

Database-style Operations on Dataframes

About the data

In this notebook, we will use daily weather data that was taken from the [National Centers for Environmental Information \(NCEI\) API](#). The data collection notebook contains the process that was followed to collect the data.

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.

Background on the data

Data meanings:

- PRCP : precipitation in millimeters
- SNOW : snowfall in millimeters
- SNWD : snow depth in millimeters
- TMAX : maximum daily temperature in Celsius
- TMIN : minimum daily temperature in Celsius
- TOBS : temperature at time of observation in Celsius
- WESF : water equivalent of snow in millimeters

Setup

```
In [8]: import pandas as pd
weather = pd.read_csv("nyc_weather_2018.csv")
weather.head()
```

```
Out[8]:
```

	attributes	datatype	date	station	value
0	„N,	PRCP	2018-01-01T00:00:00	GHCND:US1CTFR0039	0.0
1	„N,	PRCP	2018-01-01T00:00:00	GHCND:US1NJBG0015	0.0
2	„N,	SNOW	2018-01-01T00:00:00	GHCND:US1NJBG0015	0.0
3	„N,	PRCP	2018-01-01T00:00:00	GHCND:US1NJBG0017	0.0
4	„N,	SNOW	2018-01-01T00:00:00	GHCND:US1NJBG0017	0.0

Querying DataFrames

The query() method is an easier way of filtering based on some criteria. For example, we can use it to find all entries where snow was recorded:

```
In [11]: snow = weather.query("datatype == 'SNOW' and value > 0")
snow.head(5)
```

```
Out[11]:
```

	attributes	datatype	date	station	value
124	„N,	SNOW	2018-01-01T00:00:00	GHCND:US1NYWC0019	25.0
723	„N,	SNOW	2018-01-04T00:00:00	GHCND:US1NJBG0015	229.0
726	„N,	SNOW	2018-01-04T00:00:00	GHCND:US1NJBG0017	10.0
730	„N,	SNOW	2018-01-04T00:00:00	GHCND:US1NJBG0018	46.0
737	„N,	SNOW	2018-01-04T00:00:00	GHCND:US1NJS0018	10.0

This is equivalent to querying the data/weather.db SQLite database for SELECT * FROM weather WHERE datatype == "SNOW" AND value > 0 :

```
In [13]: import sqlite3

with sqlite3.connect("weather.db") as connection:
    snow_from_db = pd.read_sql(
        "SELECT * FROM weather WHERE datatype == 'SNOW' AND value > 0",
        connection
    )

snow.reset_index().drop(columns="index").equals(snow_from_db)
```

```
Out[13]: True
```

Note this is also equivalent to creating Boolean masks:

```
In [15]: weather[(weather.datatype == "SNOW") & (weather.value > 0)].equals(snow)
```

```
Out[15]: True
```

Merging DataFrames

We have data for many different stations each day; however, we don't know what the stations are just their IDs. We can join the data in the data/weather_stations.csv file which contains information from the stations endpoint of the NCEI API. Consult the weather_data_collection.ipynb notebook to see how this was collected. It looks like this:

```
In [18]: station = pd.read_csv("weather_stations.csv")
station.head()
```

```
Out[18]:
```

	id	name	latitude	longitude	elevation
0	GHCND:US1CTFR0022	STAMFORD 2.6 SSW, CT US	41.0641	-73.5770	36.6
1	GHCND:US1CTFR0039	STAMFORD 4.2 S, CT US	41.0378	-73.5682	6.4
2	GHCND:US1NJBG0001	BERGENFIELD 0.3 SW, NJ US	40.9213	-74.0020	20.1
3	GHCND:US1NJBG0002	SADDLE BROOK TWP 0.6 E, NJ US	40.9027	-74.0834	16.8
4	GHCND:US1NJBG0003	TENAFLY 1.3 W, NJ US	40.9147	-73.9775	21.6

As a reminder, the weather data looks like this:

```
In [20]: weather
```

```
Out[20]:
```

	attributes	datatype	date	station	value
0	„N,	PRCP	2018-01-01T00:00:00	GHCND:US1CTFR0039	0.0
1	„N,	PRCP	2018-01-01T00:00:00	GHCND:US1NJBG0015	0.0
2	„N,	SNOW	2018-01-01T00:00:00	GHCND:US1NJBG0015	0.0
3	„N,	PRCP	2018-01-01T00:00:00	GHCND:US1NJBG0017	0.0
4	„N,	SNOW	2018-01-01T00:00:00	GHCND:US1NJBG0017	0.0
...
80251	„W,	WDF5	2018-12-31T00:00:00	GHCND:USW00094789	130.0
80252	„W,	WSF2	2018-12-31T00:00:00	GHCND:USW00094789	9.8
80253	„W,	WSF5	2018-12-31T00:00:00	GHCND:USW00094789	12.5
80254	„W,	WT01	2018-12-31T00:00:00	GHCND:USW00094789	1.0
80255	„W,	WT02	2018-12-31T00:00:00	GHCND:USW00094789	1.0

80256 rows × 5 columns

We can join our data by matching up the station_info.id column with the weather.station column. Before doing that though, let's see how many unique values we have:

```
In [22]: station.id.describe()
```

```
Out[22]: count          262
unique          262
top    GHCND:US1CTFR0022
freq              1
Name: id, dtype: object
```

While station_info has one row per station, the weather dataframe has many entries per station. Notice it also has fewer uniques:

```
In [24]: weather.station.describe()
```

```
Out[24]: count          80256  
unique           109  
top      GHCND:USW00094789  
freq          4270  
Name: station, dtype: object
```

When working with joins, it is important to keep an eye on the row count. Some join types will lead to data loss:

```
In [26]: station.shape[0], weather.shape[0]
```

```
Out[26]: (262, 80256)
```

```
In [27]: def grc(*dfs):  
         return [df.shape[0] for df in dfs]  
grc(station, weather)
```

```
Out[27]: [262, 80256]
```

The `map()` function is more efficient than list comprehensions. We can couple this with `getattr()` to grab any attribute for multiple dataframes

```
In [29]: def getinf(attr, *dfs):  
         return list(map(lambda x: getattr(x, attr), dfs))  
  
getinf("shape", station, weather)
```

```
Out[29]: [(262, 5), (80256, 5)]
```

By default `merge()` performs an inner join. We simply specify the columns to use for the join. The left dataframe is the one we call `merge()` on, and the right one is passed in as an argument:

```
In [31]: injoin = weather.merge(station, left_on="station", right_on="id")  
injoin.sample(5, random_state=0)
```

Out[31]:

	attributes	datatype	date	station	value	i
27422	„W,	WDF5	2018-04-29T00:00:00	GHCND:USW00094741	310.0	GHCND:USW0009474
19317	„W,	WSF5	2018-03-24T00:00:00	GHCND:USW00094728	8.5	GHCND:USW0009472
13778	„W,	PGTM	2018-03-01T00:00:00	GHCND:USW00054743	2351.0	GHCND:USW0005474
39633	„W,	WSF2	2018-06-25T00:00:00	GHCND:USW00094789	11.2	GHCND:USW0009478
51025	„7,0700	TMAX	2018-08-19T00:00:00	GHCND:USC00280907	30.0	GHCND:USC0028090

We can remove the duplication of information in the station and id columns by renaming one of them before the merge and then simply using on :

In [33]: `weather.merge(station.rename(dict(id="station"), axis=1), on="station").sample(5, r`

Out[33]:

	attributes	datatype	date	station	value	name	la
27422	„W,	WDF5	2018-04-29T00:00:00	GHCND:USW00094741	310.0	TETERBORO AIRPORT, NJ US	40.
19317	„W,	WSF5	2018-03-24T00:00:00	GHCND:USW00094728	8.5	NY CITY CENTRAL PARK, NY US	40.
13778	„W,	PGTM	2018-03-01T00:00:00	GHCND:USW00054743	2351.0	CALDWELL ESSEX CO AIRPORT, NJ US	40.
39633	„W,	WSF2	2018-06-25T00:00:00	GHCND:USW00094789	11.2	JFK INTERNATIONAL AIRPORT, NY US	40.
51025	„7,0700	TMAX	2018-08-19T00:00:00	GHCND:USC00280907	30.0	BOONTON 1 SE, NJ US	40.

We are losing stations that don't have weather observations associated with them, if we don't want to lose these rows, we perform a right or left join instead of the inner join:

In [35]: `left_join = station.merge(weather, left_on="id", right_on="station", how="left")
right_join = weather.merge(station, left_on="station", right_on="id", how="right")
right_join.tail()`

Out[35]:

	attributes	datatype	date	station	value	id
80404	„W,	WDF5	2018-12-31T00:00:00	GHCND:USW00094789	130.0	GHCND:USW00094789
80405	„W,	WSF2	2018-12-31T00:00:00	GHCND:USW00094789	9.8	GHCND:USW00094789
80406	„W,	WSF5	2018-12-31T00:00:00	GHCND:USW00094789	12.5	GHCND:USW00094789
80407	„W,	WT01	2018-12-31T00:00:00	GHCND:USW00094789	1.0	GHCND:USW00094789
80408	„W,	WT02	2018-12-31T00:00:00	GHCND:USW00094789	1.0	GHCND:USW00094789

In [36]: `left_join.sort_index(axis=1).sort_values(["date", "station"]).reset_index().drop(columns=["date", "station"], inplace=True)`
`right_join.sort_index(axis=1).sort_values(["date", "station"]).reset_index().drop(columns=["date", "station"], inplace=True)`

Out[36]: True

Note we have additional rows in the left and right joins because we kept all the stations that didn't have weather observations:

In [38]: `getinf("shape", injoin, left_join, right_join)`

Out[38]: [(80256, 10), (80409, 10), (80409, 10)]

If we query the station information for stations that have NY in their name, believing that to be all the stations that record weather data for NYC and perform an outer join, we can see where the mismatches occur:

In [40]: `outer_join = weather.merge(
 station[station.name.str.contains("NY")],
 left_on="station", right_on="id", how="outer", indicator=True
)
pd.concat([outer_join.sample(4, random_state=0), outer_join[outer_join.station.isna()]])`

Out[40]:

	attributes	datatype	date	station	value	i
17259	„N,	SNOW	2018-07-20T00:00:00	GHCND:US1NJMS0075	0.0	NaN
76178	„W,	AWND	2018-01-12T00:00:00	GHCND:USW00094789	7.2	GHCND:USW0009478
73410	T„W,2400	PRCP	2018-03-16T00:00:00	GHCND:USW00094745	0.0	GHCND:USW0009474
74822	„W,	PGTM	2018-08-20T00:00:00	GHCND:USW00094745	1641.0	GHCND:USW0009474
14954	NaN	NaN	NaN	NaN	NaN	GHCND:US1NJMS003
14955	NaN	NaN	NaN	NaN	NaN	GHCND:US1NJMS003

These joins are equivalent to their SQL counterparts. Below is the inner join. Note that to use equals() you will have to do some manipulation of the dataframes to line them up:

```
In [42]: import sqlite3 as sq3

with sq3.connect("weather.db") as connection:
    inner_join_from_db = pd.read_sql("SELECT * FROM weather JOIN stations ON weather.

inner_join_from_db.shape == injoin.shape
```

Out[42]: True

Revisit the dirty data from the previous module.

```
In [44]: dirty = pd.read_csv("dirty_data2.csv", index_col="date").drop_duplicates().drop(col
dirty.head()
```

Out[44]:

	station	PRCP	SNOW	TMAX	TMIN	TOBS	WESF	inclement_w
date								
2018-01-01T00:00:00	?	0.0	0.0	5505.0	-40.0	NaN	NaN	
2018-01-02T00:00:00	GHCND:USC00280907	0.0	0.0	-8.3	-16.1	-12.2	NaN	
2018-01-03T00:00:00	GHCND:USC00280907	0.0	0.0	-4.4	-13.9	-13.3	NaN	
2018-01-04T00:00:00	?	20.6	229.0	5505.0	-40.0	NaN	19.3	
2018-01-05T00:00:00	?	0.3	NaN	5505.0	-40.0	NaN	NaN	

We need to create two dataframes for the join. We will drop some unnecessary columns as well for easier viewing:

```
In [46]: valid_station = dirty.query('station != "?").copy().drop(columns=["WESF", "station"])
sta_with_wesf = dirty.query('station == "?").copy().drop(columns=["station", "TOBS"])
```

Our column for the join is the index in both dataframes, so we must specify left_index and right_index :

```
In [48]: valid_station.merge(sta_with_wesf, left_index=True, right_index=True).query("WESF >
```

Out[48]:

	PRCP_x	SNOW_x	TMAX	TMIN	TOBS	inclement_weather_x	PRCP_y	SNOW_y
date								
2018-01-30T00:00:00	0.0	0.0	6.7	-1.7	-0.6	False	1.5	1.0
2018-03-08T00:00:00	48.8	NaN	1.1	-0.6	1.1	False	28.4	NaN
2018-03-13T00:00:00	4.1	51.0	5.6	-3.9	0.0	True	3.0	1.0
2018-03-21T00:00:00	0.0	0.0	2.8	-2.8	0.6	False	6.6	11.0
2018-04-02T00:00:00	9.1	127.0	12.8	-1.1	-1.1	True	14.0	15.0


The columns that existed in both dataframes, but didn't form part of the join got suffixes added to their names: _x_ for columns from the left dataframe and y for columns from the right dataframe. We can customize this with the suffixes argument:


```
In [50]: valid_station.merge(sta_with_wesf, left_index=True, right_index=True, suffixes=("",
```

```
Out[50]:
```

	PRCP	SNOW	TMAX	TMIN	TOBS	inclement_weather	PRCP_?	SNOW_?	W
date									

2018-01-30T00:00:00	0.0	0.0	6.7	-1.7	-0.6		False	1.5	13.0
2018-03-08T00:00:00	48.8	NaN	1.1	-0.6	1.1		False	28.4	NaN
2018-03-13T00:00:00	4.1	51.0	5.6	-3.9	0.0		True	3.0	13.0
2018-03-21T00:00:00	0.0	0.0	2.8	-2.8	0.6		False	6.6	114.0
2018-04-02T00:00:00	9.1	127.0	12.8	-1.1	-1.1		True	14.0	152.0




Since we are joining on the index, an easier way is to use the `join()` method instead of `merge()`. Note that the suffix parameter is now `lsuffix` for the left dataframe's suffix and `rsuffix` for the right one's:

```
In [52]: valid_station.join(sta_with_wesf, rsuffix="_?").query("WESF > 0").head()
```

```
Out[52]:
```

	PRCP	SNOW	TMAX	TMIN	TOBS	inclement_weather	PRCP_?	SNOW_?	W
date									

2018-01-30T00:00:00	0.0	0.0	6.7	-1.7	-0.6		False	1.5	13.0
2018-03-08T00:00:00	48.8	NaN	1.1	-0.6	1.1		False	28.4	NaN
2018-03-13T00:00:00	4.1	51.0	5.6	-3.9	0.0		True	3.0	13.0
2018-03-21T00:00:00	0.0	0.0	2.8	-2.8	0.6		False	6.6	114.0
2018-04-02T00:00:00	9.1	127.0	12.8	-1.1	-1.1		True	14.0	152.0



Joins can be very resource-intensive, so it's a good idea to figure out what type of join you need using set operations before trying the join itself. The pandas set operations are performed on the index, so whichever columns we will be joining on will need to be the index. Let's go back to the weather and station_info dataframes and set the station ID columns as the index:

```
In [54]: weather.set_index("station", inplace=True)
station.set_index("id", inplace=True)
```

The intersection will tell us the stations that are present in both dataframes. The result will be the index when performing an inner join:

```
In [56]: weather.index.intersection(station.index)
```

```
Out[56]: Index(['GHCND:US1CTFR0039', 'GHCND:US1NJBG0015', 'GHCND:US1NJBG0017',
               'GHCND:US1NJBG0018', 'GHCND:US1NJBG0023', 'GHCND:US1NJBG0030',
               'GHCND:US1NJBG0039', 'GHCND:US1NJBG0044', 'GHCND:US1NJES0018',
               'GHCND:US1NJES0024',
               ...
               'GHCND:US1NJMS0047', 'GHCND:US1NYSF0083', 'GHCND:US1NYNY0074',
               'GHCND:US1NJPS0018', 'GHCND:US1NJBG0037', 'GHCND:USC00284987',
               'GHCND:US1NJES0031', 'GHCND:US1NJMD0086', 'GHCND:US1NJMS0097',
               'GHCND:US1NJMN0081'],
              dtype='object', length=109)
```

```
In [57]: weather.index.difference(station.index)
```

```
Out[57]: Index([], dtype='object')
```

We lose 114 stations from the station_info dataframe, however:

```
In [59]: station.index.difference(weather.index)
```

```
Out[59]: Index(['GHCND:US1CTFR0022', 'GHCND:US1NJBG0001', 'GHCND:US1NJBG0002',
               'GHCND:US1NJBG0005', 'GHCND:US1NJBG0006', 'GHCND:US1NJBG0008',
               'GHCND:US1NJBG0011', 'GHCND:US1NJBG0012', 'GHCND:US1NJBG0013',
               'GHCND:US1NJBG0020',
               ...
               'GHCND:USC00308322', 'GHCND:USC00308749', 'GHCND:USC00308946',
               'GHCND:USC00309117', 'GHCND:USC00309270', 'GHCND:USC00309400',
               'GHCND:USC00309466', 'GHCND:USC00309576', 'GHCND:USW00014708',
               'GHCND:USW00014786'],
              dtype='object', length=153)
```

The symmetric difference will tell us what gets lost from both sides. It is the combination of the set difference in both directions: $216 + 114 = 330$ which was station_info shape was

```
In [61]: ny_in_name = station[station.name.str.contains('NY')]

ny_in_name.index.difference(weather.index).shape[0]\
+ weather.index.difference(ny_in_name.index).shape[0]\
== weather.index.symmetric_difference(ny_in_name.index).shape[0]
```

```
Out[61]: True
```

The union will show us everything that will be present after a full outer join. Note that since these are sets (which don't allow duplicates by definition), we must pass unique entries for union:

```
In [65]: weather.index.unique().union(station.index)
```

```
Out[65]: Index(['GHCND:US1CTFR0022', 'GHCND:US1CTFR0039', 'GHCND:US1NJBG0001',  
              'GHCND:US1NJBG0002', 'GHCND:US1NJBG0003', 'GHCND:US1NJBG0005',  
              'GHCND:US1NJBG0006', 'GHCND:US1NJBG0008', 'GHCND:US1NJBG0010',  
              'GHCND:US1NJBG0011',  
              ...  
              'GHCND:USW00014708', 'GHCND:USW00014732', 'GHCND:USW00014734',  
              'GHCND:USW00014786', 'GHCND:USW00054743', 'GHCND:USW00054787',  
              'GHCND:USW00094728', 'GHCND:USW00094741', 'GHCND:USW00094745',  
              'GHCND:USW00094789'],  
            dtype='object', length=262)
```

Note that the symmetric difference is actually the union of the set differences:

```
In [68]: ny_in_name = station[station.name.str.contains('NY')]  
ny_in_name.index.difference(weather.index).union(weather.index.difference(ny_in_name.index))  
weather.index.symmetric_difference(ny_in_name.index)  
)
```

```
Out[68]: True
```

```
In [ ]:
```