FINAL_Data_Wrangling

December 13, 2021

1 Weather data set

[1]:
```
import pandas as pd
import numpy as np
import requests
import json
import datetime
from pandas_profiling import ProfileReport
```

[2]:
```
Token= ''
```

2 Connecticut

Connecticut will be the simplest of all four states to retrieve information from. In a single API call, information regarding the full time period of interest can be gathered, without any missing values:

[3]:
```
#create empty lists to store CT data
ct_dates_temp = []
ct_dates_prcp = []
ct_temps = []
ct_prcp = []

#for each year from 2020-2021 ...
for year in range(2020, 2022):
    year = str(year)
    print('working on year '+year)

    #make the api call
    r = requests.get('https://www.ncdc.noaa.gov/cdo-web/api/v2/data?
 ↪datasetid=GHCND&datatypeid=TAVG&datatypeid=PRCP&limit=1000&stationid=GHCND:
 ↪USW00014740&startdate='+year+'-01-01&enddate='+year+'-12-31',␣
 ↪headers={'token':Token})
    #load the api response as a json
    d = json.loads(r.text)
    #get all items in the response which are average temperature readings
    ct_avg_temps = [item for item in d['results'] if item['datatype']=='TAVG']
    #get the date field from all average temperature readings
```

```
    ct_dates_temp += [item['date'] for item in ct_avg_temps]
    #get the actual average temperature from all average temperature readings
    ct_temps += [item['value'] for item in ct_avg_temps]
    #get all items in the response which are average precipitation readings
    ct_avg_prcp = [item for item in d['results'] if item['datatype']=='PRCP']
    #get the date field from all average precipitation readings
    ct_dates_prcp += [item['date'] for item in ct_avg_prcp]
    #get the actual average precipitation from all average precipitation␣
 ↪readings
    ct_prcp += [item['value'] for item in ct_avg_prcp]

print('done')
```

```
working on year 2020
working on year 2021
done
```

[4]:
```
#check length of Connecticut data list
print(len(ct_temps))
```

```
673
```

3 Maine

The remaining three states (Maine, Massachuesetts, and Vermont) will be slightly trickier to retrieve complete data from. For some reason, the API returns data from our time period of interest, but omits data from the time period of 2020-11-29 to 2020-12-31. This information *does* exist, however, so I'll make a separate, second API call to retrieve info from the missing time frame. I'll then insert it into the original lists of data using two self-defined functions, **insert_list** and **insert_state**. Lastly, I'll check the length of the aggregrated lists to make sure we have about the same number of values for each state.

[5]:
```
#Create empty lists to store ME data
#Note: avg temp not available for Maine, so will use temp min and temp max␣
 ↪instead
me_dates_temp_min = []
me_dates_temp_max = []
me_dates_prcp = []
me_temps_min = []
me_temps_max = []
me_prcp = []

#for each year from 2020-2021 ...
for year in range(2020, 2022):
    year = str(year)
    print('working on year '+year)

    #make the api call
```

```
    r = requests.get('https://www.ncdc.noaa.gov/cdo-web/api/v2/data?
↪datasetid=GHCND&datatypeid=PRCP&&datatypeid=TMAX&datatypeid=TMIN&limit=1000&stationid=GHCND
↪USW00094626&startdate='+year+'-01-01&enddate='+year+'-12-31',␣
↪headers={'token':Token})
    #load the api response as a json
    d = json.loads(r.text)
    #get all items in the response which are MIN temp readings
    me_min_temps_item = [item for item in d['results'] if␣
↪item['datatype']=='TMIN']
    #get the date field from all MIN temperature readings
    me_dates_temp_min += [item['date'] for item in me_min_temps_item]
    #get the actual min temperature from all MIN temperature readings
    me_temps_min += [item['value'] for item in me_min_temps_item]
    #get all items in the response which are MAX temperature readings
    me_max_temp_item = [item for item in d['results'] if␣
↪item['datatype']=='TMAX']
    #get the date field from all MAX temperature readings
    me_dates_temp_max += [item['date'] for item in me_max_temp_item]
    #get the actual average temperature from all MAX temperature readings
    me_temps_max += [item['value'] for item in me_max_temp_item]
    #get all items in the response which are average PRCP readings
    me_avg_prcp = [item for item in d['results'] if item['datatype']=='PRCP']
    #get the date field from all average PRCP readings
    me_dates_prcp += [item['date'] for item in me_avg_prcp]
    #get the actual average precipitation from all average PRCP readings
    me_prcp += [item['value'] for item in me_avg_prcp]

print('done')
```

```
working on year 2020
working on year 2021
done
```

[6]:
```
#Get Maine weather info for missing section: 11/30/20-12/31/20
me_dates_temp_min2 = []
me_dates_temp_max2 = []
me_dates_prcp2 = []
me_temps_min2 = []
me_temps_max2 = []
me_prcp2 = []

#select only from year 2020...
for year in range(2020, 2021):
    year = str(year)
    print('working on year '+year)

    #make the api call for specififc date range: 11/30/20-12/31/20
```

```
    r = requests.get('https://www.ncdc.noaa.gov/cdo-web/api/v2/data?
↪datasetid=GHCND&datatypeid=PRCP&&datatypeid=TMAX&datatypeid=TMIN&limit=1000&stationid=GHCND
↪USW00094626&startdate=2020-11-30&enddate=2020-12-31', headers={'token':
↪Token})
    d = json.loads(r.text)
    #get all items in the response which are MIN temp readings
    me_min_temps_item = [item for item in d['results'] if␣
↪item['datatype']=='TMIN']
    #get the date field from all MIN temperature readings
    me_dates_temp_min2 += [item['date'] for item in me_min_temps_item]
    #get the actual min temperature from all MIN temperature readings
    me_temps_min2 += [item['value'] for item in me_min_temps_item]
    #get all items in the response which are MAX temperature readings
    me_max_temp_item = [item for item in d['results'] if␣
↪item['datatype']=='TMAX']
    #get the date field from all MAX temperature readings
    me_dates_temp_max2 += [item['date'] for item in me_max_temp_item]
    #get the actual average temperature from all MAX temperature readings
    me_temps_max2 += [item['value'] for item in me_max_temp_item]
    #get all items in the response which are average PRCP readings
    me_avg_prcp2 = [item for item in d['results'] if item['datatype']=='PRCP']
    #get the date field from all average PRCP readings
    me_dates_prcp2 += [item['date'] for item in me_avg_prcp2]
    #get the actual average precipitation from all average PRCP readings
    me_prcp2 += [item['value'] for item in me_avg_prcp2]

print('done')
```

```
working on year 2020
done
```

Here I create two functions to help insert data from the missing time periods into the full data sets, **insert_list** and **insert_state**.

[7]:
```
def insert_list(base_list, inserted_list, last_pos):
    """Inserts a list into a base list at an indicated position"""
    for i in range(len(inserted_list)):
        base_list.insert(i + last_pos, inserted_list[i])
```

[8]:
```
def insert_state(list_of_lists, last_date):
    """Inserts missing weather data at date of first missing weather data"""
    last_pos = (list_of_lists[0].index(last_date)+1)
    insert_list(list_of_lists[0], list_of_lists[6], last_pos)
    insert_list(list_of_lists[1], list_of_lists[7], last_pos)
    insert_list(list_of_lists[2], list_of_lists[8], last_pos)
    insert_list(list_of_lists[3], list_of_lists[9], last_pos)
    insert_list(list_of_lists[4], list_of_lists[10], last_pos)
    insert_list(list_of_lists[5], list_of_lists[11], last_pos)
```

[9]:
```
#Create a list of lists for Maine to pass into the above functions:
maine_lol = [me_dates_temp_min, me_dates_temp_max, me_dates_prcp, me_temps_min,␣
↪me_temps_max, me_prcp, me_dates_temp_min2,␣
↪me_dates_temp_max2,me_dates_prcp2, me_temps_min2, me_temps_max2, me_prcp2]
```

[10]:
```
#Call function with date of last available data for Maine in original data list:
insert_state(maine_lol,'2020-11-29T00:00:00')
```

[11]:
```
#Check updated length of Maine data list:
print(len(me_temps_min))
```

```
671
```

4 Massachusetts

[12]:
```
#Create empty lists to store MA data
#Note: avg temp not available for Massachusetts, so will use temp min and temp␣
↪max instead
ma_dates_temp_min = []
ma_dates_temp_max = []
ma_dates_prcp = []
ma_temps_min = []
ma_temps_max = []
ma_prcp = []

#for each year from 2020-2021 ...
for year in range(2020, 2022):
    year = str(year)
    print('working on year '+year)

    #make the api call
    r = requests.get('https://www.ncdc.noaa.gov/cdo-web/api/v2/data?
↪datasetid=GHCND&datatypeid=PRCP&&datatypeid=TMAX&datatypeid=TMIN&limit=1000&stationid=GHCND
↪USC00193624&startdate='+year+'-01-01&enddate='+year+'-12-31',␣
↪headers={'token':Token})
    #load the api response as a json
    d = json.loads(r.text)
    #get all items in the response which are MIN temp readings
    ma_min_temps_item = [item for item in d['results'] if␣
↪item['datatype']=='TMIN']
    #get the date field from all MIN temperature readings
    ma_dates_temp_min += [item['date'] for item in ma_min_temps_item]
    #get the actual min temperature from all MIN temperature readings
    ma_temps_min += [item['value'] for item in ma_min_temps_item]
    #get all items in the response which are MAX temperature readings
    ma_max_temp_item = [item for item in d['results'] if␣
↪item['datatype']=='TMAX']
```

```
    #get the date field from all MAX temperature readings
    ma_dates_temp_max += [item['date'] for item in ma_max_temp_item]
    #get the actual average temperature from all MAX temperature readings
    ma_temps_max += [item['value'] for item in ma_max_temp_item]
    #get all items in the response which are average PRCP readings
    ma_avg_prcp = [item for item in d['results'] if item['datatype']=='PRCP']
    #get the date field from all average PRCP readings
    ma_dates_prcp += [item['date'] for item in ma_avg_prcp]
    #get the actual average precipitation from all average PRCP readings
    ma_prcp += [item['value'] for item in ma_avg_prcp]

print('done')
```

```
working on year 2020
working on year 2021
done
```

[13]:
```
#Get Massachusetts weather info for missing section: 11/29/20-12/31/20
ma_dates_temp_min2 = []
ma_dates_temp_max2 = []
ma_dates_prcp2 = []
ma_temps_min2 = []
ma_temps_max2 = []
ma_prcp2 = []

#select only from year 2020...
for year in range(2020, 2021):
    year = str(year)
    print('working on year '+year)

    #make the api call for specififc date range: 11/30/20-12/31/20
    r = requests.get('https://www.ncdc.noaa.gov/cdo-web/api/v2/data?
↪datasetid=GHCND&datatypeid=PRCP&&datatypeid=TMAX&datatypeid=TMIN&limit=1000&stationid=GHCND
↪USC00193624&startdate=2020-11-29&enddate=2020-12-31', headers={'token':
↪Token})
    d = json.loads(r.text)
    #get all items in the response which are MIN temp readings
    ma_min_temps_item = [item for item in d['results'] if␣
↪item['datatype']=='TMIN']
    #get the date field from all MIN temperature readings
    ma_dates_temp_min2 += [item['date'] for item in ma_min_temps_item]
    #get the actual min temperature from all MIN temperature readings
    ma_temps_min2 += [item['value'] for item in ma_min_temps_item]
    #get all items in the response which are MAX temperature readings
    ma_max_temp_item = [item for item in d['results'] if␣
↪item['datatype']=='TMAX']
    #get the date field from all MAX temperature readings
```

```
    ma_dates_temp_max2 += [item['date'] for item in ma_max_temp_item]
    #get the actual average temperature from all MAX temperature readings
    ma_temps_max2 += [item['value'] for item in ma_max_temp_item]
    #get all items in the response which are average PRCP readings
    ma_avg_prcp2 = [item for item in d['results'] if item['datatype']=='PRCP']
    #get the date field from all average PRCP readings
    ma_dates_prcp2 += [item['date'] for item in ma_avg_prcp2]
    #get the actual average precipitation from all average PRCP readings
    ma_prcp2 += [item['value'] for item in ma_avg_prcp2]

print('done')
```

```
working on year 2020
done
```

[14]:
```
#Create a list of lists for Massachusetts to pass into the above functions:
massachusetts_lol = [ma_dates_temp_min, ma_dates_temp_max, ma_dates_prcp,␣
 ↪ma_temps_min, ma_temps_max, ma_prcp, ma_dates_temp_min2, ma_dates_temp_max2,␣
 ↪ma_dates_prcp2, ma_temps_min2, ma_temps_max2, ma_prcp2]
```

[15]:
```
#Call function with date of last available data for Massachusetts in original␣
 ↪data list:
insert_state(massachusetts_lol,'2020-11-28T00:00:00')
```

[16]:
```
#Check list length
print(len(ma_temps_min))
```

```
673
```

5 Vermont

[17]:
```
#Create empty lists to store VT data
#Note: avg temp not available for Vermont, so will use temp min and temp max␣
 ↪instead
vt_dates_temp_min = []
vt_dates_temp_max = []
vt_dates_prcp = []
vt_temps_min = []
vt_temps_max = []
vt_prcp = []

#for each year from 2020-2021 ...
for year in range(2020, 2022):
    year = str(year)
    print('working on year '+year)

    #make the api call
```

```
    r = requests.get('https://www.ncdc.noaa.gov/cdo-web/api/v2/data?
↪datasetid=GHCND&datatypeid=PRCP&&datatypeid=TMAX&datatypeid=TMIN&limit=1000&stationid=GHCND
↪USW00014742&startdate='+year+'-01-01&enddate='+year+'-12-31',␣
↪headers={'token':Token})
    #load the api response as a json
    d = json.loads(r.text)
    #get all items in the response which are MIN temp readings
    vt_min_temps_item = [item for item in d['results'] if␣
↪item['datatype']=='TMIN']
    #get the date field from all MIN temperature readings
    vt_dates_temp_min += [item['date'] for item in vt_min_temps_item]
    #get the actual min temperature from all MIN temperature readings
    vt_temps_min += [item['value'] for item in vt_min_temps_item]
    #get all items in the response which are MAX temperature readings
    vt_max_temp_item = [item for item in d['results'] if␣
↪item['datatype']=='TMAX']
    #get the date field from all MAX temperature readings
    vt_dates_temp_max += [item['date'] for item in vt_max_temp_item]
    #get the actual average temperature from all MAX temperature readings
    vt_temps_max += [item['value'] for item in vt_max_temp_item]
    #get all items in the response which are average PRCP readings
    vt_avg_prcp = [item for item in d['results'] if item['datatype']=='PRCP']
    #get the date field from all average PRCP readings
    vt_dates_prcp += [item['date'] for item in vt_avg_prcp]
    #get the actual average precipitation from all average PRCP readings
    vt_prcp += [item['value'] for item in vt_avg_prcp]

print('done')
```

```
working on year 2020
working on year 2021
done
```

[18]:
```
#Get Vermont weather info for missing section: 11/29/20-12/31/20
vt_dates_temp_min2 = []
vt_dates_temp_max2 = []
vt_dates_prcp2 = []
vt_temps_min2 = []
vt_temps_max2 = []
vt_prcp2 = []

#select only from year 2020...
for year in range(2020, 2021):
    year = str(year)
    print('working on year '+year)

    #make the api call for specififc date range: 11/30/20-12/31/20
```

```
    r = requests.get('https://www.ncdc.noaa.gov/cdo-web/api/v2/data?
↪datasetid=GHCND&datatypeid=PRCP&&datatypeid=TMAX&datatypeid=TMIN&limit=1000&stationid=GHCND
↪USW00014742&startdate=2020-11-29&enddate=2020-12-31', headers={'token':
↪Token})
    d = json.loads(r.text)
    #get all items in the response which are MIN temp readings
    vt_min_temps_item = [item for item in d['results'] if␣
↪item['datatype']=='TMIN']
    #get the date field from all MIN temperature readings
    vt_dates_temp_min2 += [item['date'] for item in vt_min_temps_item]
    #get the actual min temperature from all MIN temperature readings
    vt_temps_min2 += [item['value'] for item in vt_min_temps_item]
    #get all items in the response which are MAX temperature readings
    vt_max_temp_item = [item for item in d['results'] if␣
↪item['datatype']=='TMAX']
    #get the date field from all MAX temperature readings
    vt_dates_temp_max2 += [item['date'] for item in vt_max_temp_item]
    #get the actual average temperature from all MAX temperature readings
    vt_temps_max2 += [item['value'] for item in vt_max_temp_item]
    #get all items in the response which are average PRCP readings
    vt_avg_prcp2 = [item for item in d['results'] if item['datatype']=='PRCP']
    #get the date field from all average PRCP readings
    vt_dates_prcp2 += [item['date'] for item in vt_avg_prcp2]
    #get the actual average precipitation from all average PRCP readings
    vt_prcp2 += [item['value'] for item in vt_avg_prcp2]

print('done')
```

```
working on year 2020
done
```

[19]:
```
#Create a list of lists for Vermont to pass into the above functions:
vermont_lol = [vt_dates_temp_min, vt_dates_temp_max, vt_dates_prcp,␣
↪vt_temps_min, vt_temps_max, vt_prcp, vt_dates_temp_min2, vt_dates_temp_max2,␣
↪vt_dates_prcp2, vt_temps_min2, vt_temps_max2, vt_prcp2]
```

[20]:
```
#Call function with date of last available data for Vermont in original data␣
↪list:
insert_state(vermont_lol,'2020-11-28T00:00:00')
```

[21]:
```
#Check list length
print(len(vt_temps_min))
```

```
673
```

5.0.1 Put lists of data into dataframes by state:

5.0.2 Connecticut:

[22]:
```
#Zip dates together with respective values per state:
#convert all date values to datetime objects:

#Connecticut:
df_ct_temps = pd.DataFrame(list(zip(ct_dates_temp, ct_temps)), columns =␣
↪['ct_date', 'ct_avg_temp'])
df_ct_temps['ct_date']=pd.to_datetime(df_ct_temps['ct_date'])
df_ct_prcp = pd.DataFrame(list(zip(ct_dates_prcp, ct_prcp)), columns =␣
↪['ct_date', 'ct_prcp'])
df_ct_prcp['ct_date']=pd.to_datetime(df_ct_prcp['ct_date'])
```

[23]:
```
#Since we already have avg_temp for CT, inner join with prcp:
DF_ct = pd.merge(df_ct_temps, df_ct_prcp, how = 'inner', on= ['ct_date'])
```

5.0.3 Maine:

Note: While Connecticut includes TAVG (Temp *Average*), the remaining three states do not. They do, however, include TMAX and TMIN. So we'll use those values to calculate the respective TAVG per remaining states.

[24]:
```
#Maine:
df_me_temp_min = pd.DataFrame(list(zip(me_dates_temp_min, me_temps_min)),␣
↪columns = ['me_date', 'me_temp_min'])
df_me_temp_min['me_date'] = pd.to_datetime(df_me_temp_min['me_date'])
df_me_temp_max = pd.DataFrame(list(zip(me_dates_temp_max, me_temps_max)),␣
↪columns = ['me_date', 'me_temp_max'])
df_me_temp_max['me_date'] = pd.to_datetime(df_me_temp_max['me_date'])
df_me_prcp = pd.DataFrame(list(zip(me_dates_prcp, me_prcp)), columns =␣
↪['me_date', 'me_prcp'])
df_me_prcp['me_date']=pd.to_datetime(df_me_prcp['me_date'])
```

[25]:
```
#Merge min max ME temp dfs to compute avg_temp:
DF_me_temp_both = pd.merge(df_me_temp_max, df_me_temp_min, how = 'inner', on=␣
↪['me_date'])
```

[26]:
```
#Create avg_temp column for Maine:
DF_me_temp_both['me_avg_temp'] = (DF_me_temp_both['me_temp_min'] +␣
↪DF_me_temp_both['me_temp_max']) // 2
#Drop min max temp columns:
DF_me_temp_avg = DF_me_temp_both.drop(columns=['me_temp_max', 'me_temp_min'])
```

[27]:
```
#Merge temp df with prcp df:
DF_me = pd.merge(DF_me_temp_avg, df_me_prcp, how = 'inner', on= ['me_date'])
```

[28]:
```

```

```
#Let's double check that the missing data made it into our Maine DataFrame␣
 ↪(we'll just check a small portion):
DF_me[(DF_me.me_date > '2020-11-26') & (DF_me.me_date < '2020-12-03')]
```

[28]:

	me_date	me_avg_temp	me_prcp
330	2020-11-27	44	0
331	2020-11-28	19	8
332	2020-11-29	19	0
333	2020-11-30	53	503
334	2020-11-30	53	503
335	2020-12-01	97	361
336	2020-12-02	17	5

5.0.4 Vermont:

[29]:
```
#Vermont:
df_vt_temp_min = pd.DataFrame(list(zip(vt_dates_temp_min, vt_temps_min)),␣
 ↪columns = ['vt_date', 'vt_temp_min'])
df_vt_temp_min['vt_date'] = pd.to_datetime(df_vt_temp_min['vt_date'])
df_vt_temp_max = pd.DataFrame(list(zip(vt_dates_temp_max, vt_temps_max)),␣
 ↪columns = ['vt_date', 'vt_temp_max'])
df_vt_temp_max['vt_date'] = pd.to_datetime(df_vt_temp_max['vt_date'])
df_vt_prcp = pd.DataFrame(list(zip(vt_dates_prcp, vt_prcp)), columns =␣
 ↪['vt_date', 'vt_prcp'])
df_vt_prcp['vt_date']=pd.to_datetime(df_vt_prcp['vt_date'])
```

[30]:
```
#Merge min max VT temp dfs to compute avg_temp:
DF_vt_temp_both = pd.merge(df_vt_temp_max, df_vt_temp_min, how = 'inner', on=␣
 ↪['vt_date'])
```

[31]:
```
#Create avg_temp column for Vermont:
DF_vt_temp_both['vt_avg_temp'] = (DF_vt_temp_both['vt_temp_min'] +␣
 ↪DF_vt_temp_both['vt_temp_max']) // 2
#Drop min max temp columns:
DF_vt_temp_avg = DF_vt_temp_both.drop(columns=['vt_temp_max', 'vt_temp_min'])
```

[32]:
```
#Merge temp df with prcp df:
DF_vt = pd.merge(DF_vt_temp_avg, df_vt_prcp, how = 'inner', on= ['vt_date'])
```

5.0.5 Massachusetts

[33]:
```
#Massachusetts
df_ma_temp_min = pd.DataFrame(list(zip(ma_dates_temp_min, ma_temps_min)),␣
 ↪columns = ['ma_date', 'ma_temp_min'])
df_ma_temp_min['ma_date'] = pd.to_datetime(df_ma_temp_min['ma_date'])
df_ma_temp_max = pd.DataFrame(list(zip(ma_dates_temp_max, ma_temps_max)),␣
 ↪columns = ['ma_date', 'ma_temp_max'])
```

```
df_ma_temp_max['ma_date'] = pd.to_datetime(df_ma_temp_max['ma_date'])
df_ma_prcp = pd.DataFrame(list(zip(ma_dates_prcp, ma_prcp)), columns =␣
↪['ma_date', 'ma_prcp'])
df_ma_prcp['ma_date']=pd.to_datetime(df_ma_prcp['ma_date'])
```

[34]:
```
#Merge min max MA temp dfs to compute avg_temp:
DF_ma_temp_both = pd.merge(df_ma_temp_max, df_ma_temp_min, how = 'inner', on=␣
↪['ma_date'])
```

[35]:
```
#Create avg_temp column for Massachusetts:
DF_ma_temp_both['ma_avg_temp'] = (DF_ma_temp_both['ma_temp_min'] +␣
↪DF_ma_temp_both['ma_temp_max']) // 2
#Drop min max temp columns:
DF_ma_temp_avg = DF_ma_temp_both.drop(columns=['ma_temp_max', 'ma_temp_min'])
```

[36]:
```
#Merge MA temp df with MA prcp df:
DF_ma = pd.merge(DF_ma_temp_avg, df_ma_prcp, how = 'inner', on= ['ma_date'])
```

5.1 Merge to one DataFrame:

[37]:
```
#Merge states to same df:
df_vt_me = pd.merge(DF_vt, DF_me, how = 'left', left_on= ['vt_date'], right_on=␣
↪['me_date'])
df_vt_me_ct= pd.merge(df_vt_me, DF_ct, how = 'left', left_on= ['vt_date'],␣
↪right_on= ['ct_date'])
df_weather = pd.merge(df_vt_me_ct, DF_ma, how = 'left', left_on= ['vt_date'],␣
↪right_on= ['ma_date'])
```

[38]:
```
#Drop replicate 'date' columns:
df_weather2 = df_weather.drop(columns=['me_date', 'ct_date', 'ma_date'])
df_weather2
```

[38]:

	vt_date	vt_avg_temp	vt_prcp	me_avg_temp	me_prcp	ct_avg_temp	ct_prcp	ma_avg_temp	ma_prcp
0	2020-01-01	11	3	-38.0	5.0	19.0	0.0	22.0	0.0
1	2020-01-02	14	0	-11.0	0.0	21.0			
2	2020-01-03	50	0	11.0	3.0	56.0			
3	2020-01-04	17	51	9.0	20.0	46.0			
4	2020-01-05	-49	0	-63.0	28.0	25.0			
..	…	…	…	…	…	…			
672	2021-10-30	97	109	53.0	30.0	107.0			
673	2021-10-31	114	406	100.0	523.0	151.0			
674	2021-11-01	94	0	55.0	0.0	122.0			
675	2021-11-02	64	13	47.0	0.0	79.0			
676	2021-11-03	30	76	NaN	NaN	NaN			

```
1        0.0         30.0      0.0
2        0.0         69.0      0.0
3       56.0         67.0     33.0
4        0.0         36.0     71.0
..       …           …       …
672    155.0        125.0    239.0
673     74.0        161.0    157.0
674      0.0        114.0      0.0
675      0.0         77.0      0.0
676      NaN         61.0      0.0

[677 rows x 9 columns]
```

5.2 Address NaNs

[39]:
```
#Chack for total NaNs
df_weather2.isna().sum()
```

[39]:
```
vt_date        0
vt_avg_temp    0
vt_prcp        0
me_avg_temp    2
me_prcp        2
ct_avg_temp    1
ct_prcp        1
ma_avg_temp    2
ma_prcp        2
dtype: int64
```

[40]:
```
#Check for MA NaNs first, since MA has the most:
df_weather2[df_weather2['ma_avg_temp'].isna()]
```

[40]:
```
       vt_date  vt_avg_temp  vt_prcp  me_avg_temp  me_prcp  ct_avg_temp  \
646 2021-10-04          153        0        105.0      0.0        152.0
654 2021-10-12          183        0        164.0      0.0        152.0

     ct_prcp  ma_avg_temp  ma_prcp
646    460.0          NaN      NaN
654      0.0          NaN      NaN
```

It looks like, of Massachusetts' four total missing values, three of those values are 2021-09-23 or after. It looks like Maine and Connecticut are also missing values after 2021-09-23, so I'll just cut the data at 2021-09-22.

[41]:
```
#Cut off data at 2021-09-22:
df_weather3 = df_weather2.drop(df_weather2.index[635:642])
```

[42]:
```
df_weather3
```

[42]:

	vt_date	vt_avg_temp	vt_prcp	me_avg_temp	me_prcp	ct_avg_temp	\
0	2020-01-01	11	3	-38.0	5.0	19.0	
1	2020-01-02	14	0	-11.0	0.0	21.0	
2	2020-01-03	50	0	11.0	3.0	56.0	
3	2020-01-04	17	51	9.0	20.0	46.0	
4	2020-01-05	-49	0	-63.0	28.0	25.0	
..	…	…	…	…	…	…	
672	2021-10-30	97	109	53.0	30.0	107.0	
673	2021-10-31	114	406	100.0	523.0	151.0	
674	2021-11-01	94	0	55.0	0.0	122.0	
675	2021-11-02	64	13	47.0	0.0	79.0	
676	2021-11-03	30	76	NaN	NaN	NaN	

	ct_prcp	ma_avg_temp	ma_prcp
0	0.0	22.0	0.0
1	0.0	30.0	0.0
2	0.0	69.0	0.0
3	56.0	67.0	33.0
4	0.0	36.0	71.0
..	…	…	…
672	155.0	125.0	239.0
673	74.0	161.0	157.0
674	0.0	114.0	0.0
675	0.0	77.0	0.0
676	NaN	61.0	0.0

[670 rows x 9 columns]

Since there's only one remaining NaN for MA, one for CT, and 2 for ME, instead of dropping them (and creating a gap in the time series flow), it may be worth checking if the NOAA'Climate Data Online Search' tool can't turn up any weather data available for a nearby weather station from these few dates. The interactive map feature should make it easy to quickly (visually) determine the nearest weather stations to our original one. First, check the dates we need to find values for in the CT and ME data sets.

[43]:
```
#For MA, we need 2021-09-11:
r = requests.get('https://www.ncdc.noaa.gov/cdo-web/api/v2/data?
 ↪datasetid=GHCND&datatypeid=PRCP&&datatypeid=TMAX&datatypeid=TMIN&limit=1000&stationid=GHCND
 ↪USC00194744&startdate=2021-09-11&enddate=2021-09-11', headers={'token':
 ↪Token})
d = json.loads(r.text)
print(d)
```

```
{'metadata': {'resultset': {'offset': 1, 'count': 3, 'limit': 1000}}, 'results':
[{'date': '2021-09-11T00:00:00', 'datatype': 'PRCP', 'station':
'GHCND:USC00194744', 'attributes': ',,7,0700', 'value': 3}, {'date':
'2021-09-11T00:00:00', 'datatype': 'TMAX', 'station': 'GHCND:USC00194744',
'attributes': ',,7,0700', 'value': 217}, {'date': '2021-09-11T00:00:00',
```

```
'datatype': 'TMIN', 'station': 'GHCND:USC00194744', 'attributes': ',,7,0700',
'value': 94}]}
```

[44]:
```
#Calculate TAVG:
ma_0911_tavg = (94 + 217)/2
#replace nan values from 2021-08-20
df_weather3.iloc[623,7] = ma_0911_tavg
df_weather3.iloc[623,8] = 3.0
```

Final check to make sure MA doesn't have any more NaNs:

[45]:
```
df_weather3[df_weather3['ma_avg_temp'].isna()]
```

[45]:

	vt_date	vt_avg_temp	vt_prcp	me_avg_temp	me_prcp	ct_avg_temp	ct_prcp	ma_avg_temp	ma_prcp
646	2021-10-04	153	0	105.0	0.0	152.0	460.0	NaN	NaN
654	2021-10-12	183	0	164.0	0.0	152.0	0.0	NaN	NaN

[46]:
```
#Now check missing values of ME:
df_weather3[df_weather3['me_avg_temp'].isna()]
```

[46]:

	vt_date	vt_avg_temp	vt_prcp	me_avg_temp	me_prcp	ct_avg_temp	ct_prcp	ma_avg_temp	ma_prcp
250	2020-09-07	194	0	NaN	NaN	217.0	0.0	222.0	0.0
676	2021-11-03	30	76	NaN	NaN	NaN	NaN	61.0	0.0

[47]:
```
#Find tavg and prcp from nearby ME weather station for 2020-09-07
r = requests.get('https://www.ncdc.noaa.gov/cdo-web/api/v2/data?
 ↪datasetid=GHCND&datatypeid=PRCP&&datatypeid=TMAX&datatypeid=TMIN&limit=1000&stationid=GHCND
 ↪USC00170814&startdate=2020-09-07&enddate=2020-09-07', headers={'token':
 ↪Token})
d = json.loads(r.text)
print(d)
```

```
{'metadata': {'resultset': {'offset': 1, 'count': 3, 'limit': 1000}}, 'results':
[{'date': '2020-09-07T00:00:00', 'datatype': 'PRCP', 'station':
'GHCND:USC00170814', 'attributes': ',,7,0700', 'value': 10}, {'date':
'2020-09-07T00:00:00', 'datatype': 'TMAX', 'station': 'GHCND:USC00170814',
'attributes': ',,7,0700', 'value': 244}, {'date': '2020-09-07T00:00:00',
'datatype': 'TMIN', 'station': 'GHCND:USC00170814', 'attributes': ',,7,0700',
'value': 44}]}
```

[48]:
```
#Calculate ME TAVG for 2020-09-07 (all temperatures will be converted to whole␣
↪Fahrenheit units next):
me_tavg_0907 = (244 + 44)/2
#Replace nan temp value as float
df_weather3.iloc[250, 3] = me_tavg_0907
#replace nan prcp value as float
df_weather3.iloc[250,4] = 10.0
```

[49]:
```
#Quadruple check for missing values:
df_weather3[pd.isnull(df_weather3).any(axis=1)]
```

[49]:

	vt_date	vt_avg_temp	vt_prcp	me_avg_temp	me_prcp	ct_avg_temp	ct_prcp	ma_avg_temp	ma_prcp
646	2021-10-04	153	0	105.0	0.0	152.0	460.0	NaN	NaN
654	2021-10-12	183	0	164.0	0.0	152.0	0.0	NaN	NaN
676	2021-11-03	30	76	NaN	NaN	NaN	NaN	61.0	0.0

No more missing values! Now for the COVID19 dataset.

5.3 COVID data:

COVID19 data source: "COVID-19 Data Repository by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University" or "JHU CSSE COVID-19 Data": link here.

[50]:
```
#Import COVID dataset:
cov = pd.read_csv('time_series_covid19_confirmed_US.csv')
```

[51]:
```
#Groupby states
cov_all_states = cov.groupby('Province_State').sum().reset_index()
```

[52]:
```
#Select only relevant states
cov_state = cov_all_states[cov_all_states['Province_State'].
↪isin(['Massachusetts', 'Connecticut', 'Maine', 'Vermont'])].
↪reset_index(drop=True)
```

[53]:
```
#Drop unnecessary columns and transpose
cov_state2 = cov_state.drop(columns=['UID', 'code3', 'FIPS', 'Lat', 'Long_']).T
```

[54]:
```
# Check NaNs
is_NaN = cov_state2.isnull()
row_has_NaN = is_NaN.any(axis=1)
rows_with_NaN = cov_state2[row_has_NaN]
print(rows_with_NaN)
```

```
Empty DataFrame
Columns: [0, 1, 2, 3]
Index: []
```

[55]:
```
#Replace Headers with first row
new_header = cov_state2.iloc[0] #grab the first row for the header
cov_state2 = cov_state2[1:] #take the data less the header row
cov_state2.columns = new_header #set the header row as the df header
```

[56]:
```
#Reset index
cov_state2.reset_index(inplace=True)
```

[57]:
```
#Update column names in preparation for merge with weather dataset
cov_state3=cov_state2.rename(columns={'index': 'date','Connecticut':␣
↪'CT_conf_cases', 'Maine': 'ME_conf_cases', 'Vermont': 'VT_conf_cases',␣
↪'Massachusetts' : 'MA_conf_cases'})
```

[58]:
```
#Remove index axis name
cov_state3.rename_axis('', axis=1, inplace=True)
```

[59]:
```
#Convert date column to Datetime object
cov_state3['date']=pd.to_datetime(cov_state3['date'])
```

[60]:
```
cov_state3.dtypes
```

[60]:
```
date             datetime64[ns]
CT_conf_cases            object
ME_conf_cases            object
MA_conf_cases            object
VT_conf_cases            object
dtype: object
```

5.4 Merge datasets:

[61]:
```
#Check null values
cov_state3.isna().sum()
```

[61]:
```
date             0
CT_conf_cases    0
ME_conf_cases    0
MA_conf_cases    0
VT_conf_cases    0
dtype: int64
```

[62]:
```
df_weather3[pd.isnull(df_weather3).any(axis=1)]
```

[62]:
```
        vt_date  vt_avg_temp  vt_prcp  me_avg_temp  me_prcp  ct_avg_temp  \
646 2021-10-04          153        0        105.0      0.0        152.0
654 2021-10-12          183        0        164.0      0.0        152.0
676 2021-11-03           30       76          NaN      NaN          NaN

     ct_prcp  ma_avg_temp  ma_prcp
646    460.0          NaN      NaN
654      0.0          NaN      NaN
676      NaN         61.0      0.0
```

[63]:
```
#Rename weather date column in preparation for merge
df_weather3=df_weather3.rename(columns={'vt_date':'date'})
```

[64]:
```
weather2 = df_weather3
```

[65]:
```
cov2 = cov_state3
```

[66]:
```
cov_weather = pd.merge(weather2, cov2, how = 'outer', on= 'date')
```

[67]:
```
cov_weather.iloc[624]
```

[67]:
```
date             2021-09-12 00:00:00
vt_avg_temp                      208
vt_prcp                            3
me_avg_temp                    175.0
me_prcp                          0.0
ct_avg_temp                    208.0
ct_prcp                          0.0
ma_avg_temp                    219.0
ma_prcp                          0.0
CT_conf_cases                 378933
ME_conf_cases                  80513
MA_conf_cases                 777022
VT_conf_cases                  30114
Name: 624, dtype: object
```

[68]:
```
cov_weather2 = cov_weather.iloc[21:625, :].reset_index(drop=True)
```

[69]:
```
cov_weather2
```

[69]:
```
          date  vt_avg_temp  vt_prcp  me_avg_temp  me_prcp  ct_avg_temp  \
0   2020-01-22          -28        0        -72.0      0.0        -76.0
1   2020-01-23          -27        0        -30.0      3.0        -46.0
2   2020-01-24           -8        0        -14.0      0.0          6.0
3   2020-01-25            3      119        -30.0      3.0         23.0
4   2020-01-26           22       23         11.0    132.0         49.0
..         …          …      …           …       …           …
599 2021-09-08          227      241        175.0      0.0        212.0
```

```
600 2021-09-09          197        0        169.0    353.0        219.0
601 2021-09-10          155        5        147.0    130.0        194.0
602 2021-09-11          177        0        152.0      0.0        173.0
603 2021-09-12          208        3        175.0      0.0        208.0

     ct_prcp  ma_avg_temp  ma_prcp CT_conf_cases ME_conf_cases MA_conf_cases  \
0        0.0        -53.0      0.0             0             0             0
1        0.0          0.0      0.0             0             0             0
2        0.0         19.0      0.0             0             0             0
3      218.0         33.0     25.0             0             0             0
4        0.0         64.0    193.0             0             0             0
..       …           …       …               …             …             …
599      0.0        222.0      0.0        377682         78803        772742
600     38.0        216.0     81.0        378308         79423        775149
601      0.0        203.0     76.0        378933         79929        777022
602      0.0        155.5      3.0        378933         80513        777022
603      0.0        219.0      0.0        378933         80513        777022

    VT_conf_cases
0               0
1               0
2               0
3               0
4               0
..            …
599         29436
600         29588
601         29735
602         29934
603         30114

[604 rows x 13 columns]
```

[70]:
```
#Any remaining missing values?
cov_weather2.isna().sum()
```

[70]:
```
date             0
vt_avg_temp      0
vt_prcp          0
me_avg_temp      0
me_prcp          0
ct_avg_temp      0
ct_prcp          0
ma_avg_temp      0
ma_prcp          0
CT_conf_cases    0
ME_conf_cases    0
```

```
MA_conf_cases    0
VT_conf_cases    0
dtype: int64
```

[71]:
```
cov_weather2[pd.isnull(cov_weather2).any(axis=1)]
```

[71]:
```
Empty DataFrame
Columns: [date, vt_avg_temp, vt_prcp, me_avg_temp, me_prcp, ct_avg_temp,
ct_prcp, ma_avg_temp, ma_prcp, CT_conf_cases, ME_conf_cases, MA_conf_cases,
VT_conf_cases]
Index: []
```

[72]:
```
profile = ProfileReport(cov_weather2, title="Cov_Weather Pandas Profiling␣
↪Report", explorative=True)
```

[73]:
```
profile
```

```
Summarize dataset:   0%|          | 0/27 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]

<IPython.core.display.HTML object>
```

[73]:

[74]:
```
profile.to_file('pprofile_cov_weather.html')
```

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

[75]:
```
# save the data to a new csv file
cov_weather2.to_csv('cleaned_cov_weather4.csv', index=False)
```

[ ]: