Reshaping Data

Setup

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

```
import pandas as pd

long_df = pd.read_csv(
    '/content/long_data.csv',
    usecols=['date', 'datatype', 'value']
).rename(
    columns={
        'value' : 'temp_C'
        }
).assign(
    date=lambda x: pd.to_datetime(x.date),
    temp_F=lambda x: (x.temp_C * 9/5) + 32
)
long_df.head()
```

	datatype	date	temp_C	temp_F
0	TMAX	2018-10-01	21.1	69.98
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

Transposing

Transposing swaps the rows and the columns. We use the T attribute to do so: 0

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

	0	1	2	3	4
datatype	TMAX	TMIN	TOBS	TMAX	TMIN
date	2018-10-01 00:00:00	2018-10-01 00:00:00	2018-10-01 00:00:00	2018-10-02 00:00:00	2018-10-02 00:00:00
temp_C	21.1	8.9	13.9	23.9	13.9
temp_F	69.98	48.02	57.02	75.02	57.02

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.

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

datatype	TMAX	TMIN	TOBS
date			
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

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

```
pd.pivot(
long_df, index=long_df.date, columns=long_df.datatype, values=long_df.temp_C
).head()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3801             try:
-> 3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:

                              7 frames
pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: Timestamp('2018-10-01 00:00:00')

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3802                 return self._engine.get_loc(casted_key)
   3803             except KeyError as err:
-> 3804                 raise KeyError(key) from err
   3805             except TypeError:
   3806                 # If we have a listlike key, _check_indexing_error will raise

KeyError: Timestamp('2018-10-01 00:00:00')
```

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

```
pivoted_df.describe()
```

datatype	TMAX	TMIN	TOBS
count	31.000000	31.000000	31.000000
mean	16.829032	7.561290	10.022581
std	5.714962	6.513252	6.596550
min	7.800000	-1.100000	-1.100000
25%	12.750000	2.500000	5.550000
50%	16.100000	6.700000	8.300000
75%	21.950000	13.600000	16.100000
max	26.700000	17.800000	21.700000

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

```
pivoted_df = long_df.pivot(
    index='date', columns='datatype', values=['temp_C', 'temp_F']
)
pivoted_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

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

```
pivoted_df['temp_F']['TMIN'].head()
```

```
date
2018-10-01    48.02
2018-10-02    57.02
2018-10-03    60.08
2018-10-04    53.06
2018-10-05    53.06
Name: TMIN, dtype: float64
```

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() :

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

```
MultiIndex([('2018-10-01', 'TMAX'),
            ('2018-10-01', 'TMIN'),
            ('2018-10-01', 'TOBS'),
            ('2018-10-02', 'TMAX'),
            ('2018-10-02', 'TMIN'),
            ('2018-10-02', 'TOBS'),
            ('2018-10-03', 'TMAX'),
            ('2018-10-03', 'TMIN'),
            ('2018-10-03', 'TOBS'),
            ('2018-10-04', 'TMAX'),
            ('2018-10-04', 'TMIN'),
            ('2018-10-04', 'TOBS'),
            ('2018-10-05', 'TMAX'),
            ('2018-10-05', 'TMIN'),
            ('2018-10-05', 'TOBS'),
            ('2018-10-06', 'TMAX'),
            ('2018-10-06', 'TMIN'),
            ('2018-10-06', 'TOBS'),
            ('2018-10-07', 'TMAX'),
            ('2018-10-07', 'TMIN'),
            ('2018-10-07', 'TOBS'),
            ('2018-10-08', 'TMAX'),
            ('2018-10-08', 'TMIN'),
            ('2018-10-08', 'TOBS'),
            ('2018-10-09', 'TMAX'),
            ('2018-10-09', 'TMIN'),
            ('2018-10-09', 'TOBS'),
            ('2018-10-10', 'TMAX'),
            ('2018-10-10', 'TMIN'),
            ('2018-10-10', 'TOBS'),
            ('2018-10-11', 'TMAX'),
            ('2018-10-11', 'TMIN'),
            ('2018-10-11', 'TOBS'),
            ('2018-10-12', 'TMAX'),
            ('2018-10-12', 'TMIN'),
            ('2018-10-12', 'TOBS'),
            ('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', 'TMAX'),
            ('2018-10-17', 'TMIN'),
            ('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'),
```

Notice there are now 2 index sections of the dataframe:

```
multi_index_df.head()
```

		temp_C	temp_F
date	**datatype**		
2018-10-01	**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

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

```
unstacked_df = multi_index_df.unstack()
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

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:

```
extra_data = long_df.append(
    [{'datatype' : 'TAVG', 'date': '2018-10-01', 'temp_C': 10, 'temp_F': 50}]
).set_index(['date', 'datatype']).sort_index()

extra_data.head(8)
```

```
<ipython-input-22-2a0bc74ba139>:1: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a fut
  extra_data = long_df.append(
<ipython-input-22-2a0bc74ba139>:3: FutureWarning: Inferring datetime64[ns] from data containing strings is deprecated and will b
  ).set_index(['date', 'datatype']).sort_index()
```

		temp_C	temp_F
date	**datatype**		
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	17.2	62.96
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:

```
extra_data.unstack().head()
```

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

To 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:

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

	temp_C				temp_F			
datatype	**TAVG**	**TMAX**	**TMIN**	**TOBS**	**TAVG**	**TMAX**	**TMIN**	**TOBS**
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

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

	date	TMAX	TMIN	TOBS
0	2018-10-01	21.1	8.9	13.9
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

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

```
melted_df = wide_df.melt(
    id_vars='date',
    value_vars=['TMAX', 'TMIN', 'TOBS'],
    value_name='temp_C',
    var_name='measurement'
)
melted_df.head()
```

	date	measurement	temp_C
0	2018-10-01	TMAX	21.1
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

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

```
pd.melt(
    wide_df,
    id_vars='date',
    value_vars=['TMAX', 'TMIN', 'TOBS'],
    value_name='temp_C',
    var_name='measurement'
).head()
```

	date	measurement	temp_C
0	2018-10-01	TMAX	21.1
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 MultiIndex . To illustrate this, let's set our index to be the date column:

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

	TMAX	TMIN	TOBS
date			
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:

```
stacked_series = wide_df.stack()
stacked_series.head()
```

```
date
2018-10-01  TMAX    21.1
            TMIN     8.9
            TOBS    13.9
2018-10-02  TMAX    23.9
            TMIN    13.9
dtype: float64
```

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:

```
stacked_df = stacked_series.to_frame('values')
stacked_df = stacked_series.to_frame('values')
stacked_df.head()
```

		values
date		
2018-10-01	**TMAX**	21.1
	TMIN	8.9
	TOBS	13.9
2018-10-02	**TMAX**	23.9
	TMIN	13.9

Once again, we have a MultiIndex :

```
stacked_df.index
```

```
MultiIndex([('2018-10-01', 'TMAX'),
            ('2018-10-01', 'TMIN'),
            ('2018-10-01', 'TOBS'),
            ('2018-10-02', 'TMAX'),
            ('2018-10-02', 'TMIN'),
            ('2018-10-02', 'TOBS'),
            ('2018-10-03', 'TMAX'),
            ('2018-10-03', 'TMIN'),
            ('2018-10-03', 'TOBS'),
            ('2018-10-04', 'TMAX'),
            ('2018-10-04', 'TMIN'),
            ('2018-10-04', 'TOBS'),
            ('2018-10-05', 'TMAX'),
            ('2018-10-05', 'TMIN'),
            ('2018-10-05', 'TOBS'),
            ('2018-10-06', 'TMAX'),
            ('2018-10-06', 'TMIN'),
            ('2018-10-06', 'TOBS'),
            ('2018-10-07', 'TMAX'),
            ('2018-10-07', 'TMIN'),
            ('2018-10-07', 'TOBS'),
            ('2018-10-08', 'TMAX'),
            ('2018-10-08', 'TMIN'),
            ('2018-10-08', 'TOBS'),
            ('2018-10-09', 'TMAX'),
            ('2018-10-09', 'TMIN'),
            ('2018-10-09', 'TOBS'),
            ('2018-10-10', 'TMAX'),
            ('2018-10-10', 'TMIN'),
            ('2018-10-10', 'TOBS'),
            ('2018-10-11', 'TMAX'),
            ('2018-10-11', 'TMIN'),
            ('2018-10-11', 'TOBS'),
            ('2018-10-12', 'TMAX'),
            ('2018-10-12', 'TMIN'),
            ('2018-10-12', 'TOBS'),
            ('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', 'TMAX'),
            ('2018-10-17', 'TMIN'),
            ('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'),
```

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

```
stacked_df.index.names
```

```
FrozenList(['date', None])
```

We can use rename() to address this though:

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

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