# Lab 4: Get NDAWN Data

This tool retrieves the NDAWN station information (station name, station number, and station location) from the NDAWN HTML. It also retrieves daily temperature information based on user input variables and concatenates them to output one CSV with all the specified information for each station. The script will also output a summarizing statistics about the range of the maximum and minimum daily temperatures.

In [ ]:
```
import datetime
import pandas as pd
import requests
import os

from bs4 import BeautifulSoup
from io import StringIO
```

In [ ]:
```
def get_stations(url, out_path):
    """
    parameter
    ----------

    return
    ------
    """

    r = requests.get(url)
    soup = BeautifulSoup(r.content, 'html.parser')
    results = soup.find(id="station-map")


    # Extract station name and number
    split_html = []
    for item in results:
        if item == "\n":
            pass
        else:
            new = str(item)[6:-2]
            split_html.append(new.split('"'))


    # create lookup table for stations with station name and number
    stations = {}
    for station in split_html:
        station_name = str(station[1]).split()
        if len(station_name) == 3:
            key = f"{station_name[0]} {station_name[1]}"
            stations[key] = {"station_number": station[5][27:]}
        else:
            stations[station_name[0]] = {"station_number": station[5][27:]} #
 append station numbers


    # extract and add lat long coordinates for each stations to lookup table
    for station in stations:
        num = stations[station]["station_number"]
        url_2 = f"https://ndawn.ndsu.nodak.edu/station-info.html?station={num}
"
        r_2 = requests.get(url_2)

        # Get latitude and longitude for each station
        soup_2 = BeautifulSoup(r_2.content, 'html.parser')
        details = soup_2.find(id="details")
        table = details.find("table")
        lat_html =(table.findAll("tr"))[3]
        long_html = (table.findAll("tr"))[4]
        lat = lat_html.getText()[9:-1] # extract latitude
        long= long_html.getText()[10:-1] # extract longitude
        stations[station]["x"] = long # append lat/long to stations lookup dic
tionary
        stations[station]["y"] = lat

        stations_df = pd.DataFrame(stations).transpose()
```

```
        stations_df.to_csv(os.path.join(out_path,"stations.csv"), index_label=
"station_name", index=True)

    return stations
```

In [ ]:
```
def set_variables(time_option):
    ''' Formats variables based on user input from daily variables
    table for url to retrieve csv from NDAWN.


    parameters
    ----------
    time_option: str

    return
    ------
    format_vars: str
        formatted weather variables from user inputto be used in ndawn url
    '''
    # create lookup dictionary for each daily weather varible code
    time_url = f"https://ndawn.ndsu.nodak.edu/weather-data-{time_option}.html"
    r = requests.get(time_url)
    soup = BeautifulSoup(r.content, 'html.parser')
    table = soup.find(id="table-vars")
    values = table.findAll(value=True)

    # extract variable code and description
    weather_vars = {}
    for item in values:
        trim_value = str(item).split('"')
        if "selected" in trim_value[0]:
            desc = trim_value[4].split("<")
            weather_vars[trim_value[3]] = (desc[0].replace(">",""))
        else:
            desc = trim_value[2].split("<")
            weather_vars[trim_value[1]] = (desc[0].replace(">",""))



    # print possible variable codes for user
    for key in weather_vars:
        print(f"{key}: {weather_vars[key]}")

    in_vars = input("\nEnter the variable codes seperated by a comma or 'all'
 to obtain data for all variables: ")
    if in_vars == "all":
        list_vars = []
        for key in weather_vars:
            list_vars.append(key)
    else:
        list_vars = in_vars.split(",")

    for var in list_vars:
        assert var in weather_vars, "Invalid variable entered"

    format_vars = f"&variable={'&variable='.join(list_vars)}"
    return format_vars
```

In [ ]:
```
def get_data(time_option, base_url, variables, out_path):
    """

    parameters
    ----------
    time_option: str

    base_url: str

    variables: str


    return
    ------
    ndawn_df: pandas dataframe
    """

    # Set 30 day range for data
    start = input("Enter dataset begin date YYYY-MM-DD: ")
    startdate = datetime.datetime.strptime(start, '%Y-%m-%d') #format date
    enddate = startdate + datetime.timedelta(29) #calculate date 30 days from
 start
    assert enddate <= datetime.datetime.now(), "End date is in the future of t
oday's date. Unable to fetch data"
    print(f"End date: {enddate.strftime('%Y-%m-%d')}")

    print("Extracting data...")
    master = []
    for station in stations:
        station_num = (stations[station]["station_number"])
        year = enddate.strftime('%Y')
        end_date = enddate.strftime('%Y-%m-%d') # format end date

        # construct url
        url = f"{base_url}{station_num}{variables}&year={year}&ttype=daily&beg
in_date={start}&end_date={end_date}"
        r = requests.get(url)
        assert r.status_code is 200, ("Url is not valid")

        # Sourced from Jeff's code
        # Convert csv data to string
        content = str(r.content)

        # Remove large, unnecessary header
        trimContent = content[content.find('Station'):len(content)]

        # Replace newline/return with string literal newline
        formatContent = trimContent.replace('\\r\\n', '\n')

        # Convert content to file object
        contentFile = StringIO(formatContent)
        df = pd.read_csv(contentFile, header = [0,1],)

        master.append(df)

    # convert list to dataframe
```

```
    ndawn_df = pd.concat(master, axis=0, ignore_index=True)
    old_cols = list(ndawn_df.columns) # get column names

    # adjust column names
    new_cols = []

    # Iterate through column names
    for number in range(0, len(old_cols)):
        # If no unit, keep header unchanged, pass into new list
        if 'Unnamed' in old_cols[number][1]:
            new_cols.append(old_cols[number][0])
        # If unit exists, concatenate header and unit, pass into new list
        else:
            newHeader = old_cols[number][0] + ' (' + old_cols[number][1] + ')
 '
            new_cols.append(newHeader)

    # Rename columns to new columns names
    ndawn_df.columns = new_cols

    # Create new column with the date
    ndawn_df['Date']= pd.to_datetime(ndawn_df[['Year',
                                               'Month',
                                               'Day']])


    # remove invalid stations
    ndawn_df.drop(ndawn_df[ndawn_df["Station Name"] == "'"].index , inplace=Tr
ue)

    return ndawn_df

    print("Data extracted. Exporting file...")
    ndawn_df.to_csv(os.path.join(out_path,"ndawn_data.csv"), index=True, index
_label="ID")

    print("Completed")
```

In [ ]:

```
# Example code
url = r"https://ndawn.ndsu.nodak.edu/"
out_path = r"C:\Users\msong\Desktop\arc21\lab4\output data"

time_option = "daily"
base_url = r"https://ndawn.ndsu.nodak.edu/table.csv?station="

# get stations dynamically from ndawn website
stations = get_stations(url, out_path)

# Obtain daily data from each ndawn weather station
variables = set_variables(time_option)
data = get_data(time_option, base_url, variables, out_path)
```

In [ ]:
```
# Get statistics about ndawn information
cols = list(data.columns)

max_max = data[cols[7]].max() # upper range of max daily temp
max_min = data[cols[7]].min() # lower range of max daily temp

min_max = data[cols[9]].max() # upper range of min daily temp
min_min = data[cols[9]].min() # lower range of min daily temp

print(f"Maximum temperature range: {max_min} to {max_max}\nMinimum temperature
range: {min_min} to {min_max}")
```

In [ ]:
```

```