           temperature windspeed   event
day
2017-01-01          32         6    Rain
2017-01-04           9         9   Sunny
2017-01-05          28      Snow    Snow
2017-01-06           7         7     NaN
2017-01-07          32      Rain    Rain
2017-01-08       Sunny     Sunny   Sunny
2017-01-09         NaN       NaN     NaN
2017-01-10          34         8  Cloudy
2017-01-11          40        12   Sunny
```

```
new_df = df.fillna(method="ffill", axis="columns")
new_df
```

```
           temperature windspeed   event
day
2017-01-01          32         6    Rain
2017-01-04         NaN         9   Sunny
2017-01-05          28        28    Snow
2017-01-06         NaN         7       7
2017-01-07          32        32    Rain
2017-01-08         NaN       NaN   Sunny
2017-01-09         NaN       NaN     NaN
2017-01-10          34         8  Cloudy
2017-01-11          40        12   Sunny
```

limit parameter

```
new_df = df.fillna(method="ffill",limit=1)
new_df
```

```
            temperature  windspeed   event
day
2017-01-01         32.0        6.0    Rain
2017-01-04         32.0        9.0   Sunny
2017-01-05         28.0        9.0    Snow
2017-01-06         28.0        7.0    Snow
2017-01-07         32.0        7.0    Rain
```

```
2017-01-08         32.0        NaN   Sunny
2017-01-09          NaN        NaN   Sunny
2017-01-10         34.0        8.0  Cloudy
2017-01-11         40.0       12.0   Sunny
```

Fill na using column names and dict

```
new_df = df.fillna({
        'temperature': 0,
        'windspeed': 0,
        'event': 'No Event'
    })
new_df
```

```
            temperature  windspeed     event
day
2017-01-01         32.0        6.0      Rain
2017-01-04          0.0        9.0     Sunny
2017-01-05         28.0        0.0      Snow
2017-01-06          0.0        7.0  No Event
2017-01-07         32.0        0.0      Rain
2017-01-08          0.0        0.0     Sunny
2017-01-09          0.0        0.0  No Event
2017-01-10         34.0        8.0    Cloudy
2017-01-11         40.0       12.0     Sunny
```

interpolate

Interpolation is a technique in Python used to estimate unknown data points between two known data points

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

```
            temperature  windspeed   event
day
2017-01-01    32.000000       6.00    Rain
2017-01-04    30.000000       9.00   Sunny
2017-01-05    28.000000       8.00    Snow
2017-01-06    30.000000       7.00     NaN
2017-01-07    32.000000       7.25    Rain
2017-01-08    32.666667       7.50   Sunny
2017-01-09    33.333333       7.75     NaN
2017-01-10    34.000000       8.00  Cloudy
2017-01-11    40.000000      12.00   Sunny
```

```
new_df = df.interpolate(method="time")
new_df
```

```
            temperature  windspeed   event
day
2017-01-01    32.000000       6.00    Rain
```

```
2017-01-04    29.000000       9.00   Sunny
2017-01-05    28.000000       8.00    Snow
2017-01-06    30.000000       7.00     NaN
2017-01-07    32.000000       7.25    Rain
2017-01-08    32.666667       7.50   Sunny
2017-01-09    33.333333       7.75     NaN
2017-01-10    34.000000       8.00  Cloudy
2017-01-11    40.000000      12.00   Sunny
```

Notice that in above temperature on 2017-01-04 is 29 instead of 30 (in plain linear interpolate)

There are many other methods for interpolation such as quadratic, piecewise_polynomial, cubic etc. Just google "dataframe interpolate" to see complete documentation

Inserting Missing Dates

```
dt = pd.date_range("01-01-2017","01-11-2017")
idx = pd.DatetimeIndex(dt)
df.reindex(idx)
```

```
            temperature  windspeed   event
2017-01-01         32.0        6.0    Rain
2017-01-02          NaN        NaN     NaN
2017-01-03          NaN        NaN     NaN
2017-01-04          NaN        9.0   Sunny
2017-01-05         28.0        NaN    Snow
2017-01-06          NaN        7.0     NaN
2017-01-07         32.0        NaN    Rain
2017-01-08          NaN        NaN   Sunny
2017-01-09          NaN        NaN     NaN
2017-01-10         34.0        8.0  Cloudy
2017-01-11         40.0       12.0   Sunny
```

```
new_df = df.interpolate(method="time")
new_df
```

```
            temperature  windspeed   event
day
2017-01-01    32.000000       6.00    Rain
2017-01-04    29.000000       9.00   Sunny
2017-01-05    28.000000       8.00    Snow
2017-01-06    30.000000       7.00     NaN
2017-01-07    32.000000       7.25    Rain
2017-01-08    32.666667       7.50   Sunny
2017-01-09    33.333333       7.75     NaN
2017-01-10    34.000000       8.00  Cloudy
2017-01-11    40.000000      12.00   Sunny
```

Handling Missing Data - replace method

```
import pandas as pd
import numpy as np
df = pd.read_csv("/weather_data2.csv")
df
```

```
          day  temperature  windspeed  event
0  01-01-2017           32          6   Rain
1  01-02-2017       -99999          7  Sunny
2  01-03-2017           28     -99999   Snow
3  01-04-2017       -99999          7      0
4  01-05-2017           32     -99999   Rain
5  01-06-2017           31          2  Sunny
6  01-06-2017           34          5      0
```

Replacing single value

```
new_df = df.replace(-99999, value=np.NaN)
new_df
```

```
          day  temperature  windspeed  event
0  01-01-2017         32.0        6.0   Rain
1  01-02-2017          NaN        7.0  Sunny
2  01-03-2017         28.0        NaN   Snow
3  01-04-2017          NaN        7.0      0
4  01-05-2017         32.0        NaN   Rain
5  01-06-2017         31.0        2.0  Sunny
6  01-06-2017         34.0        5.0      0
```

Replacing list with single value

```
new_df = df.replace(to_replace=[-99999,-88888], value=0)
new_df
```

```
          day  temperature  windspeed  event
0  01-01-2017           32          6   Rain
1  01-02-2017            0          7  Sunny
2  01-03-2017           28          0   Snow
3  01-04-2017            0          7      0
4  01-05-2017           32          0   Rain
5  01-06-2017           31          2  Sunny
6  01-06-2017           34          5      0
```

Replacing per column

```
new_df = df.replace({
        'temperature': -99999,
        'windspeed': -99999,
        'event': '0'
    }, np.nan)
new_df
```

```
          day  temperature  windspeed  event
0  01-01-2017         32.0        6.0   Rain
1  01-02-2017          NaN        7.0  Sunny
2  01-03-2017         28.0        NaN   Snow
3  01-04-2017          NaN        7.0    NaN
4  01-05-2017         32.0        NaN   Rain
5  01-06-2017         31.0        2.0  Sunny
6  01-06-2017         34.0        5.0    NaN
```

Replacing by using mapping

```
new_df = df.replace({
        -99999: np.nan,
        'no event': 'Sunny',
    })
new_df
```

```
          day  temperature  windspeed  event
0  01-01-2017         32.0        6.0   Rain
1  01-02-2017          NaN        7.0  Sunny
2  01-03-2017         28.0        NaN   Snow
3  01-04-2017          NaN        7.0      0
4  01-05-2017         32.0        NaN   Rain
5  01-06-2017         31.0        2.0  Sunny
6  01-06-2017         34.0        5.0      0
```