

## Database-style Operations on Dataframes

### 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() # Display the first few rows of the DataFrame
```

	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	

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:

```
# Filter the weather DataFrame to include only rows where datatype is "SNOW" and value is greater than 0
snow_data = weather.query('datatype == "SNOW" and value > 0')
snow_data.head() # Display the first few rows of the DataFrame
```

	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:US1NJS0018	„N,0700	10.0	

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

```
import sqlite3

# Create connection to the SQLite database
with sqlite3.connect('/content/weather.db') as connection:
    # Read snow data from the weather table where datatype is 'SNOW' and value is greater than 0
    snow_data_from_db = pd.read_sql(
        'SELECT * FROM weather WHERE datatype == "SNOW" AND value > 0',
        connection
    )
# Reset index and drop redundant index column before comparison
snow_data.reset_index().drop(columns='index').equals(snow_data_from_db)

True
```

Note this is also equivalent to creating Boolean masks:

```
# Check if the subset of 'weather' dataframe containing only snow data (datatype == 'SNOW')
#and non-zero snow values (value > 0) is equal to another dataframe 'snow_data'.
weather[(weather.datatype == 'SNOW') & (weather.value > 0)].equals(snow_data)

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() # Display the first few rows of the DataFrame
```

	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	

Next steps: [View recommended plots](#)

As a reminder, the weather data looks like this:

```
weather.head() # Display the first few rows of the data
```

	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	

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() # to generate descriptive statistics for the 'id' column in station_info dataframe
```

```
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:

```
weather.station.describe() # to generate descriptive statistics for the weather station data
```

```
count          92313
unique          114
top    GHCND:USW00014734
freq          6817
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:

```
# Get the number of rows in the station_info and weather DataFrame
station_info.shape[0], weather.shape[0]
```

```
(320, 92313)
```

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] # Return a list containing the row counts of each dataframe.
get_row_count(station_info, weather) # Call the functions
```

```
[320, 92313]
```

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))
# Use the map function to iterate over the dataframes
# Convert the result into a list
get_info('shape', station_info, weather) # Call the functions with the attribute 'shape'
```

```
[(320, 5), (92313, 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:

```
# Merge the 'weather' df with the 'station_info' df using an inner join
inner_join = weather.merge(station_info, left_on='station', right_on='id')
# Display a random sample of 5 rows from the merged df
inner_join.sample(5, random_state=0)
```

	date	datatype	station	attributes	value		id	name	latitude
50262	2018-11-20T00:00:00	PRCP	GHCND:USW00014732	T,,W,2400	0.0	GHCND:USW00014732		LAGUARDIA AIRPORT, NY US	40.77945
88343	2018-10-06T00:00:00	SNOW	GHCND:US1NJBG0003	,,N,0730	0.0	GHCND:US1NJBG0003		TENAFLY 1.3 W, NJ US	40.91467
62350	2018-05-14T00:00:00	TMIN	GHCND:USW00054787	,,W,	10.6	GHCND:USW00054787		FARMINGDALE REPUBLIC AIRPORT, NY US	40.73443
40536	2018-01-04T00:00:00	TMAX	GHCND:USC00301309	,,7,0700	-1.1	GHCND:USC00301309		CENTERPORT, NY US	40.88345
33639	2018-07-25T00:00:00	TMAX	GHCND:USC00280907	,,7,0700	30.0	GHCND:USC00280907		BOONTON 1 SE, NJ US	40.89174

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:

```
# Merge weather data with station information based on the 'station' column
# Renaming the 'id' column of station_info to 'station' for alignment with weather data
# Sampling 5 random rows from the merged dataset with a random state of 0
weather.merge(station_info.rename(dict(id='station'), axis=1), on='station').sample(5, random_state=0)
```

	date	datatype	station	attributes	value	name	latitude	longitude	elevation
50262	2018-11-20T00:00:00	PRCP	GHCND:USW00014732	T,,W,2400	0.0	LAGUARDIA AIRPORT, NY US	40.77945	-73.88027	3.0
88343	2018-10-06T00:00:00	SNOW	GHCND:US1NJBG0003	,,N,0730	0.0	TENAFLY 1.3 W, NJ US	40.91467	-73.97750	21.6
62350	2018-05-14T00:00:00	TMIN	GHCND:USW00054787	,,W,	10.6	FARMINGDALE REPUBLIC AIRPORT, NY US	40.73443	-73.41637	22.8
40536	2018-01-04T00:00:00	TMAX	GHCND:USC00301309	,,7,0700	-1.1	CENTERPORT, NY US	40.88345	-73.37309	9.1
33639	2018-07-25T00:00:00	TMAX	GHCND:USC00280907	,,7,0700	30.0	BOONTON 1 SE, NJ US	40.89174	-74.39635	85.3

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:

```
# get left join between station_info and weather dataframes based on 'id' column in station_info and 'station' column in weather
left_join = station_info.merge(weather, left_on='id', right_on='station', how='left')
# get right join between weather and station_info dataframes based on 'station' column in weather and 'id' column in station_info
right_join = weather.merge(station_info, left_on='station', right_on='id', how='right')
right_join.tail() # Display the last few rows
```

	date	datatype	station	attributes	value	id	name	latitude
92514	2018-12-31T00:00:00	WDF5	GHCND:USW00094789	,,W,	130.0	GHCND:USW00094789	JFK INTERNATIONAL AIRPORT, NY US	40.6391
92515	2018-12-31T00:00:00	WSF2	GHCND:USW00094789	,,W,	9.8	GHCND:USW00094789	JFK INTERNATIONAL AIRPORT, NY US	40.6391
92516	2018-12-31T00:00:00	WSF5	GHCND:USW00094789	,,W,	12.5	GHCND:USW00094789	JFK INTERNATIONAL AIRPORT, NY US	40.6391
92517	2018-12-31T00:00:00	WT01	GHCND:USW00094789	,,W,	1.0	GHCND:USW00094789	JFK INTERNATIONAL AIRPORT, NY US	40.6391
92518	2018-12-31T00:00:00	WT02	GHCND:USW00094789	,,W,	1.0	GHCND:USW00094789	JFK INTERNATIONAL AIRPORT, NY US	40.6391

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')
)
# Perform right join and left join, sort columns by index, sort values by 'date' and 'station', reset index, drop 'index'
# Check if the resulting dataframes are equal
```

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)
# this represents different types of joins and returns information about them based on the specified attribute.

[(92313, 10), (92519, 10), (92519, 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:

```
# Merge weather data with station information for stations in NY
outer_join = weather.merge(
    station_info[station_info.name.str.contains('NY')], # Filter stations located in NY
    left_on='station', # Column to join from the left DataFrame (weather)
    right_on='id', # Column to join from the right DataFrame (station_info)
    how='outer', # Perform an outer join to include all rows from both DataFrames
    indicator=True # Add a column to indicate the source of each row (weather, station_info, or both)
)

# Randomly sample 4 rows from the merged df with a specified random state and append any rows where station is NaN
outer_join.sample(4, random_state=0).append(outer_join[outer_join.station.isna()].head(2))

<ipython-input-17-b7ca4817982f>:11: FutureWarning: The frame.append method is deprecated and will be removed from
outer_join.sample(4, random_state=0).append(outer_join[outer_join.station.isna()].head(2))
```

	date	datatype	station	attributes	value	id	name	latitud
<b>34015</b>	2018-09-26T00:00:00	SNOW	GHCND:USC00280907	„7,	0.0	NaN	NaN	NaN
<b>75473</b>	2018-07-01T00:00:00	TMAX	GHCND:USW00094745	„W,2400	34.4	GHCND:USW00094745	WESTCHESTER CO AIRPORT, NY US	41.06231
<b>2245</b>	2018-02-22T00:00:00	SNOW	GHCND:US1NJBG0023	„N,0800	0.0	NaN	NaN	NaN
<b>43582</b>	2018-06-14T00:00:00	SNOW	GHCND:USC00308577	„7,	0.0	GHCND:USC00308577	SYOSSET, NY US	40.82161
<b>92313</b>	NaN	NaN	NaN	NaN	NaN	GHCND:US1NJHD0018	KEARNY 1.7 NNW, NJ US	40.77431
<b>92314</b>	NaN	NaN	NaN	NaN	NaN	GHCND:US1NJMS0036	PARSIPPANY TROY HILLS TWP 2.1 E, NJ US	40.86561

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('/content/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

True
```

Revisit the dirty data from the previous module.

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

	station	PRCP	SNOW	TMAX	TMIN	TOBS	WESF	inclement_weather
date								
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

Next steps: [View recommended plots](#)

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

```
# Filter out rows with valid station values (not '?') and create a copy
valid_station = dirty_data.query('station != "?"').copy().drop(columns=['WESF', 'station'])

# Filter out rows with station value as '?' and drop unnecessary columns
station_with_wesf = dirty_data.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:

```
valid_station.merge(
    station_with_wesf, left_index=True, right_index=True # Merge two dataframes, valid_station and station_with_wesf,
).query('WESF > 0').head() # Filter the merged dataframe to retain only rows where the 'WESF' column is greater th
```

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

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()
# Merge and filter to get rows where 'WESF' column is greater than 0, then display the first few rows
```

	PRCP	SNOW	TMAX	TMIN	TOBS	inclement_weather	PRCP_?	SNOW_?	WESF	inclement_weather_?
date										
2018-01-30T00:00:00	0.0	0.0	6.7	-1.7	-0.6	False	1.5	13.0	1.8	True
2018-03-08T00:00:00	48.8	NaN	1.1	-0.6	1.1	False	28.4	NaN	28.7	NaN
2018-03-13T00:00:00	4.1	51.0	5.6	-3.9	0.0	True	3.0	13.0	3.0	True
2018-03-14T00:00:00	0.0	0.0	0.0	0.0	0.0	False	0.0	11.0	0.0	True

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()
# Joining valid_station with station_with_wesf using the indices, filtering rows where 'WESF' column is greater than 0
```

	PRCP	SNOW	TMAX	TMIN	TOBS	inclement_weather	PRCP_?	SNOW_?	WESF	inclement_weather_?
date										
2018-01-30T00:00:00	0.0	0.0	6.7	-1.7	-0.6	False	1.5	13.0	1.8	True
2018-03-08T00:00:00	48.8	NaN	1.1	-0.6	1.1	False	28.4	NaN	28.7	NaN
2018-03-13T00:00:00	4.1	51.0	5.6	-3.9	0.0	True	3.0	13.0	3.0	True
2018-03-14T00:00:00	0.0	0.0	0.0	0.0	0.0	False	0.0	11.0	0.0	True

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) # Setting the 'station' column as the index for the weather df
station_info.set_index('id', inplace=True) # Setting the 'id' column as the index for the station_info df
```

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) # Finding the intersection of indices between the weather and station_info df

Index(['GHCND:US1CTFR0039', 'GHCND:US1NJBG0015', 'GHCND:US1NJBG0017',
      'GHCND:US1NJBG0018', 'GHCND:US1NJBG0023', 'GHCND:US1NJBG0030',
      'GHCND:US1NJBG0039', 'GHCND:US1NJBG0044', 'GHCND:US1NJBG0018',
      'GHCND:US1NJBG0024',
      ...,
      'GHCND:USC00284987', 'GHCND:US1NJBG0031', 'GHCND:US1NJBG0029',
      'GHCND:US1NJBG0086', 'GHCND:US1NJBG0097', 'GHCND:US1NJBG0081',
      'GHCND:US1NJBG0088', 'GHCND:US1NJBG0033', 'GHCND:US1NJBG0040',
      'GHCND:US1NJBG0029'],
      dtype='object', length=114)
```

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) # Finding the indices in the weather DataFrame that are not present in the station_info df
```



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

We lose 153 stations from the station\_info dataframe, however:

```
station_info.index.difference(weather.index) # Finding the indices in the station_info DataFrame that are not present
```

```
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: