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

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

	date	datatype	station	attributes	value
0	2018-01-01T00:00:00	PRCP	GHCND:US1CTFR0039	,,N,0800	0.0
1	2018-01-01T00:00:00	PRCP	GHCND:US1NJBG0015	,,N,1050	0.0
2	2018-01-01T00:00:00	SNOW	GHCND:US1NJBG0015	,,N,1050	0.0
3	2018-01-01T00:00:00	PRCP	GHCND:US1NJBG0017	,,N,0920	0.0
4	2018-01-01T00:00:00	SNOW	GHCND:US1NJBG0017	,,N,0920	0.0
...	...	...	...	...	...
91317	2018-12-31T00:00:00	WDF5	GHCND:USW00094789	,,W,	130.0
91318	2018-12-31T00:00:00	WSF2	GHCND:USW00094789	,,W,	9.8
91319	2018-12-31T00:00:00	WSF5	GHCND:USW00094789	,,W,	12.5
91320	2018-12-31T00:00:00	WT01	GHCND:USW00094789	,,W,	1.0
91321	2018-12-31T00:00:00	WT02	GHCND:USW00094789	,,W,	1.0

91322 rows × 5 columns

Next steps: View recommended plots

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:

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

	date	datatype	station	attributes	value
127	2018-01-01T00:00:00	SNOW	GHCND:US1NYWC0019	,,N,1700	25.0
816	2018-01-04T00:00:00	SNOW	GHCND:US1NJBG0015	,,N,1600	229.0
819	2018-01-04T00:00:00	SNOW	GHCND:US1NJBG0017	,,N,0830	10.0
823	2018-01-04T00:00:00	SNOW	GHCND:US1NJBG0018	,,N,0910	46.0
830	2018-01-04T00:00:00	SNOW	GHCND:US1NJES0018	,,N,0700	10.0
...	...	...	...	...	...
89313	2018-12-24T00:00:00	SNOW	GHCND:US1NJMS0097	,,N,0700	25.0
89323	2018-12-24T00:00:00	SNOW	GHCND:US1NJPS0012	,,N,0700	3.0
89332	2018-12-24T00:00:00	SNOW	GHCND:US1NJPS0025	,,N,0600	20.0
89360	2018-12-24T00:00:00	SNOW	GHCND:US1NYWC0018	,,N,0800	18.0
90905	2018-12-30T00:00:00	SNOW	GHCND:US1NYWC0018	,,N,0900	5.0

659 rows × 5 columns

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

```
import sqlite3 as sq3

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

```
True
```

```
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:

```
ststinf = p.read_csv('/content/weather_stations 8.1.csv')
ststinf
```

	id	name	latitude	longitude	elevation
0	GHCND:US1CTFR0022	STAMFORD 2.6 SSW, CT US	41.064100	-73.577000	36.6
1	GHCND:US1CTFR0039	STAMFORD 4.2 S, CT US	41.037788	-73.568176	6.4
2	GHCND:US1NJBG0001	BERGENFIELD 0.3 SW, NJ US	40.921298	-74.001983	20.1
3	GHCND:US1NJBG0002	SADDLE BROOK TWP 0.6 E, NJ US	40.902694	-74.083358	16.8
4	GHCND:US1NJBG0003	TENAFLY 1.3 W, NJ US	40.914670	-73.977500	21.6
...	...	...	...	...	...
315	GHCND:USW00054787	FARMINGDALE REPUBLIC AIRPORT, NY US	40.734430	-73.416370	22.8
316	GHCND:USW00094728	NY CITY CENTRAL PARK, NY US	40.778980	-73.969250	42.7
317	GHCND:USW00094741	TETERBORO AIRPORT, NJ US	40.858980	-74.056160	0.8
318	GHCND:USW00094745	WESTCHESTER CO AIRPORT, NY US	41.062360	-73.704540	112.9
319	GHCND:USW00094789	JFK INTERNATIONAL AIRPORT, NY US	40.639150	-73.763900	2.7

Next steps: View recommended plots

As a reminder, the weather data looks like this:

```
wthr
```

	date	datatype	station	attributes	value
0	2018-01-01T00:00:00	PRCP	GHCND:US1CTFR0039	,,N,0800	0.0
1	2018-01-01T00:00:00	PRCP	GHCND:US1NJBG0015	,,N,1050	0.0
2	2018-01-01T00:00:00	SNOW	GHCND:US1NJBG0015	,,N,1050	0.0
3	2018-01-01T00:00:00	PRCP	GHCND:US1NJBG0017	,,N,0920	0.0
4	2018-01-01T00:00:00	SNOW	GHCND:US1NJBG0017	,,N,0920	0.0
...	...	...	...	...	...
91317	2018-12-31T00:00:00	WDF5	GHCND:USW00094789	,,W,	130.0
91318	2018-12-31T00:00:00	WSF2	GHCND:USW00094789	,,W,	9.8
91319	2018-12-31T00:00:00	WSF5	GHCND:USW00094789	,,W,	12.5
91320	2018-12-31T00:00:00	WT01	GHCND:USW00094789	,,W,	1.0
91321	2018-12-31T00:00:00	WT02	GHCND:USW00094789	,,W,	1.0

91322 rows × 5 columns

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:

```
ststinf.id.describe()
```

```
count                  320
unique                 320
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:

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

```
count                91322
unique                 114
top       GHCND:USW00014734
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:

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

```
(320, 91322)
```

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

```
[320, 91322]
```

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

```
def getinf(attr, *dfs):
  return list(map(lambda x: getattr(x, attr), dfs))
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:

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

	date	datatype	station	attributes	value	id	
24218	2018-03-19T00:00:00	PRCP	GHCND:US1NYNS0036	,,N,0615	0.0	GHCND:US1NYNS0036	S' S
39269	2018-06-30T00:00:00	SNWD	GHCND:USC00289187	,,7,0700	0.0	GHCND:USC00289187	RAYI
82228	2018-11-17T00:00:00	WDF2	GHCND:USW00094789	,,W,	270.0	GHCND:USW00094789	INTE A
21949	2018-06-01T00:00:00	PRCP	GHCND:US1NYNS0007	,,N,0700	3.0	GHCND:US1NYNS0007	FL 0
53218	2018-05-28T00:00:00	ASLP	GHCND:USW00014734	,,W,	10176.0	GHCND:USW00014734	INTE AIRF

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

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
4 rijoin.tail()
```

	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	,,W,	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

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

```
True
```

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

```
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:

```
oujoin = wthr.merge(
    ststinf[ststinf.name.str.contains('NY')],
    left_on='station',right_on='id', how='outer', indicator=True
)
oujoin.sample(4, random_state=0).append(oujoin[oujoin.station.isna()].head(2))
```

```
<ipython-input-60-fccc62d2ea39>:5: FutureWarning: The frame.append method is deprecated and will l
  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	,,W,	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	,,W,	11.2	GHCND:USW00094789	INTERI 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:

```
import sqlite3 as sq3

with sq3.connect('/content/weather 8.1.db') as connection:
  ijfdb = p.read_sql('SELECT * FROM weather JOIN stations ON weather.station == stations.id',connection)

ijfdb.shape == injoin.shape
```

```
True
```

Revisit the dirty data from the previous module.

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

ddat
```

date	station	PRCP	SNOW	TMAX	TMIN	TOBS	WESF	inclement_weather
2018-01-01T00:00:00	?	0.0	0.0	5505.0	-40.0	NaN	NaN	NaN
2018-01-02T00:00:00	GHCND:USC00280907	0.0	0.0	-8.3	-16.1	-12.2	NaN	False
2018-01-03T00:00:00	GHCND:USC00280907	0.0	0.0	-4.4	-13.9	-13.3	NaN	False
2018-01-04T00:00:00	?	20.6	229.0	5505.0	-40.0	NaN	19.3	True
2018-01-05T00:00:00	?	0.3	NaN	5505.0	-40.0	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...
2018-12-22T00:00:00	GHCND:USC00280907	12.2	0.0	16.1	6.7	6.7	NaN	False
2018-12-27T00:00:00	GHCND:USC00280907	0.0	0.0	5.6	-2.2	-1.1	NaN	False

Next steps: View recommended plots

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

```
1 valid_st = ddat.query('station != "?"').copy().drop(columns=['WESF','station'])
2 sta_wwesf = ddat.query('station == "?"').copy().drop(columns=['station', 'TOBS', 'TMIN', 'TMAX'])
```

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

```
1 valid_st.merge(sta_wwesf, left_index=True, right_index=True).query('WESF>0').head()
```

date	PRCP_x	SNOW_x	TMAX	TMIN	TOBS	inclement_weather_x	PRCP_y	SNOW_y	WESF	inclement
2018-01-30T00:00:00	0.0	0.0	6.7	-1.7	-0.6	False	1.5	13.0	1.8	
2018-03-08T00:00:00	48.8	NaN	1.1	-0.6	1.1	False	28.4	NaN	28.7	
2018-03-13T00:00:00	4.1	51.0	5.6	-3.9	0.0	True	3.0	13.0	3.0	

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:

```
1 valid_st.merge(sta_wwesf, left_index=True, right_index=True, suffixes = ('','_?')).query('WESF>0').head()
```

date	PRCP	SNOW	TMAX	TMIN	TOBS	inclement_weather	PRCP_?	SNOW_?	WESF	inclement_weat
2018-01-30T00:00:00	0.0	0.0	6.7	-1.7	-0.6	False	1.5	13.0	1.8	
2018-03-08T00:00:00	48.8	NaN	1.1	-0.6	1.1	False	28.4	NaN	28.7	
2018-03-13T00:00:00	4.1	51.0	5.6	-3.9	0.0	True	3.0	13.0	3.0	

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:

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

date	PRCP	SNOW	TMAX	TMIN	TOBS	inclement_weather	PRCP_?	SNOW_?	WESF	inclement_weat
2018-01-30T00:00:00	0.0	0.0	6.7	-1.7	-0.6	False	1.5	13.0	1.8	
2018-03-	48.8	NaN	1.1	-0.6	1.1	False	28.4	NaN	28.7	

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:

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

```
wthr.index.intersection(ststinf.index)
```

```
Index(['GHCND:US1CTFR0039', 'GHCND:US1NJBG0015', 'GHCND:US1NJBG0017',
       'GHCND:US1NJBG0018', 'GHCND:US1NJBG0023', 'GHCND:US1NJBG0030',
       'GHCND:US1NJBG0039', 'GHCND:US1NJBG0044', 'GHCND:US1NJES0018',
       'GHCND:US1NJES0024',
       ...
       'GHCND:USC00284987', 'GHCND:US1NJES0031', 'GHCND:US1NJES0029',
       'GHCND:US1NJMD0086', 'GHCND:US1NJMS0097', 'GHCND:US1NJMN0081',
       'GHCND:US1NJMD0088', 'GHCND:US1NJES0033', 'GHCND:US1NJES0040',
       'GHCND:US1NYQN0029'],
      dtype='object', length=114)
```

```
wthr.index.difference(ststinf.index)
```

```
Index([], dtype='object')
```

We lose 153 stations from the station_info dataframe, however:

```
ststinf.index.difference(wthr.index)
```

```
Index(['GHCND:US1CTFR0022', 'GHCND:US1NJBG0001', 'GHCND:US1NJBG0002',
       'GHCND:US1NJBG0005', 'GHCND:US1NJBG0006', 'GHCND:US1NJBG0008',
       'GHCND:US1NJBG0011', 'GHCND:US1NJBG0012', 'GHCND:US1NJBG0013',
       'GHCND:US1NJBG0020',
       ...
       'GHCND:USC00308749', 'GHCND:USC00308946', 'GHCND:USC00309117',
       'GHCND:USC00309270', 'GHCND:USC00309400', 'GHCND:USC00309466',
       'GHCND:USC00309576', 'GHCND:USC00309580', 'GHCND:USW00014708',
       'GHCND:USW00014786'],
      dtype='object', length=206)
```

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

```
nyiname = ststinf[ststinf.name.str.contains('NY')]

nyiname.index.difference(wthr.index).shape[0]\
+ wthr.index.difference(nyiname.index).shape[0]\
== wthr.index.symmetric_difference(nyiname.index).shape[0]
```

```
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:

```
wthr.index.unique().union(ststinf.index)
```

```
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',
```