Using the NCEI API

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

See what datasets are available

```
response = make_request('datasets', {'startdate':'2018-10-01'})
response.status_code

#in this part, we can change the date and check if the dataset is available
```

```
200
```

Get the keys of the result

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

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

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

#offset is the beginning of the record
#limit is the limitations amount of the result that must be returned
#count is the number of data returned
```

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

Figure out what data is in the result

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

#we can use indexing to specifically choose for the data
```

```
{'uid': 'gov.noaa.ncdc:C00861',
 'mindate': '1750-02-01',
 'maxdate': '2024-03-11',
 'name': 'Daily Summaries',
 'datacoverage': 1,
 'id': 'GHCND'}
```

Parse the result

```
[(data['id'], data['name']) for data in response.json()['results']]

#we are fetching out the dictionary keys : id and name in the results
#using for loop, we are getting each value of the dictionary keys specified
```

```
[('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')]
```

Figure out which data category we want

```
response = make_request( # making a request if the given parameters are available
'datacategories', # -> endpoint
payload={
'datasetid' : 'GHCND' # -> GHCND is our data category specified by the additional parameters datasetid
}
)
response.status_code # checking if its OK, if output is 200, it is OK else its not

#using datacategories endpoint, we can specify the data category that we want
```

```
200
```

```
response.json()['results']
#we are displaying all the results of the category that we have chosen
#inside the result portion of the JSON response, the values/elements of our list are dictionaries with name and id as their keys
```

```
[{'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'}]
```

Grab the data type ID for the Temperature category

```
response = make_request(  # making a request if the given parameters are available
'datatypes', # -> endpoint
payload={
'datacategoryid' : 'TEMP', # our new category specified by the data category id
'limit' : 100 # limitions of our results that will get returned
}
)
response.status_code
```

```
200
```

```
[(datatype['id'], datatype['name']) for datatype in response.json()['results']][-5:]

# since we did fetch the TEMP, we are getting the id and the name of the TEMP
# the TEMP category have its own id and name, in this case, we are printing the last 5 of our id and name
```

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

Removing the index slicing to check the results

```
# if we remove the index slicing in the code cell above, we can see the results properly
[(datatype['id'], datatype['name']) for datatype in response.json()['results']]
```

```
 ('MLY-DUTR-STDDEV',
  'Long-term standard deviations of monthly diurnal temperature range'),
 ('MLY-TAVG-NORMAL', 'Long-term averages of monthly average temperature'),
 ('MLY-TAVG-STDDEV',
  'Long-term standard deviations of monthly average temperature'),
 ('MLY-TMAX-AVGNDS-GRTH040',
  'Long-term average number of days per month where tmax is greater than or equal to 40F'),
 ('MLY-TMAX-AVGNDS-GRTH050',
  'Long-term average number of days per month where tmax is greater than or equal to 50F'),
 ('MLY-TMAX-AVGNDS-GRTH060',
  'Long-term average number of days per month where tmax is greater than or equal to 60F'),
 ('MLY-TMAX-AVGNDS-GRTH070',
  'Long-term average number of days per month where tmax is greater than or equal to 70F'),
 ('MLY-TMAX-AVGNDS-GRTH080',
  'Long-term average number of days per month where tmax is greater than or equal to 80F'),
 ('MLY-TMAX-AVGNDS-GRTH090',
  'Long-term average number of days per month where tmax is greater than or equal to 90F'),
 ('MLY-TMAX-AVGNDS-GRTH100',
  'Long-term average number of days per month where tmax is greater than or equal to 100F'),
 ('MLY-TMAX-AVGNDS-LSTH032',
  'Long-term average number of days per month where tmax is less than or equal to 32F'),
 ('MLY-TMAX-NORMAL', 'Long-term averages of monthly maximum temperature'),
 ('MLY-TMAX-STDDEV',
  'Long-term standard deviations of monthly maximum temperature'),
 ('MLY-TMIN-AVGNDS-LSTH000',
  'Long-term average number of days per month where tmin is less than or equal to 0F'),
 ('MLY-TMIN-AVGNDS-LSTH010',
  'Long-term average number of days per month where tmin is less than or equal to 10F'),
 ('MLY-TMIN-AVGNDS-LSTH020',
  'Long-term average number of days per month where tmin is less than or equal to 20F'),
 ('MLY-TMIN-AVGNDS-LSTH032',
  'Long-term average number of days per month where tmin is less than or equal to 32F'),
 ('MLY-TMIN-AVGNDS-LSTH040',
  'Long-term average number of days per month where tmin is less than or equal to 40F'),
 ('MLY-TMIN-AVGNDS-LSTH050',
  'Long-term average number of days per month where tmin is less than or equal to 50F'),
 ('MLY-TMIN-AVGNDS-LSTH060',
  'Long-term average number of days per month where tmin is less than or equal to 60F'),
 ('MLY-TMIN-AVGNDS-LSTH070',
  'Long-term average number of days per month where tmin is less than or equal to 70F'),
 ('MLY-TMIN-NORMAL', 'Long-term averages of monthly minimum temperature'),
 ('MLY-TMIN-STDDEV',
  'Long-term standard deviations of monthly minimum temperature'),
 ('MMNT', 'Monthly Mean minimum temperature'),
 ('MMXT', 'Monthly Mean maximum temperature'),
 ('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

```
response = make_request( # making a request
'locationcategories', # changing the endpoint to location categories
{
'datasetid' : 'GHCND' # specifying our category of the location that we want
}
)
response.status_code # checking if our request is possible
```

```
200
```

```
import pprint
pprint.pprint(response.json())

# displaying all the keys and its value of the metadata
# since we have 12 different ids or dictionary keys, the count in the resultset is 12
# the limitation is limited to 25 results to be returned
# offset is the starting point of our resultset, since we have start at the first resultset, the value of our offset is 1
```

```
{'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

```
def get_item(name, what, endpoint, start=1, end=None):
  #name : parameter is for what item to look for
  #what : a dictionary specifying what the item in name is
  #endpoint : where to look for the item
  #start :

  mid = (start + (end if end else 1)) // 2
  name = name.lower()

  payload = {
      'datasetid' : 'GHCND',
      'sortfield' : 'name',
      'offset' : mid,
      'limit' : 1
  }

  response = make_request(endpoint, {**payload, **what})

  if response.ok:
    end = end if end else response.json()['metadata']['resultset']['count']
    current_name = response.json()['results'][0]['name'].lower()
    if name in current_name:
      return response.json()['results'][0]
    else:
      if start >= end:
        return()
      elif name < current_name:
        return get_item(name, what, endpoint, start, mid - 1)
      elif name > current_name:
        return get_item(name, what, endpoint, mid + 1, end )
  else:
    print(f'Response not OK, status: {response.status_code}')

def get_location(name):
  return get_item(name, {'locationcategoryid' : 'CITY'}, 'locations')


nyc = get_location('New York')
nyc
```

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

TRYING ANOTHER CITIES

```
bstn = get_location('Boston') #trying boston city
bstn
```

```
     {'mindate': '1884-11-01',
      'maxdate': '2024-03-11',
      'name': 'Boston, MA US',
      'datacoverage': 1,
      'id': 'CITY:US250002'}
```

```
tky = get_location('Tokyo') #trying tokyo
tky
```

```
     {'mindate': '1949-01-01',
      'maxdate': '2024-03-10',
      'name': 'Tokyo, JA',
      'datacoverage': 1,
      'id': 'CITY:JA000016'}
```

Get the station ID for Central Park

```
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}
```

REQUESTING THE TEMPERATURE 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

```
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

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

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

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

```
Found!
```

Using a different station

```
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}
```

```
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
```

```
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

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

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

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