```
# https://www.ncdc.noaa.gov/cdo-web/webservices/v2
```

Collecting temperature data from an API

Using the NCEI API

```
import requests
def make_request(endpoint, payload=None):
    """
    Make a request to a specific endpoint on the weather API
    passing headers and optional payload.

    Parameters:
        - endpoint: The endpoint of the API you want to
                    make a GET request to.
        - payload: A dictionary of data to pass along
                   with the request.

    Returns:
        Response object.
    """
    return requests.get(
        f'https://www.ncdc.noaa.gov/cdo-web/api/v2/{endpoint}',
        headers={
            'token': 'zVKGhvsaEJxUSbmXMizCLbJbGdQYMcIh'
        },
        params=payload
    )
```

```
# see available datasets
response = make_request('datasets')
response.status_code
```

```
200
```

200 status code means OK

```
# .keys() used for accessing keys in a dictionary
response.json().keys()
```

```
dict_keys(['metadata', 'results'])
```

in this case, metadata and results are the keys, let's try to remove the .keys() for further visualization

```
response.json()
```

```
{'metadata': {'resultset': {'offset': 1, 'count': 12, 'limit': 25}},
 'results': [{'name': 'City', 'id': 'CITY'},
  {'name': 'Climate Division', 'id': 'CLIM_DIV'},
  {'name': 'Climate Region', 'id': 'CLIM_REG'},
  {'name': 'Country', 'id': 'CNTRY'},
  {'name': 'County', 'id': 'CNTY'},
  {'name': 'Hydrologic Accounting Unit', 'id': 'HYD_ACC'},
  {'name': 'Hydrologic Cataloging Unit', 'id': 'HYD_CAT'},
  {'name': 'Hydrologic Region', 'id': 'HYD_REG'},
  {'name': 'Hydrologic Subregion', 'id': 'HYD_SUB'},
  {'name': 'State', 'id': 'ST'},
  {'name': 'US Territory', 'id': 'US_TERR'},
  {'name': 'Zip Code', 'id': 'ZIP'}]}
```

```
# index metadata to yield its values, note that metadata is the key
# in dictionaries, there are two parts, {keys: values}
response.json()['metadata']
```

```
{'resultset': {'offset': 1, 'count': 11, 'limit': 25}}
```

```
# index metadata to yield its values, note that results is the key
response.json()['results']
```

```
[{'name': 'City', 'id': 'CITY'},
 {'name': 'Climate Division', 'id': 'CLIM_DIV'},
 {'name': 'Climate Region', 'id': 'CLIM_REG'},
 {'name': 'Country', 'id': 'CNTRY'},
 {'name': 'County', 'id': 'CNTY'},
 {'name': 'Hydrologic Accounting Unit', 'id': 'HYD_ACC'},
 {'name': 'Hydrologic Cataloging Unit', 'id': 'HYD_CAT'},
 {'name': 'Hydrologic Region', 'id': 'HYD_REG'},
 {'name': 'Hydrologic Subregion', 'id': 'HYD_SUB'},
 {'name': 'State', 'id': 'ST'},
 {'name': 'US Territory', 'id': 'US_TERR'},
 {'name': 'Zip Code', 'id': 'ZIP'}]
```

```
response.json()['results'][0].keys()
```

```
dict_keys(['uid', 'mindate', 'maxdate', 'name', 'datacoverage', 'id'])
```

```
# for loop to traverse through the results while getting id and name
[(data['id'], data['name']) for data in response.json()['results']]
```

```
     [('GHCND', 'Daily Summaries'),
      ('GSOM', 'Global Summary of the Month'),
      ('GSOY', 'Global Summary of the Year'),
      ('NEXRAD2', 'Weather Radar (Level II)'),
      ('NEXRAD3', 'Weather Radar (Level III)'),
      ('NORMAL_ANN', 'Normals Annual/Seasonal'),
      ('NORMAL_DLY', 'Normals Daily'),
      ('NORMAL_HLY', 'Normals Hourly'),
      ('NORMAL_MLY', 'Normals Monthly'),
      ('PRECIP_15', 'Precipitation 15 Minute'),
      ('PRECIP_HLY', 'Precipitation Hourly')]
```

```
 # get data category id
 response = make_request(
  'datacategories',
  payload={
    'datasetid' : 'GHCND',
  }
 )
 response.status_code
```

```
     200
```

```
# because we made a different request,
# this will yield different results based on data categories on datasetid : GHCND
response.json()['results']
```

```
     [{'name': 'Evaporation', 'id': 'EVAP'},
      {'name': 'Land', 'id': 'LAND'},
      {'name': 'Precipitation', 'id': 'PRCP'},
      {'name': 'Sky cover & clouds', 'id': 'SKY'},
      {'name': 'Sunshine', 'id': 'SUN'},
      {'name': 'Air Temperature', 'id': 'TEMP'},
      {'name': 'Water', 'id': 'WATER'},
      {'name': 'Wind', 'id': 'WIND'},
      {'name': 'Weather Type', 'id': 'WXTYPE'}]
```

```
 # get data type id
 response = make_request(
  'datatypes',
  payload={
    'datacategoryid' : 'TEMP',
    'limit' : 100
  }
 )
 response.status_code
```

```
     200
```

```
# for loop to traverse to the data types and return its id and temperature
[(datatype['id'], datatype['name']) for datatype in response.json()['results']][-5:] # look at the last 5
```

```
     [('MNTM', 'Monthly mean temperature'),
      ('TAVG', 'Average Temperature.'),
      ('TMAX', 'Maximum temperature'),
      ('TMIN', 'Minimum temperature'),
      ('TOBS', 'Temperature at the time of observation')]
```

```
# get location category id
response = make_request(
    'locationcategories',
    {
         'datasetid' : 'GHCND'
    }
)
response.status_code
```

```
     200
```

```
# for an alternative, we use pprint to print dictionaries in an easier-to-read format
import pprint
pprint.pprint(response.json())
```

```
     {'metadata': {'resultset': {'count': 12, 'limit': 25, 'offset': 1}},
      'results': [{'id': 'CITY', 'name': 'City'},
                  {'id': 'CLIM_DIV', 'name': 'Climate Division'},
                  {'id': 'CLIM_REG', 'name': 'Climate Region'},
                  {'id': 'CNTRY', 'name': 'Country'},
                  {'id': 'CNTY', 'name': 'County'},
                  {'id': 'HYD_ACC', 'name': 'Hydrologic Accounting Unit'},
                  {'id': 'HYD_CAT', 'name': 'Hydrologic Cataloging Unit'},
                  {'id': 'HYD_REG', 'name': 'Hydrologic Region'},
                  {'id': 'HYD_SUB', 'name': 'Hydrologic Subregion'},
                  {'id': 'ST', 'name': 'State'},
                  {'id': 'US_TERR', 'name': 'US Territory'},
                  {'id': 'ZIP', 'name': 'Zip Code'}]}
```

```
def get_item(name, what, endpoint, start=1, end=None):
    """
    Grab the JSON payload for a given field by name using binary search.
    Parameters:
        - name: The item to look for.
        - what: Dictionary specifying what the item in `name` is.
        - endpoint: Where to look for the item.
        - start: The position to start at. We don't need to touch this, but the
                 function will manipulate this with recursion.
        - end: The last position of the cities. Used to find the midpoint, but
               like `start` this is not something we need to worry about.
    Returns:
        Dictionary of the information for the item if found otherwise
        an empty dictionary.
    """
    # find the midpoint which we use to cut the data in half each time
    mid = (start + (end if end else 1)) // 2

    # lowercase the name so this is not case-sensitive
    name = name.lower()

    # define the payload we will send with each request
    payload = {
        'datasetid' : 'GHCND',
        'sortfield' : 'name',
        'offset' : mid, # we will change the offset each time
        'limit' : 1 # we only want one value back
    }

    # make our request adding any additional filter parameters from `what`
    response = make_request(endpoint, {**payload, **what})

    if response.ok:
        # if response is ok, grab the end index from the response metadata the first time through
        end = end if end else response.json()['metadata']['resultset']['count']

        # grab the lowercase version of the current name
        current_name = response.json()['results'][0]['name'].lower()

        # if what we are searching for is in the current name, we have found our item
        if name in current_name:
            return response.json()['results'][0] # return the found item
        else:
            if start >= end:
                # if our start index is greater than or equal to our end, we couldn't find it
                return {}
            elif name < current_name:
                # our name comes before the current name in the alphabet, so we search further to the left
                return get_item(name, what, endpoint, start, mid - 1)
            elif name > current_name:
                # our name comes after the current name in the alphabet, so we search further to the right
                return get_item(name, what, endpoint, mid + 1, end)
    else:
        # response wasn't ok, use code to determine why
        print(f'Response not OK, status: {response.status_code}')
def get_location(name):
    """
    Grab the JSON payload for the location by name using binary search.
    Parameters:
        - name: The city to look for.
    Returns:
        Dictionary of the information for the city if found otherwise
        an empty dictionary.
    """
    return get_item(name, {'locationcategoryid' : 'CITY'}, 'locations')


# get NYC id
nyc = get_location('New York')
nyc

     {'mindate': '1869-01-01',
      'maxdate': '2024-03-11',
      'name': 'New York, NY US',
      'datacoverage': 1,
      'id': 'CITY:US360019'}


central_park = get_item('NY City Central Park', {'locationid' : nyc['id']}, 'stations')
central_park

     {'elevation': 42.7,
      'mindate': '1869-01-01',
      'maxdate': '2024-03-10',
      'latitude': 40.77898,
      'name': 'NY CITY CENTRAL PARK, NY US',
      'datacoverage': 1,
      'id': 'GHCND:USW00094728',
      'elevationUnit': 'METERS',
      'longitude': -73.96925}


# get NYC daily summaries data
response = make_request(
    'data',
    {
        'datasetid' : 'GHCND',
        'stationid' : central_park['id'],
        'locationid' : nyc['id'],
        'startdate' : '2018-10-01',
        'enddate' : '2018-10-31',
        'datatypeid' : ['TMIN', 'TMAX', 'TOBS'], # temperature at time of observation, min, and max
        'units' : 'metric'
```

```
        'units'  :  'metric',
        'limit' : 1000
    }
 )
response.status_code
```

```
200
```

```
# create dataframe for our examples
import pandas as pd

df = pd.DataFrame(response.json()['results'])
df.head()
```

	date	datatype	station	attributes	value
0	2018-10-01T00:00:00	TMAX	GHCND:USW00094728	,,W,2400	24.4
1	2018-10-01T00:00:00	TMIN	GHCND:USW00094728	,,W,2400	17.2
2	2018-10-02T00:00:00	TMAX	GHCND:USW00094728	,,W,2400	25.0
3	2018-10-02T00:00:00	TMIN	GHCND:USW00094728	,,W,2400	18.3
4	2018-10-03T00:00:00	TMAX	GHCND:USW00094728	,,W,2400	23.3

Next steps: View recommended plots

```
# gets all the certain values in the datatype column
df.datatype.unique()
```

```
array(['TMAX', 'TMIN'], dtype=object)
```

```
# if the item inside the if statement could be located, it would print Found!
if get_item(
  'NY City Central Park', {'locationid' : nyc['id'], 'datatypeid': 'TOBS'}, 'stations'
 ):
  print('Found!')
```

```
Found!
```

```
# using the get_item function but without any if statement
# if this occurs, it would return the items for La Guardia
laguardia = get_item(
    'LaGuardia', {'locationid' : nyc['id']}, 'stations'
)
laguardia
```

```
{'elevation': 3,
 'mindate': '1939-10-07',
 'maxdate': '2024-03-11',
 'latitude': 40.77945,
 'name': 'LAGUARDIA AIRPORT, NY US',
 'datacoverage': 1,
 'id': 'GHCND:USW00014732',
 'elevationUnit': 'METERS',
 'longitude': -73.88027}
```

```
# to check if the keys inside could be found
response = make_request(
    'data',
    {
        'datasetid' : 'GHCND',
        'stationid' : laguardia['id'],
        'locationid' : nyc['id'],
        'startdate' : '2018-10-01',
        'enddate' : '2018-10-31',
        'datatypeid' : ['TMIN', 'TMAX', 'TAVG'], # temperature at time of observation, min, and max
        'units' : 'metric',
        'limit' : 1000
    }
)
response.status_code
```

```
200
```

```
# get first five values in the dataframe
df = pd.DataFrame(response.json()['results'])
df.head()
```

	date	datatype	station	attributes	value
0	2018-10-01T00:00:00	TAVG	GHCND:USW00014732	H,,S,	21.2
1	2018-10-01T00:00:00	TMAX	GHCND:USW00014732	,,W,2400	25.6
2	2018-10-01T00:00:00	TMIN	GHCND:USW00014732	,,W,2400	18.3
3	2018-10-02T00:00:00	TAVG	GHCND:USW00014732	H,,S,	22.7
4	2018-10-02T00:00:00	TMAX	GHCND:USW00014732	,,W,2400	26.1

Next steps: View recommended plots

```
# to count the number of values per datatype
df.datatype.value_counts()
```

```
TAVG    31
TMAX    31
TMIN    31
Name: datatype, dtype: int64
```

```
df.to_csv('7.2 data/nyc_temperatures.csv', index=False)
```