Database-style Operations on Dataframes

About the data In this notebook, we will using 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

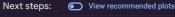
Data meanings:

- 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

```
1 import pandas as p
2 wthr = p.read_csv('/content/nycweather2k18 8.1.csv')
3 wthr
```



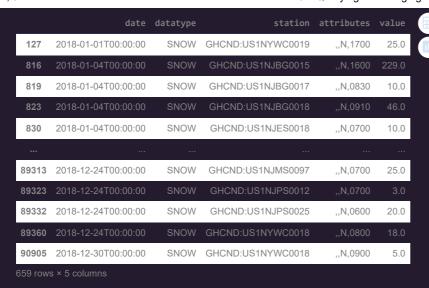


Querying DataFrames

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:

```
1 snwdat = wthr.query('datatype == "SNOW" and value >0')
```

2 snwdat



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

```
1 import sqlite3 as sq3
3 with sq3.connect('/content/weather 8.1.db') as connection:
     snwdat fdb = p.read sql(
         'SELECT * FROM weather WHERE datatype == "SNOW" and value > 0',
8 snwdat.reset_index().drop(columns='index').equals(snwdat_fdb)
```

True

1 wthr[(wthr.datatype == 'SNOW') & (wthr.value > 0)].equals(snwdat)

True

Merging DataFrames

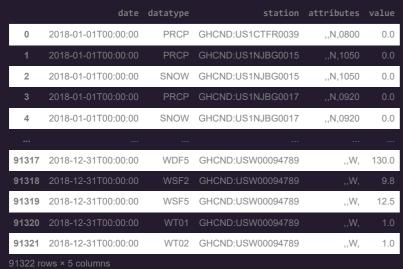
We have data for many different stations each day, however, we don't know what the stations are just their IDs. We can join the data in the data/weather_stations.csv file which contains information from the stations endpoint of the NCEI API. Consult the weather_data_collection.ipynb notebook to see how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

1 ststinf = p.read_csv('/content/weather_stations 8.1.csv') 2 ststinf id GHCND:US1CTFR0022 STAMFORD 2.6 SSW, CT US 41.064100 -73.577000 36.6 2 GHCND:US1NJBG0001 BERGENFIELD 0.3 SW, NJ US 40.921298 -74.001983 20.1 4 GHCND:US1NJBG0003 TENAFLY 1.3 W. NJ US 40.914670 -73.977500 21.6 FARMINGDALE REPUBLIC 315 GHCND:USW00054787 40.734430 -73.416370 22.8 AIRPORT, NY US GHCND:USW00094741 TETERBORO AIRPORT, NJ US 40.858980 -74.056160 0.8 WESTCHESTER CO AIRPORT, NY JFK INTERNATIONAL AIRPORT, NY 319 GHCND:USW00094789 40.639150 -73.763900 27

Next steps: View recommended plots

As a reminder, the weather data looks like this:

1 wthr



Next steps: View recommended plots

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:

```
1 ststinf.id.describe()
                               320
```

unique 320 GHCND:US1CTFR0022 top Name: id, dtvpe: object

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

```
1 wthr.station.describe()
```

91322 count unique 114 GHCND: USW00014734 top freq 6744 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:

```
1 ststinf.shape[0], wthr.shape[0]
```

(320, 91322)

```
1 def grc(*dfs):
2 return [df.shape[0] for df in dfs]
3 grc(ststinf, wthr)
```

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

```
1 def getinf(attr, *dfs):
2 return list(map(lambda x: getattr(x, attr), dfs))
3 getinf('shape', ststinf,wthr)
    [(320, 5), (91322, 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:

```
1 injoin = wthr.merge(ststinf,left_on='station', right_on='id')
2 injoin.sample(5, random_state=0)
```



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:

1 wthr.merge(ststinf.rename(dict(id='station'),axis=1),on='station').sample(5, random_state=0)

	date	datatype	station	attributes	value	name	latitude
24218	2018-03- 19T00:00:00	PRCP	GHCND:US1NYNS0036	,,N,0615	0.0	SYOSSET 2.0 SSW, NY US	40.787036
39269	2018-06- 30T00:00:00	SNWD	GHCND:USC00289187	,,7,0700	0.0	WANAQUE RAYMOND DAM, NJ US	41.044400
82228	2018-11- 17T00:00:00	WDF2	GHCND:USW00094789	,,W,	270.0	JFK INTERNATIONAL AIRPORT, NY US	40.639150
21949	2018-06- 01T00:00:00	PRCP	GHCND:US1NYNS0007	,,N,0700	3.0	FLORAL PARK 0.4 W, NY US	40.723000
4)

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:

```
1 lejoin = ststinf.merge(wthr, left_on='id',right_on='station', how='left')
2 rijoin = wthr.merge(ststinf, left_on='station',right_on='id', how='right')
3
...
```

	date	datatype	station	attributes	value	id	
91523	2018-12- 31T00:00:00	WDF5	GHCND:USW00094789	,,W,	130.0	GHCND:USW00094789	INTERN AIRI
91524	2018-12- 31T00:00:00	WSF2	GHCND:USW00094789		9.8	GHCND:USW00094789	INTERN AIRI
91525	2018-12- 31T00:00:00	WSF5	GHCND:USW00094789	,,W,	12.5	GHCND:USW00094789	INTERN AIRI
91526	2018-12- 31T00:00:00	WT01	GHCND:USW00094789	,,W,	1.0	GHCND:USW00094789	INTERN AIRI
91527	2018-12- 31T00:00:00	WT02	GHCND:USW00094789	,,W,	1.0	GHCND:USW00094789	INTERN AIRI

```
1 lejoin.sort_index(axis=1).sort_values(['date','station']).reset_index().drop(columns='index').equals(
2     rijoin.sort_index(axis=1).sort_values(['date','station']).reset_index().drop(columns='index')
3 )
True
```

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

```
1 getinf('shape', injoin, lejoin, rijoin)
[(91322, 10), (91528, 10), (91528, 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:

<ipython-input-60-fccc62d2ea39>:5: FutureWarning: The frame.append method is deprecated and will I
oujoin.sample(4, random_state=0).append(oujoin[oujoin.station.isna()].head(2))

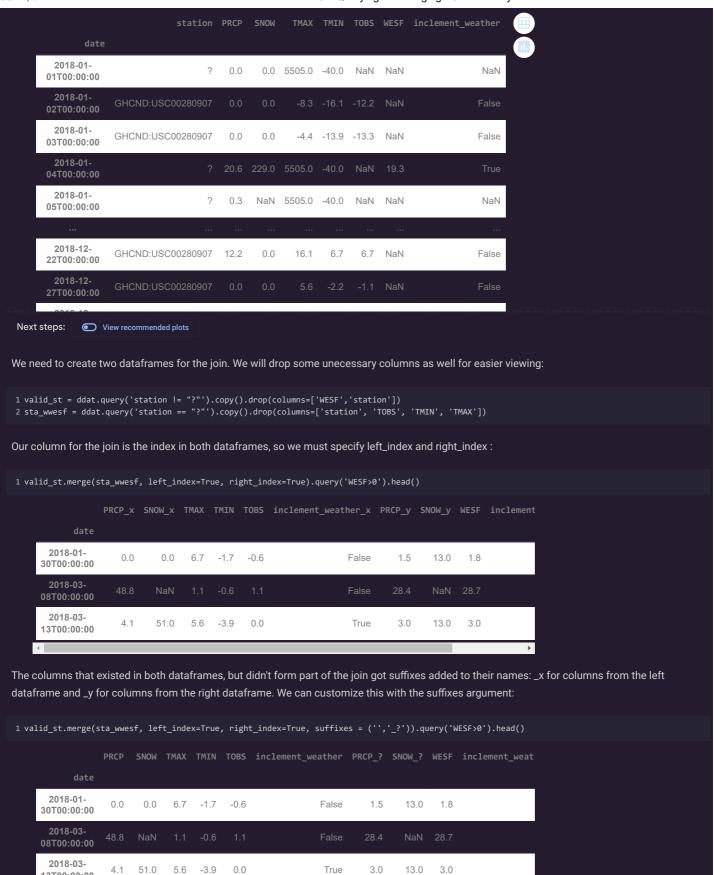
	date	datatype	station	attributes	value	id	
72786	2018-12- 03T00:00:00	WSF5	GHCND:USW00094741	,,W,	14.8	NaN	
75733	2018-10- 29T00:00:00	WSF5	GHCND:USW00094745		12.5	GHCND:USW00094745	WEST(CO.
65872	2018-05- 01T00:00:00	ADPT	GHCND:USW00094728	,,W,	-11.0	GHCND:USW00094728	CENTR
80308	2018-08- 04T00:00:00	WSF5	GHCND:USW00094789		11.2	GHCND:USW00094789	INTERN AIR
91322	NaN	NaN	NaN	NaN	NaN	GHCND:US1NJHD0018	KE NN
91323	NaN	NaN	NaN	NaN	NaN	GHCND:US1NJMS0036	PAF TR TWF

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:

```
1 import sqlite3 as sq3
2
3 with sq3.connect('<u>/content/weather</u> 8.1.db') as connection:
4   ijfdb = p.read_sql('SELECT * FROM weather JOIN stations ON weather.station == stations.id',connection)
5
6  ijfdb.shape == injoin.shape
True
```

Revisit the dirty data from the previous module.

```
1 ddat = p.read_csv('/content/dirty_data.csv', index_col='date').drop_duplicates().drop(columns='SNWD')
2
3 ddat
```



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 Isuffix for the left dataframe's suffix and rsuffix for the right one's:

True

```
1 valid_st.join(sta_wwesf, rsuffix='_?').query('WESF >0').head()
```

13T00:00:00

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:

```
1 wthr.set_index('station', inplace=True)
2 ststinf.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:

We lose 153 stations from the station_info dataframe, however:

The symmetric difference will tell us what gets lost from both sides. It is the combination of the set difference in both directions:

```
1 nyiname = ststinf[ststinf.name.str.contains('NY')]
2
3 nyiname.index.difference(wthr.index).shape[0]\
4 + wthr.index.difference(nyiname.index).shape[0]\
5 == wthr.index.symmetric_difference(nyiname.index).shape[0]
```

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: