Collecting weather data from an API

About the data

in this notebook, we will be collecting daily weather data dffrom the National Center for Environmental Information(NCEI) API. We will use the Global Historical Climatology Network - daily (GHCND) data set.

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 resoiurce was tcreated when NCEU was called the NCDC, Should the URL for this resource change in the future, you can search for the NCEI weahter API to fund the updated one.

Using the NCEI API

Paste your token below.

```
import requests
def make_request(endpoint, payload=None):
  return requests.get(
    f'https://www.ncdc.noaa.gov/cdo-web/api/v2/{endpoint}', # This is an API call function that passes an Endpoint which an API that you wa
    headers={
      'token': 'SsFLqwUFuYIHQRPEeWFPQjuqZLCySibB' # Ur personal Token, every account registered will have unique token
    },
    params=payload # Parameter to append; optional
  )
```

Collect All Data Points for 2018 In NYC (Various Stations)

We can make a loop to query for all the data points one day at a time. Here we create a list of all the results:

```
import datetime
from IPython import display # for updating the cell dynamically
current = datetime.date(2018, 1, 1)
end = datetime.date(2018, 2, 2)  # I Changed the End time because it is taking too long; changed it to one month interval
results = []
while current < end: # Basically ends when meets end
  # update the cell with status information
  display.clear_output(wait=True)  # This clears the output display or the console for display
  display.display(f'Gathering data for {str(current)}') # idk should have use just print HAHAHAH
  response = make_request(
  'data', # getting data as endpoint
    {
    'datasetid' : 'GHCND', # Global Historical Climatology Network - Daily (GHCND) dataset;
    # The Global Historical Climatology Network daily (GHCNd) is an integrated database of daily climate summaries from land surface statio
    'locationid' : 'CITY:US360019', # NYC
    'startdate' : current,
    'enddate' : current,
    'units' : 'metric',
    'limit' : 1000 # max allowed
    }
)
  if response.ok:
  # we extend the list instead of appending to avoid getting a nested list
    results.extend(response.json()['results'])
  # update the current date to avoid an infinite loop
  current += datetime.timedelta(days=1) # Increment the current date by one day
```

```
'Gathering data for 2018-02-01'
```

```
import pandas as pd
df = pd.DataFrame(results)
df.head()
```

	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

```
# Same Concept but much faster
start_date = datetime.date(2018, 1, 1)
end_date = datetime.date(2018, 2, 2)
data = []
response = make_request(
    'data',
    {
        'datasetid': 'GHCND',
        'locationid': 'CITY:US360019',  # NYC
        'startdate': start_date,
        'enddate': end_date,
        'units': 'metric',
        'limit': 1000
    }
)

if response.ok:
    data.extend(response.json()['results'])
else:
    print("Failed to fetch data:", response.status_code)
```

```
import pandas as pd
df = pd.DataFrame(data)
df.head()
```

	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

Now, we can create a dataframe with all this data. Notice there are multiple stations with values for each datatype on a given day. We don't know what the stations are, but we can look them up and add them to the data:

Save this data to a file:

```
df.to_csv('nyc_weather_2018.csv', index=False) # converts the datas to csv file with no index to be saved
```

and write it to the database:

```
import sqlite3

with sqlite3.connect('weather.db') as connection: # proper way of accessing or connecting to a database
  df.to_sql(
    'weather', connection, index=False, if_exists='replace'  # specifies the name which is weather for the first parameter,
    # next the connection which is the name of the database to be used, then the index is set to false meaning will not use the df index as
    # and then lastly is the if_exist whicih is set to replace which will replace the column if there is already existing column name like
)
```

For learning about merging dataframes, we will also get the data mapping station IDs to information about the station:

```
response = make_request(
  'stations', # getting stations endpoint
  {
    'datasetid' : 'GHCND', # Global Historical Climatology Network - Daily (GHCND) dataset; NOAA datasets
    'locationid' : 'CITY:US360019', # NYC
    'limit' : 1000 # max allowed
  }
)
stations = pd.DataFrame(response.json()['results'])[['id', 'name', 'latitude', 'longitude', 'elevation']] # fetches only the lists of colum
stations.to_csv('weather_stations.csv', index=False) # turns it into CSV file
stations.head() # lemme see sample
```

	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

```
with sqlite3.connect('weathers.db') as connection:
  stations.to_sql(
    'stations', connection, index=False, if_exists='replace'
  )
# converts the stations into database
```