Database-style Operations on Dataframes

Name: Corpuz, Micki Laurren B.

Section: CPE22S3

Instructor: Engr. Roman Richard

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

In [86]:
```
import pandas as pd
weather = pd.read_csv('nyc_weather_2018.csv')
weather.head()
```

Out[86]:

	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

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 [87]:
```
snowdata = weather.query('datatype == "SNOW" and value > 0')
snowdata.head(5)
```

Out[87]:

	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

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

In [88]:
```
import sqlite3

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

snowdata.reset_index().drop(columns='index').equals(snow_data_from_db)
```

Out[88]: True

Note this is also equivalent to creating Boolean masks:

In [89]:
```
weather[(weather.datatype == 'SNOW') & (weather.value > 0)].equals(snowdata)
```

Out[89]: 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 [90]:
```
station_info = pd.read_csv('weather_stations.csv')
station_info.head()
```

Out[90]:

	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

As a reminder, the weather data looks like this:

In [91]:
```
weather
```

Out[91]:

	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
...	...	...	...	...	...
90528	2018-12-31T00:00:00	WDF5	GHCND:USW00094789	,,W,	130.0
90529	2018-12-31T00:00:00	WSF2	GHCND:USW00094789	,,W,	9.8
90530	2018-12-31T00:00:00	WSF5	GHCND:USW00094789	,,W,	12.5
90531	2018-12-31T00:00:00	WT01	GHCND:USW00094789	,,W,	1.0
90532	2018-12-31T00:00:00	WT02	GHCND:USW00094789	,,W,	1.0

90533 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 [92]:
```
station_info.id.describe()
```

Out[92]:
```
count                 330
unique                330
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 [93]:
```
weather.station.describe()
```

Out[93]:
```
count                90533
unique                 114
top       GHCND:USW00014734
freq                  6688
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 [94]:
```
station_info.shape[0], weather.shape[0]
```

Out[94]:
```
(330, 90533)
```

In [95]:
```
def grc(*dfs):
  return [df.shape[0] for df in dfs]
grc(station_info, weather)
```

Out[95]:
```
[330, 90533]
```

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

In [96]:
```
def getinf(attr, *dfs):
  return list(map(lambda x: getattr(x, attr), dfs))
getinf('shape', station_info,weather)
```

Out[96]:
```
[(330, 5), (90533, 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 [97]:
```
injoin = weather.merge(station_info,left_on='station', right_on='id')
injoin.sample(5, random_state=0)
```

Out[97]:

	date	datatype	station	attributes	value	i
66921	2018-09-27T00:00:00	TOBS	GHCND:USC00284987	,,7,0630	17.8	GHCND:USC0028498
8091	2018-01-31T00:00:00	PRCP	GHCND:US1NYRC0002	T,,N,0700	0.0	GHCND:US1NYRC000
85623	2018-12-11T00:00:00	SNWD	GHCND:US1NJMN0048	,,N,0700	0.0	GHCND:US1NJMN004
66887	2018-09-27T00:00:00	PRCP	GHCND:US1NYSF0062	,,N,0900	0.8	GHCND:US1NYSF006
52731	2018-07-29T00:00:00	TMAX	GHCND:USW00054787	,,W,	27.8	GHCND:USW0005478

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 [98]:

```
weather.merge(station_info.rename(dict(id='station'),axis=1),on='station').sample(5
```

Out[98]:

	date	datatype	station	attributes	value	name	lati
66921	2018-09-27T00:00:00	TOBS	GHCND:USC00284987	,,7,0630	17.8	LONG BRANCH OAKHURST, NJ US	40.29
8091	2018-01-31T00:00:00	PRCP	GHCND:US1NYRC0002	T,,N,0700	0.0	STATEN ISLAND 1.4 SE, NY US	40.56
85623	2018-12-11T00:00:00	SNWD	GHCND:US1NJMN0048	,,N,0700	0.0	LONG BRANCH 0.5 W, NJ US	40.29
66887	2018-09-27T00:00:00	PRCP	GHCND:US1NYSF0062	,,N,0900	0.8	COPIAGUE 0.4 ENE, NY US	40.67
52731	2018-07-29T00:00:00	TMAX	GHCND:USW00054787	,,W,	27.8	FARMINGDALE REPUBLIC AIRPORT, NY US	40.73

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 [99]:
```
left_join = station_info.merge(weather, left_on='id',right_on='station', how='left'
right_join = weather.merge(station_info, left_on='station',right_on='id', how='righ

right_join.tail()
```

Out[99]:

	date	datatype	station	attributes	value	id
90744	2018-12-31T00:00:00	WDF5	GHCND:USW00094789	,,W,	130.0	GHCND:USW0009478
90745	2018-12-31T00:00:00	WSF2	GHCND:USW00094789	,,W,	9.8	GHCND:USW0009478
90746	2018-12-31T00:00:00	WSF5	GHCND:USW00094789	,,W,	12.5	GHCND:USW0009478
90747	2018-12-31T00:00:00	WT01	GHCND:USW00094789	,,W,	1.0	GHCND:USW0009478
90748	2018-12-31T00:00:00	WT02	GHCND:USW00094789	,,W,	1.0	GHCND:USW0009478

In [100…
```
left_join.sort_index(axis=1).sort_values(['date','station']).reset_index().drop(col
    right_join.sort_index(axis=1).sort_values(['date','station']).reset_index().dro
)
```

Out[100… 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 [101…
```
getinf('shape', injoin, left_join, right_join)
```

Out[101… [(90533, 10), (90749, 10), (90749, 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 [102…
```
outer_join = weather.merge(
    station_info[station_info.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[102…

	date	datatype	station	attributes	value	id
19271	2018-04-12T00:00:00	SNWD	GHCND:US1NJMS0089	,,N,0800	0.0	GHCND:US1NJMS0089
79459	2018-08-14T00:00:00	TMIN	GHCND:USW00094741	,,W,	21.1	NaN
48418	2018-03-08T00:00:00	WT04	GHCND:USC00301309	,,7,	1.0	GHCND:USC00301309
89873	2018-11-19T00:00:00	TMIN	GHCND:USW00094789	,,W,2400	3.3	GHCND:USW00094789
7099	NaN	NaN	NaN	NaN	NaN	GHCND:US1NJHD0018
15029	NaN	NaN	NaN	NaN	NaN	GHCND:US1NJMS0036

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 [103…

```
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[103…

```
True
```

Revisit the dirty data from the previous module.

In [104…

```
dirty_data = pd.read_csv('dirty_data2.csv', index_col='date').drop_duplicates().dro
dirty_data.head()
```

Out[104…

date	station	PRCP	SNOW	TMAX	TMIN	TOBS	WESF	inclement_w
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 unecessary columns as well for easier viewing:

In [105…

```
valid_station = dirty_data.query('station != "?"').copy().drop(columns=['WESF','sta
sta_with_wesf = dirty_data.query('station == "?"').copy().drop(columns=['station',
```

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

In [106…

```
valid_station.merge(sta_with_wesf, left_index=True, right_index=True).query('WESF>0
```

Out[106…

date	PRCP_x	SNOW_x	TMAX	TMIN	TOBS	inclement_weather_x	PRCP_y	SNOW
2018-01-30T00:00:00	0.0	0.0	6.7	-1.7	-0.6	False	1.5	1:
2018-03-08T00:00:00	48.8	NaN	1.1	-0.6	1.1	False	28.4	N.
2018-03-13T00:00:00	4.1	51.0	5.6	-3.9	0.0	True	3.0	1:
2018-03-21T00:00:00	0.0	0.0	2.8	-2.8	0.6	False	6.6	11
2018-04-02T00:00:00	9.1	127.0	12.8	-1.1	-1.1	True	14.0	15.

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 [107…
```
valid_station.merge(sta_with_wesf, left_index=True, right_index=True, suffixes = ('
```

Out[107…

date	PRCP	SNOW	TMAX	TMIN	TOBS	inclement_weather	PRCP_?	SNOW_?	W
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 [108…
```
valid_station.join(sta_with_wesf, rsuffix='_?').query('WESF >0').head()
```

Out[108…

date	PRCP	SNOW	TMAX	TMIN	TOBS	inclement_weather	PRCP_?	SNOW_?	W
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 [109…
```
weather.set_index('station', inplace=True)
station_info.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 [110…
```
weather.index.intersection(station_info.index)
```

Out[110…
```
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)
```

In [111…
```
weather.index.difference(station_info.index)
```

Out[111…
```
Index([], dtype='object')
```

We lose 114 stations from the station_info dataframe, however:

In [114…
```
station_info.index.difference(weather.index)
```

Out[114…
```
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=216)
```

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 [115…
```
ny_in_name = station_info[station_info.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[115…
```
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 [116…

```
weather.index.unique().union(station_info.index)
```

Out[116…

```
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=330)
```

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

In [117…

```
ny_in_name = station_info[station_info.name.str.contains('NY')]
ny_in_name.index.difference(weather.index).union(weather.index.difference(ny_in_nam
weather.index.symmetric_difference(ny_in_name.index)
)
```

Out[117…

```
True
```