```
import requests

def make_request(endpoint, payload=None):
  return requests.get(
      f'https://www.ncdc.noaa.gov/cdo-web/api/v2/{endpoint}',
      headers={'token': 'WBKsJKcmUpXqKqMAoGBZuuFwgqCWaktf'},
      params= payload
    )
```

```
response = make_request('datasets', {'startdate':'2024-01-01'})
response.status_code
```

```
200
```

```
response.json().keys()
```

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

```
response.json()['metadata']
```

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

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

```
[(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
```

```
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
```

```
[(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')]
```

Determine which Location Category we want

Now that we know which datatypes we will be collecting, we need to find the location to use. First, we need to figure out the location category. This is obtained from the locationcategories endpoint by passing the datasetid :

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

```
200
```

```
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'}]}
```

Get NYC Location ID

In order to find the location ID for New York, we need to search through all the cities available. Since we can ask the API to return the cities sorted, we can use binary search to find New York quickly without having to make many requests or request lots of data at once. The following function makes the first request to see how big the list of cities is and looks at the first value. From there it decides if it needs to move towards the beginning or end of the list by comparing the city we are looking for to others alphabetically. Each time it makes a request it can rule out half of the remaining data to search.

```
def get_item(name, what, endpoint, start=1, end=None):

    # 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):
    return get_item(name, {'locationcategoryid' : 'CITY'}, 'locations')
```

When we use binary search to find New York, we find it in just 8 requests despite it being close to the middle of 1,983 entries:

```
# 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'}
```

Get the station ID for Central Park

The most granular data is found at the station level:

```
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}
```

Request the temperature data

Finally, we have everything we need to make our request for the New York temperature data. For this we use the data endpoint and provide all the parameters we picked up throughout our exploration of the API:

```
# 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',
        'limit' : 1000
    }
)
response.status_code
```

```
200
```

Create a DataFrame

The Central Park station only has the daily minimum and maximum temperatures.

```
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

We didn't get TOBS because the station doesn't measure that:

```
df.datatype.unique()
```

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

Despite showing up in the data as measuring it... Real-world data is dirty!

```
if get_item(
    'NY City Central Park', {'locationid' : nyc['id'], 'datatypeid': 'TOBS'}, 'stations'
):
    print('Found!')
```

```
Found!
```

Using a different station

Let's use LaGuardia airport instead. It contains TAVG (average daily temperature):

```
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}
```

We make our request using the LaGuardia airport station this time and ask for TAVG instead of TOBS .

```
# get NYC daily summaries data
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
```

The request was successful, so let's make a 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

We should check we got what we wanted: 31 entries for TAVG, TMAX, and TMIN (1 per day):

```
df.datatype.value_counts()
```

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

Write the data to a CSV file for use in other notebooks.

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