# Reshaping Data

#### About the data

In this notebook, we will using daily temperature data from the National Centers for Environmental Information (NCEI) API. We will use the Global Historical Climatology Network - Daily (GHCND) data set; see the documentation here.

This data was collected for New York City for October 2018, using the Boonton 1 station (GHCNDUSC00280907). It contains:

- the daily minimum temperature (TMIN)
- · the daily maximum temperature (TMAX)
- the daily temperature at time of observation (TOBS)

Note: The NCEI is part of the National Oceanic and Atmospheric Administration (NOAA) and, as you can see from the URL for the API, this resource was created when the NCEI was called the NCDC. Should the URL for this resource change in the future, you can search for the NCEI weather API to find the updated one.

#### Setup

We need to import pandas and read in the long-format data to get started:

```
1 import pandas as pd
2 long_df = pd.read_csv(
3 'data/long_data.csv',
4 usecols=['date', 'datatype', 'value']
5 ).rename(
   columns={
      'value' : 'temp_C'
7
8 }
9 ).assign(
10 date=lambda x: pd.to_datetime(x.date),
   temp_F=lambda x: (x.temp_C * 9/5) + 32
12)
13 long_df.head()
```

	datatype	date	temp_C	temp_F	#
0	TMAX	2018-10-01	21.1	69.98	ıl.
1	TMIN	2018-10-01	8.9	48.02	
2	TOBS	2018-10-01	13.9	57.02	
3	TMAX	2018-10-02	23.9	75.02	
4	TMIN	2018-10-02	13.9	57.02	

Next steps:

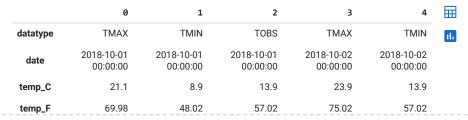


View recommended plots

# Transposing

Transposing swaps the rows and the columns. We use the T attribute to do so:

```
1 long_df.head().T
```



Next steps:

View recommended plots

## → Pivoting

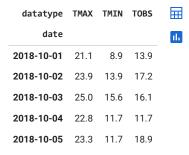
Going from long to wide format.

pivot()

We can restructure our data by picking a column to go in the index ( index ), a column whose unique values will become column names ( columns ), and the values to place in those columns ( values ). The pivot() method can be used when we don't need to perform any aggregation in addition to our restructuring (when our index is unique); if this is not the case, we need the pivot\_table() method which we will cover in future modules.

```
1 pivoted_df = long_df.pivot(
2 index='date', columns='datatype', values='temp_C'
3 )
```

4 pivoted\_df.head()



Next steps:

View recommended plots

Note there is also the pd.pivot() function which yields equivalent results:

1 pd.pivot(data=long\_df, index='date', columns='datatype', values='temp\_C').head()

datatype	TMAX	TMIN	TOBS	
date				ıl.
2018-10-01	21.1	8.9	13.9	
2018-10-02	23.9	13.9	17.2	
2018-10-03	25.0	15.6	16.1	
2018-10-04	22.8	11.7	11.7	
2018-10-05	23.3	11.7	18.9	

Now that the data is pivoted, we have wide-format data that we can grab summary statistics with:

```
1 pivoted_df.describe()
```



We can also provide multiple values to pivot on, which will result in a hierarchical index:

```
1 pivoted_df = long_df.pivot(
2  index='date', columns='datatype', values=['temp_C', 'temp_F']
3 )
4 pivoted_df.head()
```

	temp_C			temp_F			$\blacksquare$
datatype	TMAX	TMIN	TOBS	TMAX	TMIN	TOBS	ıl.
date							
2018-10-01	21.1	8.9	13.9	69.98	48.02	57.02	
2018-10-02	23.9	13.9	17.2	75.02	57.02	62.96	
2018-10-03	25.0	15.6	16.1	77.00	60.08	60.98	
2018-10-04	22.8	11.7	11.7	73.04	53.06	53.06	
2018-10-05	23.3	11.7	18.9	73.94	53.06	66.02	

With the hierarchical index, if we want to select TMIN in Fahrenheit, we will first need to select 'temp\_F' and then 'TMIN':

#### unstack()

We have been working with a single index throughout this chapter; however, we can create an index from any number of columns with set\_index(). This gives us a MultiIndex where the outermost level corresponds to the first element in the list provided to set\_index():

```
1 multi_index_df = long_df.set_index(['date', 'datatype'])
2 multi_index_df.index
```

```
( 2018-10-1/ , IMIN ),
('2018-10-17', 'TOBS'),
 ('2018-10-18', 'TMAX'),
 ('2018-10-18', 'TMIN'),
 ('2018-10-18', 'TOBS'),
 ('2018-10-19', 'TMAX'),
 ('2018-10-19', 'TMIN'),
('2018-10-19', 'TOBS'),
 ('2018-10-20', 'TMAX'),
 ('2018-10-20', 'TMIN'),
 ('2018-10-20', 'TOBS'),
 ('2018-10-21', 'TMAX'),
 ('2018-10-21', 'TMIN'),
('2018-10-21', 'TOBS'),
 ('2018-10-22', 'TMAX'),
 ('2018-10-22', 'TMIN'),
('2018-10-22', 'TOBS'),
 ('2018-10-23', 'TMAX'),
('2018-10-23', 'TMIN'),
 ('2018-10-23', 'TOBS'),
                     'TMAX'),
 ('2018-10-24', 'TMAX'), ('2018-10-24', 'TMIN'),
 ('2018-10-24', 'TOBS'),
 ('2018-10-25', 'TMAX'),
('2018-10-25', 'TMIN'),
 ('2018-10-25', 'TOBS'),
 ('2018-10-26',
                     'TMAX'),
                     'TMIN'),
 ('2018-10-26',
 ('2018-10-26', 'TOBS'),
 ('2018-10-27', 'TMAX'),
('2018-10-27', 'TMIN'),
 ('2018-10-27',
                     'TOBS'),
 ('2018-10-28', 'TMAX'),
 ('2018-10-28',
                     'TMIN'),
 ('2018-10-28', 'TOBS'),
 ('2018-10-29', 'TMAX'),
 ('2018-10-29',
                     'TMIN'),
 ('2018-10-29', 'TOBS'),
 ('2018-10-30', 'TMAX'),
 ('2018-10-30', 'TMIN'),
 ('2018-10-30', 'TOBS'),
('2018-10-31', 'TMAX'),
('2018-10-31', 'TMIN'),
('2018-10-31', 'TOBS')],
names=['date', 'datatype'])
```

Notice there are now 2 index sections of the dataframe:

#### 1 multi\_index\_df.head()

	temp_C	temp_F	
datatype			ılı
TMAX	21.1	69.98	
TMIN	8.9	48.02	
TOBS	13.9	57.02	
TMAX	23.9	75.02	
TMIN	13.9	57.02	
	TMIN TOBS TMAX	datatype  TMAX 21.1  TMIN 8.9  TOBS 13.9  TMAX 23.9	TMAX         21.1         69.98           TMIN         8.9         48.02           TOBS         13.9         57.02           TMAX         23.9         75.02

Next steps: View recommended plots

With the MultiIndex, we can no longer use pivot(). We must now use unstack(), which by default moves the innermost index onto the columns:

```
1 unstacked_df = multi_index_df.unstack()
2 unstacked_df.head()
```

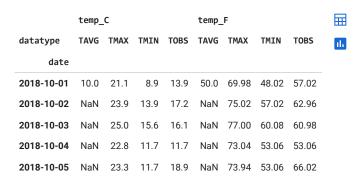
```
temp_C
                                                    ▦
                                temp_F
    datatype
               TMAX TMIN TOBS TMAX TMIN
                                            TOBS
          date
    2018-10-01
               21.1
                      8.9
                           13.9 69.98 48.02 57.02
    2018-10-02
               23.9
                     13.9
                           17.2 75.02 57.02 62.96
    2018-10-03
               25.0
                     15.6 16.1 77.00 60.08 60.98
    2018-10-04
               22.8
                    11.7
                          11.7 73.04 53.06 53.06
    2018-10-05
               23.3
                    11.7
                           18.9 73.94 53.06 66.02
Next steps:
           View recommended plots
```

The unstack() method also provides the fill\_value parameter, which let's us fill-in any NaN values that might arise from this restructuring of the data. Consider the case that we have data for the average temperature on October 1, 2018, but no other date:

```
1
1 extra_data = long_df.append(
    [{'datatype' : 'TAVG', 'date': '2018-10-01', 'temp_C': 10, 'temp_F': 50}]
3 ).set_index(['date', 'datatype']).sort_index()
5 extra_data.head(8)
    <ipython-input-18-2fd676795dfa>:1: FutureWarning: The frame.append method is deprecated
      extra_data = long_df.append(
    <ipython-input-18-2fd676795dfa>:3: FutureWarning: Inferring datetime64[ns] from data cor
      ).set_index(['date', 'datatype']).sort_index()
                          temp_C temp_F
           date datatype
                                            ıl.
     2018-10-01
                  TAVG
                             10.0
                                    50.00
                  TMAX
                             21.1
                                    69.98
                  TMIN
                              8.9
                                    48.02
                  TOBS
                             13.9
                                    57.02
     2018-10-02
                  TMAX
                             23.9
                                    75.02
                  TMIN
                             13.9
                                    57.02
                  TOBS
                                    62.96
                             17.2
     2018-10-03
                  TMAX
                             25.0
                                    77.00
```

If we use unstack() in this case, we will have NaN for the TAVG columns every day but October 1, 2018:

1 extra\_data.unstack().head()



o address this, we can pass in an appropriate fill\_value . However, we are restricted to passing in a value for this, not a strategy (like we saw with fillna()), so while -40 is definitely not be the best value, we can use it to illustrate how this works, since this is the temperature at which Fahrenheit and Celsius are equal:

```
1 extra_data.unstack(fill_value=-40).head()
```

	temp_C			temp_F					
datatype	TAVG	TMAX	TMIN	TOBS	TAVG	TMAX	TMIN	TOBS	ıl.
date									
2018-10-01	10.0	21.1	8.9	13.9	50.0	69.98	48.02	57.02	
2018-10-02	-40.0	23.9	13.9	17.2	-40.0	75.02	57.02	62.96	
2018-10-03	-40.0	25.0	15.6	16.1	-40.0	77.00	60.08	60.98	
2018-10-04	-40.0	22.8	11.7	11.7	-40.0	73.04	53.06	53.06	
2018-10-05	-40.0	23.3	11.7	18.9	-40.0	73.94	53.06	66.02	

# Melting

Going from wide to long format.

## Setup

```
1 wide_df = pd.read_csv('/content/data/wide_data.csv')
2 wide_df.head()
```

	date	TMAX	TMIN	TOBS	
0	2018-10-01	21.1	8.9	13.9	11.
1	2018-10-02	23.9	13.9	17.2	
2	2018-10-03	25.0	15.6	16.1	
3	2018-10-04	22.8	11.7	11.7	
4	2018-10-05	23.3	11.7	18.9	

Next steps:



View recommended plots

melt()

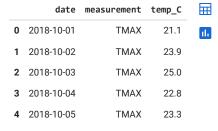
In order to go from wide format to long format, we use the melt() method. We have to specify:

- which column contains the unique identifier for each row ( date , here) to id\_vars
- the column(s) that contain the values ( TMAX , TMIN , and TOBS , here) to value\_vars

Optionally, we can also provide:-

- value\_name : what to call the column that will contain all the values once melted
- var\_name : what to call the column that will contain the names of the variables being measured

```
1 melted_df = wide_df.melt(
2 id vars='date',
3 value_vars=['TMAX', 'TMIN', 'TOBS'],
   value_name='temp_C',
   var_name='measurement'
6)
7 melted_df.head()
```



Just as we also had pd.pivot() there is a pd.melt():

```
1 pd.melt(
2  wide_df,
3  id_vars='date',
4  value_vars=['TMAX', 'TMIN', 'TOBS'],
5  value_name='temp_C',
6  var_name='measurement'
7 ).head()
8
```

	date	measurement	temp_C	
0	2018-10-01	TMAX	21.1	ıl.
1	2018-10-02	TMAX	23.9	
2	2018-10-03	TMAX	25.0	
3	2018-10-04	TMAX	22.8	
4	2018-10-05	TMAX	23.3	

stack()

Another option is stack() which will pivot the columns of the dataframe into the innermost level of a Multilndex . To illustrate this, let's set our index to be the date column:

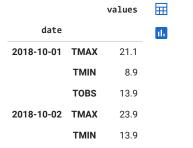
```
1 wide_df.set_index('date', inplace=True)
2 wide_df.head()
```

	TMAX	TMIN	TOBS	
date				ıl.
2018-10-01	21.1	8.9	13.9	
2018-10-02	23.9	13.9	17.2	
2018-10-03	25.0	15.6	16.1	
2018-10-04	22.8	11.7	11.7	
2018-10-05	23.3	11.7	18.9	

By running stack() now, we will create a second level in our index which will contain the column names of our dataframe (TMAX, TMIN, TOBS). This will leave us with a Series containing the values:

We can use the to\_frame() method on our Series object to turn it into a DataFrame . Since the series doesn't have a name at the moment, we will pass in the name as an argument:

1 stacked\_df = stacked\_series.to\_frame('values')
2 stacked\_df.head()



Next steps: View recommended plots

Once again, we have a MultiIndex:

## 1 stacked\_df.index

('2018-10-13', 'TMAX'),
('2018-10-13', 'TMIN'),
('2018-10-13', 'TOBS'),
('2018-10-14', 'TMAX'),
('2018-10-14', 'TMIN'),
('2018-10-14', 'TOBS'),
('2018-10-15', 'TMAX'),
('2018-10-15', 'TMIN'),
('2018-10-15', 'TOBS'),
('2018-10-16', 'TMAX'),
('2018-10-16', 'TMIN'),
('2018-10-16', 'TOBS'),
('2018-10-17', 'TMIN'),
('2018-10-17', 'TMIN'),
('2018-10-17', 'TMIN'),
('2018-10-17', 'TOBS'),
('2018-10-17', 'TOBS'),
('2018-10-18', 'TMAX'),

```
( 2018-10-30 , IMIN ),
('2018-10-30', 'TOBS'),
('2018-10-31', 'TMAX'),
('2018-10-31', 'TMIN'),
('2018-10-31', 'TOBS')],
names=['date', None])
```

Unfortunately, we don't have a name for the datatype level:

```
1 stacked_df.index.names
    FrozenList(['date', None])
```

We can use rename() to address this though:

```
1 stacked_df.index.rename(['date', 'datatype'], inplace=True)
2 stacked_df.index.names

FrozenList(['date', 'datatype'])
```

1