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': 'ADPwCcoPIaSsUZaapBWqIqMcJIbfpgVB'},
                      params=payload)
```

Seeing what datasets available

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

```
200
```

always have error because of mispelled words and missing letters

Getting the keys of results

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

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

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

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

Figuring out what data is in result

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

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

Parsing the result

```
[(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')]
```

Figuring out which data category we want

```
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'}]
```

Grabbing the data type id for temperature

```
response = make_request('datatypes', payload={'datacategoryid':'TEMP', 'limit':100})
response.status_code
```

```
200
```

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

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

figuring out the size and limit

```
response = make_request('datatypes', payload={'datacategoryid':'TEMP', 'limit':20})
response.status_code
```

```
200
```

```
data = [(datatype['id'], datatype['name']) for datatype in response.json()['results']]
j = 0
for i in data:
  print(i)
  j = j+1
print(j)
```

```
('CDSD', 'Cooling Degree Days Season to Date')
('DATN', 'Number of days included in the multiday minimum temperature (MDTN)')
('DATX', 'Number of days included in the multiday maximum temperature (MDTX)')
('DLY-DUTR-NORMAL', 'Long-term averages of daily diurnal temperature range')
('DLY-DUTR-STDDEV', 'Long-term standard deviations of daily diurnal temperature range
('DLY-TAVG-NORMAL', 'Long-term averages of daily average temperature')
('DLY-TAVG-STDDEV', 'Long-term standard deviations of daily average temperature')
('DLY-TMAX-NORMAL', 'Long-term averages of daily maximum temperature')
('DLY-TMAX-STDDEV', 'Long-term standard deviations of daily maximum temperature')
('DLY-TMIN-NORMAL', 'Long-term averages of daily minimum temperature')
('DLY-TMIN-STDDEV', 'Long-term standard deviations of daily minimum temperature')
('EMNT', 'Extreme minimum temperature for the period.')
('EMXT', 'Extreme maximum temperature for the period.')
('HDSD', 'Heating Degree Days Season to Date')
('HLY-DEWP-10PCTL', 'Dew point 10th percentile')
('HLY-DEWP-90PCTL', 'Dew point 90th percentile')
('HLY-DEWP-NORMAL', 'Dew point mean')
('HLY-HIDX-NORMAL', 'Heat index mean')
('HLY-TEMP-10PCTL', 'Temperature 10th percentile')
('HLY-TEMP-90PCTL', 'Temperature 90th percentile')
20
```

Determining which location category we want

```
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'}]}
```

Getting NYC location ID

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

Getting 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
```

```
 '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'], 'locati
response.status_code
```

```
200
```

Creating 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 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'], 'locatic
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('/content/nyc_temperatures.csv', index=False)
```