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 pd
weather = pd.read_csv('/content/nyc_weather_2018.csv')
weather.head()
```

	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

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.

```
snow_data = weather.query('datatype == "SNOW" and value > 0') # Filters the datatype with snow and value higher than 0
snow_data.head()
```

	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:US1NJES0018	10.0

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

```
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
  )
snow_data.reset_index().drop(columns='index').equals(snow_data_from_db)
 # when reseting index it will keep the old index so it must be dropped see below
```

```
True
```

```
reset = snow_data.reset_index()
reset.head()
# the new index was created on the most left then the index column was the old default index that was selected during query()
```

	index	attributes	datatype	date	station	value
0	124	,,N,	SNOW	2018-01-01T00:00:00	GHCND:US1NYWC0019	25.0
1	723	,,N,	SNOW	2018-01-04T00:00:00	GHCND:US1NJBG0015	229.0
2	726	,,N,	SNOW	2018-01-04T00:00:00	GHCND:US1NJBG0017	10.0
3	730	,,N,	SNOW	2018-01-04T00:00:00	GHCND:US1NJBG0018	46.0
4	737	,,N,	SNOW	2018-01-04T00:00:00	GHCND:US1NJES0018	10.0

```
# Here is the dataframe when you drop the index column
reset = snow_data.reset_index().drop(columns = 'index')
print(reset.head())
print(len(reset))
```

```
  attributes datatype                 date            station  value
0       ,,N,     SNOW  2018-01-01T00:00:00  GHCND:US1NYWC0019   25.0
1       ,,N,     SNOW  2018-01-04T00:00:00  GHCND:US1NJBG0015  229.0
2       ,,N,     SNOW  2018-01-04T00:00:00  GHCND:US1NJBG0017   10.0
3       ,,N,     SNOW  2018-01-04T00:00:00  GHCND:US1NJBG0018   46.0
4       ,,N,     SNOW  2018-01-04T00:00:00  GHCND:US1NJES0018   10.0
667
```

```
# Now comparing them by their length and entity would return true
print(snow_data_from_db.head())
print(len(snow_data_from_db))
```

```
  attributes datatype                 date            station  value
0       ,,N,     SNOW  2018-01-01T00:00:00  GHCND:US1NYWC0019   25.0
1       ,,N,     SNOW  2018-01-04T00:00:00  GHCND:US1NJBG0015  229.0
2       ,,N,     SNOW  2018-01-04T00:00:00  GHCND:US1NJBG0017   10.0
3       ,,N,     SNOW  2018-01-04T00:00:00  GHCND:US1NJBG0018   46.0
4       ,,N,     SNOW  2018-01-04T00:00:00  GHCND:US1NJES0018   10.0
667
```

Note this is also equivalent to creating Boolean masks:

```
weather[(weather.datatype == 'SNOW') & (weather.value > 0)].equals(snow_data)
# same results as the snow_data Query() so returns true
```

```
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:

```
station_info = pd.read_csv('/content/weather_stations.csv')
station_info.head()
```

	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

Next steps: View recommended plots

As a reminder, the weather data looks like this:

```
weather.head()
```

	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

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:

```
station_info.id.describe()
```

```
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:

```
station_info.shape[0], weather.shape[0]
```

```
(262, 80256)
```

Since we will be doing this often, it makes more sense to write a function:

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

```
[262, 80256]
```

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

```
def get_info(attr, *dfs):
  return list(map(lambda x: getattr(x, attr), dfs))
get_info('shape', station_info, weather)
# returns rows and column in a tuple
```

```
[(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:

```
inner_join = weather.merge(station_info, left_on='station', right_on='id')
inner_join.sample(5, random_state=0)
# This would display the merged data on each row, but I think tries to find the most similar identity  which in this case was the station a
```

	attributes	datatype	date	station	value	
27422	,,N,	PRCP	2018-01-23T00:00:00	GHCND:US1NYSF0061	2.3	GHCND:US1NYSF0(
19317	T,,N,	PRCP	2018-08-10T00:00:00	GHCND:US1NJUN0014	0.0	GHCND:US1NJUN0(
13778	,,N,	WESF	2018-02-18T00:00:00	GHCND:US1NJMS0089	19.6	GHCND:US1NJMS0(

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 :

```
print(dict(id='station'))
```

```
{'id': 'station'}
```

```
weather.merge(station_info.rename(dict(id='station'), axis=1), on='station').sample(5, random_state=0)
# dict is a must since .rename function accepts mapper which is a dict or axis(1) == column name
# when you merge the columns with the same name, it will make it as one and now the outputs was merged data of weather and station info
```

	attributes	datatype	date	station	value	name	l
27422	,,N,	PRCP	2018-01-23T00:00:00	GHCND:US1NYSF0061	2.3	CENTERPORT 0.9 SW, NY US	
19317	T,,N,	PRCP	2018-08-10T00:00:00	GHCND:US1NJUN0014	0.0	WESTFIELD 0.6 NE, NJ US	
13778	,,N,	WESF	2018-02-18T00:00:00	GHCND:US1NJMS0089	19.6	PARSIPPANY TROY HILLS TWP 1.3, NJ US	

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:

```
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='right')
right_join.tail()
# results to having Nan values so that entries wont end up waste
```

	attributes	datatype	date	station	value	
80404	,,W,	WDF5	2018-12-31T00:00:00	GHCND:USW00094789	130.0	GHCND:USW000947
80405	,,W,	WSF2	2018-12-31T00:00:00	GHCND:USW00094789	9.8	GHCND:USW000947

```
print(station_info.shape[0], weather.shape[0])
print(len(left_join))
print(len(right_join)) # Both retain data
print(len(inner_join)) # merges data
```

```
262 80256
80409
80409
80256
```

The left and right join as we performed above are equivalent because the side that we kept the rows without matches was the same in both cases:

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

```
True
```

```
r = right_join.sort_index(axis=1).sort_values(['date', 'station']).head()
l = left_join.sort_index(axis=1).sort_values(['date', 'station']).head()
r.equals(l)
# doesnt need to reset index and drop the old index if would just need to compare
```

```
True
```

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

```
get_info('shape', inner_join, left_join, right_join)
```

```
[(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:

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

```
<ipython-input-105-81b63e73e04e>:5: FutureWarning: The frame.append method is deprecate
  outer_join.sample(4, random_state=0).append(outer_join[outer_join.station.isna()].hea
```

	attributes	datatype	date	station	value	
17259	,,N,	PRCP	2018-05-15T00:00:00	GHCND:US1NJPS0022	0.3	N
76178	,,N,	PRCP	2018-05-19T00:00:00	GHCND:US1NJPS0015	8.1	N
73410	,,N,	MDPR	2018-08-05T00:00:00	GHCND:US1NYNS0018	12.2	GHCND:US1NYNS0
			2018-04-			

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

with sqlite3.connect('weather.db') as connection:
  inner_join_from_db = pd.read_sql(
      'SELECT * FROM weather JOIN stations ON weather.station == stations.id',
      connection
      )
inner_join_from_db.shape == inner_join.shape
# the weather.db has two table which is the weather and stations
# inner join has already the attributes of joining two datas which is also identical to the weather.db tables
# so when they are compared '==' it sets true
```

```
True
```

Revisit the dirty data from the previous module.

```
dirty_data = pd.read_csv(
    '/content/dirty_data2.csv', index_col='date'
).drop_duplicates().drop(columns='SNWD')
dirty_data.head()
```

	station	PRCP	SNOW	TMAX	TMIN	TOBS	WESF	inclement_weathe
date								
2018-01-01T00:00:00	?	0.0	0.0	5505.0	-40.0	NaN	NaN	Na
2018-01-02T00:00:00	GHCND:USC00280907	0.0	0.0	-8.3	-16.1	-12.2	NaN	Fals
2018-01-03T00:00:00	GHCND:USC00280907	0.0	0.0	-4.4	-13.9	-13.3	NaN	Fals

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:

```
valid_station = dirty_data.query('station != "?"').copy().drop(columns=['WESF', 'station']) # gets everything on station that is not  ?
station_with_wesf = dirty_data.query('station == "?"').copy().drop(columns=['station', 'TOBS', 'TMIN', 'TMAX']) # gets station that is equa
```

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

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

	PRCP_x	SNOW_x	TMAX	TMIN	TOBS	inclement_weather_x	PRCP_y	SNOW_y	WESF
date									
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:

```
valid_station.merge(
station_with_wesf, left_index=True, right_index=True, suffixes=('', '_?')
).query('WESF > 0').head()
```

	PRCP	SNOW	TMAX	TMIN	TOBS	inclement_weather	PRCP_?	SNOW_?	WESF	incl
date										
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:

```
valid_station.join(station_with_wesf, rsuffix='_?').query('WESF > 0').head()
```

	PRCP	SNOW	TMAX	TMIN	TOBS	inclement_weather	PRCP_?	SNOW_?	WESF	incl
date										
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	

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:

```
weather.set_index('station', inplace=True)
station_info.set_index('id', inplace=True)
# return no copy of old index but replaces the old
```

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

```
weather.index.intersection(station_info.index)
```

```
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)
```

The set difference will tell us what we lose from each side. When performing an inner join, we lose nothing from the weather dataframe:

```
weather.index.difference(station_info.index)
```

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

We lose 153 stations from the station_info dataframe, however:

```
station_info.index.difference(weather.index)
```

```
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:

```
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]
```

```
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:

```
weather.index.unique().union(station_info.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',
       'GHCND:USW00094789'],
      dtype='object', length=262)
```

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

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

```
True
```