



AOSC447\_HW\_01 (Score: 93.0 / 100.0)

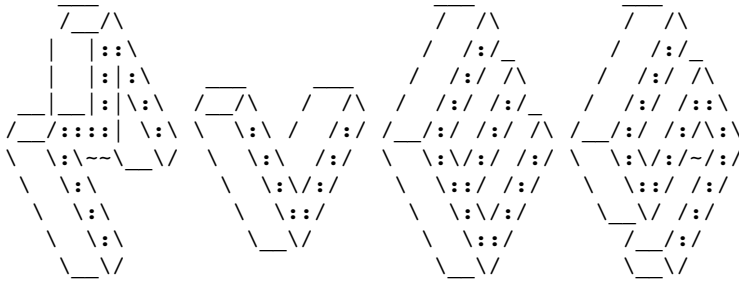
1. [Test cell](#) (Score: 10.0 / 10.0)
2. [Test cell](#) (Score: 12.0 / 12.0)
3. [Test cell](#) (Score: 11.0 / 11.0)
4. [Test cell](#) (Score: 5.0 / 5.0)
5. [Test cell](#) (Score: 8.0 / 8.0)
6. [Test cell](#) (Score: 2.0 / 3.0)
7. [Test cell](#) (Score: 7.0 / 7.0)
8. [Test cell](#) (Score: 2.0 / 6.0)
9. [Test cell](#) (Score: 8.0 / 8.0)
10. [Test cell](#) (Score: 2.0 / 5.0)
11. [Test cell](#) (Score: 10.0 / 10.0)
12. [Test cell](#) (Score: 1.0 / 2.0)
13. [Test cell](#) (Score: 5.0 / 5.0)
14. [Comment](#)
15. [Test cell](#) (Score: 10.0 / 8.0)



## Instruction, please READ ME first!

1. **Please don't download and then upload the file, EDIT IT DIRECTLY!**
2. **Please do NOT ADD OR DELETE ANY CELLS! Otherwise you may LOSE points.**
3. Before you submit your work, make sure everything runs as expected. **Restart** the kernel (in the menubar, select "Runtime" and then → "Restart and run all").
4. This assignment **takes about 3 minutes to complete**. If your code takes more than 10 minutes, you need to adjust it before submission. If your code takes more than 15 minutes, it will not be graded.
5. **Only edit cells saying "## YOUR CODE HERE". Remove all** not-implemented or in-executable codes before your submission, once you fill in your code. Otherwise you will receive a zero score for the invalid cell.
6. After you are done, you need to share it through colab (input our emails again, even though mine is listed). This just notifies us   , it will NOT submit your work to the grading system.
7. To submit, complete the survey in the last cell, and you will get a confirmation email for a successful submission.
8. Don't hesitate to restart the kernel and run all cells. This will **not** accidentally submit it multiple times.
9. You are encouraged to write more comments for your codes and add function headers to increase readability. For the same reason, each line is limited to 79 characters.

Make sure you fill in any place that says YOUR CODE HERE or "YOUR ANSWER HERE".



## AOSC447: Machine Learning in Earth Science

Spring 2021

### Prof. Xin-Zhong Liang

You will use USEPA's Air Quality System (AQS) database to put your knowledge and understanding learned in classroom into practice of how to use Pandas for data preparation and manipulation and also to visualize your results. Please write Python codes to do the following tasks:

In [1]:

```
##@title Click me before you start! Don't edit me! {display-mode: "form"}
import this
import time
import getpass
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from os import path
from subprocess import check_call
from IPython.core.magic import register_cell_magic

@register_cell_magic
def playground(line, cell):
    if eval(line):
        get_ipython().ex(cell)
    else:
        return

tot_start = time.time()
user = getpass.getuser()
student = True if user != "easm" else False

if student: # It's you!
    cmd = "wget --no-check-certificate 'https://docs.google.com/uc?export=download&id=1tEIOLG0urw2R3rvf0EvkBxhV3oVLxob6' -O mles.cpython-36m-x86_64-linux-gnu.so"
    check_call(cmd, shell=True)
    cmd = "wget --no-check-certificate 'https://docs.google.com/uc?export=download&id=1hSCuvWlkhc_LgqTWGrFH2cyNdIcGw-3h' -O states_ref.pkl"
    check_call(cmd, shell=True)

else: #this is CS
    import os
    if not os.path.islink("mles.cpython-38-darwin.so"):
        os.symlink("/Users/easm/HW01/mles.cpython-38-darwin.so", "./mles.cpython-38-darwin.so")
    if not os.path.islink("states_ref.pkl"):
        os.symlink("/Users/easm/HW01/states_ref.pkl_mac", "./states_ref.pkl")

import mles
```

Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.  
Special cases aren't special enough to break the rules.  
Although practicality beats purity.  
Errors should never pass silently.  
Unless explicitly silenced.  
In the face of ambiguity, refuse the temptation to guess.  
There should be one-- and preferably only one --obvious way to do it.  
Although that way may not be obvious at first unless you're Dutch.  
Now is better than never.  
Although never is often better than \*right\* now.  
If the implementation is hard to explain, it's a bad idea.  
If the implementation is easy to explain, it may be a good idea.  
Namespaces are one honking great idea -- let's do more of those!

1. If you have less than 210 csv files, you can download this tarball from this link: [https://drive.google.com/file/d/1llePY812qe\\_8D08pIRj8B3kDOJPq0spx/view?usp=sharing](https://drive.google.com/file/d/1llePY812qe_8D08pIRj8B3kDOJPq0spx/view?usp=sharing)
2. More importantly, you can stick with the shared csv files. No need to get a full list of data if the uploading takes forever. For this task, I will grade based on your understanding not the outcome.
3. If you can not get an identical outcome, it's okay. Those are your helpers. They count only one or two points. They will guide you to achieve a correct answer, not to restrict you.
4. Don't worry about the reading data session. If you can get the correct answer, that would be great ( and don't worry if your data are slightly off). Otherwise, you can start from the question after the checkpoint.
5. For 447 students, please don't worry about how many lines in each function. That information is only to help you not to overwork. It will NOT hurt your scores.
6. Use any colors or styles, all acceptable!
7. Those tasks are independent, you can solve them in any order you prefer.
8. For seasoned programmers, try `%timeit` your function and the `mles` demo function, if you can beat any of them, please let me know! You will earn the bonus!
9. If you are sure you spot bugs in the demo function, please report me and you will also earn the bonus!

1. Task 4.1, The Julian day number equals the "day of the year".  
note for 4.1 if you use group by day+month, then I will treat your answer as correct as well. Meanwhile, if you'd like to get an identical outcome as the reference function, you need to use group by "day of year".
2. Task 4.2, see the upper right corner of the example outcome. Why is it white?
3. Task 4.2, needs to consider leap year, so assume 365 days for all years is not valid.
4. Task 4.2, use the monthly mean of daily anomaly from 4.1.
5. Task 4.3, use the monthly mean of the daily mean (original input). Sorry for the function name, which is miss leading.
6. Task 1 and Task 2, the raw data size is larger than the total RAM.  
If you are using shared CSV, some of you reported the CSV they downloaded is incompleated.

Therefore, you will not get an equal outcome for the first grading cell.

For your sake of self-checking, you can compare the subset, again I will grade those tasks based on your understanding.

In [2]:

```
%%playground student
### This is your playground, do whatever test at here!

### This is your playground, do whatever test at here!
```

## Data Description

Please download the csv files from: <https://drive.google.com/drive/folders/1WfZg-lwYW10z60u4n5GWREZN2JfPTUGa?usp=sharing>.

Put the unzipped folder into your google-driver home folder, keep the name as HW01. The total size of data is about 16G.

In [3]:

```
#Don't edit me!

if user != "easm": # this is student on colab
    from google.colab import drive
    drive.mount('/content/drive')

selected_cols = [ 'State Name', 'Parameter Name',
                  'Pollutant Standard', 'Date Local',
                  'Observation Percent', 'Arithmetic Mean', 'AQI' ]

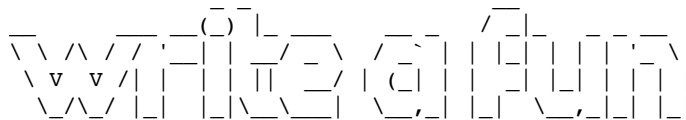
pollutants = {42101:"Carbon monoxide",
              42401:"Sulfur dioxide",
              42602:"Nitrogen dioxide (NO2)",
              44201:"Ozone",
              81102:"PM10 Total 0-10um STP",
              88101:"PM2.5 - Local Conditions" }

YB,YE=1986,2019

notstates = [ "Guam", "Virgin Islands", "Puerto Rico",
              "DC", "Country Of Mexico",
              "Canada", "District Of Columbia" ]

col_newname = { "Arithmetic Mean": "Mean",
                "Date Local": "Date",
                "State Name": "State",
                "Parameter Name": "Parameter" }
```

## Task 1



```
df_counties = readcsv(YB,YE,pollutants,rootdir = "/content/drive/MyDrive/HW01/")
```

### Args:

1. YB,YE: Begin and end of selected years (integer)
2. pollutants: dictionary contains the pollutants' information (use the above provided "pollutants")
3. rootdir: where the HW01 shared data is (default value is "/content/drive/MyDrive/HW01/")

**Returns:** list of DataFrames contains all **county-level** raw DataFrames.

### This function will:

1. Loop over all selected years
2. Loop over all pollutants in the input pollutants dictionary.
3. Read in each pollutant in each year into a DataFrame and append them into a list ( this will be returned)

### DEMO

```
dfs_counties = mles.readcsv(YB,YE,pollutants)
```

In [4]:

Student's answer

(Top)

```
import numpy as np
import pandas as pd
import os
from scipy import stats
import matplotlib.pyplot as plt
def readcsv(YB,YE,pollutants,rootdir):
    filelist = []
    for i in os.listdir(rootdir):
        for k in range(YB,YE+1):
            for l in pollutants.keys():
                if(str(k) in i):
                    if((str(l) in i)):
                        filelist.append(i)
    dfs = []

    for j in filelist:
        a = "/content/drive/MyDrive/HW01/" + j
        df = pd.read_csv(a,sep = ',',chunksize=1000)
        chunklist = []
        for chunk in df:

            dft= pd.DataFrame(chunk)
            chunklist.append(dft)

        df_concat = pd.concat(chunklist)
        dfs.append(df_concat)
    return dfs
```

In [5]:

```
%%playground student
### This is your playground, do whatever test at here!

### This is your playground, do whatever test at here!
```

In [6]:

Grade cell: cell-f1fe3984cc60c1a9

Score: 10.0 / 10.0 (Top)

```
##@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself aga
in! {display-mode: "form"}

user = getpass.getuser()

if user == "easm":
    rootdir = "/Users/easm/HW01/HW01/"
    pfname = "states_ref.pkl"
else:
    rootdir = "/content/drive/MyDrive/HW01/"
    pfname = f"/content/drive/MyDrive/states_test.pkl"
    assert path.isdir(rootdir), "Stopped, please put the shared HW01 in your gdirver first!%s"%
rootdir

if user=="easm" or (not path.exists(pfname)) : # Chao will test all
    start = time.time()
    dfs_counties = readcsv(YB,YE,pollutants,rootdir)
    end = time.time()
    print("total time %s"%(end-start))
```

```
-----
FileNotFoundError                                Traceback (most recent call last)
<ipython-input-6-515f54265a4b> in <module>
    13 if user=="easm" or (not path.exists(pfname)) : # Chao will test all
    14     start = time.time()
--> 15     dfs_counties = readcsv(YB,YE,pollutants,rootdir)
```

```

16     end = time.time()
17     print("total time %s"%(end-start))

<ipython-input-4-0f0f9d95dd48> in readcsv(YB, YE, pollutants, rootdir)
16     for j in filelist:
17         a = "/content/drive/MyDrive/HW01/" + j
--> 18         df = pd.read_csv(a, sep = ',', chunksize=1000)
19         chunklist = []
20         for chunk in df:

~/opt/anaconda3/lib/python3.8/site-packages/pandas/io/parsers.py in read_csv(filepath_or_buffer,
sep, delimiter, header, names, index_col, usecols, squeeze, prefix, mangle_dupe_cols, dtype, eng
ine, converters, true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows,
na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates,
infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates, iterator, chunksize, c
ompression, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapechar,
comment, encoding, dialect, error_bad_lines, warn_bad_lines, delim_whitespace, low_memory,
memory_map, float_precision, storage_options)
608     kwds.update(kwds_defaults)
609
--> 610     return _read(filepath_or_buffer, kwds)
611
612

~/opt/anaconda3/lib/python3.8/site-packages/pandas/io/parsers.py in _read(filepath_or_buffer,
kwds)
460
461     # Create the parser.
--> 462     parser = TextFileReader(filepath_or_buffer, **kwds)
463
464     if chunksize or iterator:

~/opt/anaconda3/lib/python3.8/site-packages/pandas/io/parsers.py in __init__(self, f, engine,
**kwds)
817         self.options["has_index_names"] = kwds["has_index_names"]
818
--> 819         self._engine = self._make_engine(self.engine)
820
821     def close(self):

~/opt/anaconda3/lib/python3.8/site-packages/pandas/io/parsers.py in _make_engine(self, engine)
1048         )
1049         # error: Too many arguments for "ParserBase"
--> 1050         return mapping[engine](self.f, **self.options) # type: ignore[call-arg]
1051
1052     def _failover_to_python(self):

~/opt/anaconda3/lib/python3.8/site-packages/pandas/io/parsers.py in __init__(self, src, **kwds)
1865
1866     # open handles
--> 1867     self._open_handles(src, kwds)
1868     assert self.handles is not None
1869     for key in ("storage_options", "encoding", "memory_map", "compression"):

~/opt/anaconda3/lib/python3.8/site-packages/pandas/io/parsers.py in _open_handles(self, src,
kwds)
1360         Let the readers open IOHandles after they are done with their potential raises.
1361         """
--> 1362         self.handles = get_handle(
1363             src,
1364             "r",

~/opt/anaconda3/lib/python3.8/site-packages/pandas/io/common.py in get_handle(path_or_buf, mode,
encoding, compression, memory_map, is_text, errors, storage_options)
640         errors = "replace"
641         # Encoding
--> 642         handle = open(
643             handle,
644             ioargs.mode,

FileNotFoundError: [Errno 2] No such file or directory:
'/content/drive/MyDrive/HW01/daily_42401_1997.csv'

```

## Task 2: Prepare Data

# Writio Gifman

```
df_states_student = Prepare_State_df(dfs,select_col,notstates,col_newname,pfname=None)
```

## Args:

1. dfs: list county-level DataFrames from the last cell
2. select\_col: list of columns you would like to keep
3. notstates: list of strings, those are not states
4. col\_newname: a dictionary of mapping between new and old names
5. pfname: if provided, it will write out the pickle file

**Returns:** A **DataFrame** contains all **state-mean** pollution information..

## This function will:

1. Pickup the selected columns
2. Drop the records of pollutant standards: "CO 1-hour 1971" and "SO2 3-hour 1971".
3. Drop the records with "Observation Percent" smaller than 20.
4. Drop the records with "State Name" in the list of notstates.
5. Drop the whole column of "Observation Percent".
6. Reset index.
7. Rename the columns using the dictionary above (col\_newname).
8. Change types of data "State"->'category', "Parameter"->'category'.
9. **First** Concatenate all DataFrames into a big one. How to do it without bust your memory??? That's the key to this problem!
10. **Then** Calculate daily means at the state level, that is, averaged over all the counties within each state. Do that for all states and put these daily records into a single Pandas DataFrame structure, named "df\_states\_student".
11. Write out the DataFrame per user request (if provided pfname). It includes all necessary information so that you can load them to recover the whole DataFrame.

## Note:

1. The reason to calculate the state-level means at the final step is to make sure counties in different files would be averaged properly. This will not happen for this particular data, but this can happen in your future work.
2. Use for loop less than three times, code amount should be less than 25 lines.
3. Execution time should be less than 15 minutes.

## DEMO

```
df_states_student =  
mles.Prepare_State_df(dfs_counties,selected_cols,notstates,col_newname,pfname)
```

In [7]:

Student's answer

(Top)



```
def Prepare_State_df(dfs,select_col,notstates,col_newname,pfname=None):
    dfl = []
    for i in dfs:
        dfl[i] = dfs[select_col]
        dfl[i][dfl[i]['Pollutant Standard']!='CO 1-hour 1971']
        dfl[i][dfl[i]['Pollutant Standard'] != 'SO2 3-hour 1971']
        dfl[i][not[dfl[i]['State Name'] in notstates]]
        dfl[i][not[dfl[i]['Observation Percent']<20]]
        dfl[i].drop(['Observation Percent'], axis = 1)
        dfl[i].reset_index()
        dfl[i].rename(columns = col_newname, inplace = True)
        for m in ['State', 'Parameter']:
            dfl[m] = dfl[m].astype('category')
    df_states1 = pd.concat(dfl,axis = 1)
    a =df_states1.groupby(['Date','State','Parameter'])['AQI'].mean().reset_index()
    b =df_states1.groupby(['Date','State','Parameter'])['Mean'].mean().reset_index()
    a['Mean'] = b['Mean']
    if(pfname == True):
        print(a)
    df_states_student = a
    return df_states_student
```

In [8]:

```
%%playground student
### This is your playground, do whatever test at here!

### This is your playground, do whatever test at here!
```

In [9]:

Grade cell: cell-e9b30656ccdb5fb9

Score: 12.0 / 12.0 (Top)

```
##@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself aga
in! {display-mode: "form"}

with mles.clock_ticking(1000):
    if user=="easm" or (not path.exists(pfname)) : # Chao will test all
        start = time.time()
        df_states_student = Prepare_State_df(dfs_counties,selected_cols,notstates,col_newname,pf
name)
        end = time.time()
        print("total time %s"%(end-start))
        print("total rows%s"%len(df_states_student.index))
    else:
        df_states_student = pd.read_pickle(pfname)
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-9-dd076d877b02> in <module>
      4     if user=="easm" or (not path.exists(pfname)) : # Chao will test all
      5         start = time.time()
----> 6         df_states_student = Prepare_State_df(dfs_counties,selected_cols,notstates,col_new
name,pfname)
      7         end = time.time()
      8         print("total time %s"%(end-start))

NameError: name 'dfs_counties' is not defined
```

In [10]:

Grade cell: cell-0478ae57545ee625

Score: 11.0 / 11.0 (Top)

```
##@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself aga
in! {display-mode: "form"}
mles.checkit(Prepare_State_df,25,3)
```



```

AssertionError                                Traceback (most recent call last)
<ipython-input-10-ac2f7be15bc6> in <module>
      1 #@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself again! {display-mode: "form"}
----> 2 mles.checkit(Prepare_State_df,25,3)

mles.pyx in mles.checkit()

AssertionError: Your fun has too long line! Each line should be less than 80 characters!

```

## **\*\*Checkpoint\*\***

Until this point, you are supposed to get an identical DataFrame as the example below. Here is the head of the df\_states:

	Date	State	Parameter	Mean	AQI
0	1986-01-01	Alabama	Carbon monoxide	1.01447	39.25
1	1986-01-01	Alabama	PM10 Total 0-10um STP	40	37
2	1986-01-01	Alabama	Sulfur dioxide	6.75198	32.25
3	1986-01-01	Alaska	Carbon monoxide	0.75329	12.75
4	1986-01-01	Alaska	PM10 Total 0-10um STP	17.5	16.5

No matter what your previous outcome is, for the following task, you will all use this example DataFrame: df\_states to complete your assignment. In our later discussion, we will use features to describe the names of each column.

In [11]:

```
df_states = pd.read_pickle("states_ref.pkl")
```

In [12]:

Grade cell: `cell-9182d1b73a252012` Score: 5.0 / 5.0 (Top)

```

#@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself again! {display-mode: "form"}
assert np.allclose(df_states.loc[:,["Mean","AQI"]].values,
                    df_states_student.loc[:,["Mean","AQI"]].values,equal_nan=True)
assert (df_states.columns==df_states_student.columns).all()
assert (df_states.dtypes == df_states_student.dtypes).all()

```

```

-----
NameError                                Traceback (most recent call last)
<ipython-input-12-b305a7368fee> in <module>
      1 #@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself again! {display-mode: "form"}
      2 assert np.allclose(df_states.loc[:,["Mean","AQI"]].values,
----> 3                     df_states_student.loc[:,["Mean","AQI"]].values,equal_nan=True)
      4 assert (df_states.columns==df_states_student.columns).all()
      5 assert (df_states.dtypes == df_states_student.dtypes).all()

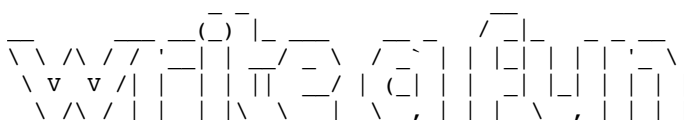
NameError: name 'df_states_student' is not defined

```

## **Task 3: Explore and Visualize Data**

Do the following data analyses and visualizations, and interpret what the results tell you in code comments.

### **Task 3.1**



```
df_mode_by_pollution = mode_by_pollution(df_states,vname,pltbar=True)
```

#### Args:

1. df\_states: DataFrame contains all daily state-level data
2. vname: feature you are going to analyze (either "AQI" or "Mean")
3. pltbar: whether plot a figure

**Returns:** A DataFrames contains each pollutant's mode value among all states' daily values.

#### This function will:

1. Calculate mode value for a specific feature
2. Draw the bar plot per user's request (default: True) whose y-axis represents mode value, the x-axis represents different pollutions.

#### Note:

1. Don't use for loop.
2. Code amount should be less than 5 lines.
3. Execution time should be less than 5 seconds

#### DEMO

```
df_mode_by_pollution_m = df_states.HW1.mode_by_pollution("AQI")
```

In [13]:

Student's answer

(Top)

```
def mode_by_pollution(df,vname:str,pltbar:bool=True):
    a = df_states.groupby(['Parameter'])[vname].apply(pd.Series.mode).reset_index().drop(['level_1'],axis = 1)
    if(pltbar == True):
        plt.bar(a['Parameter'],df_states.groupby(['Parameter'])[vname].apply(pd.Series.mode) , color='maroon', width = 0.4)
        plt.show()
    return a
```

In [14]:

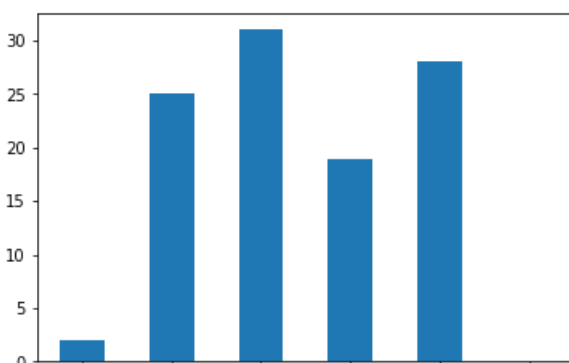
```
%%playground student
### This is your playground, do whatever test at here!

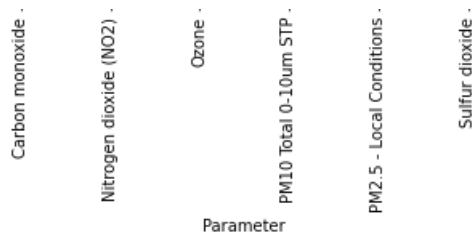
### This is your playground, do whatever test at here!
```

#### Task 3.1 grading section, **DO NOT** edit me, answer in the above cell!

In [15]:

```
##@title Demo Cell, don't edit me, check your outcome with me! {display-mode: "form"}
df_mode_by_pollution = df_states.HW1.mode_by_pollution("AQI")
```





In [16]:

```
%%playground student
### This is your playground, do whatever test at here!

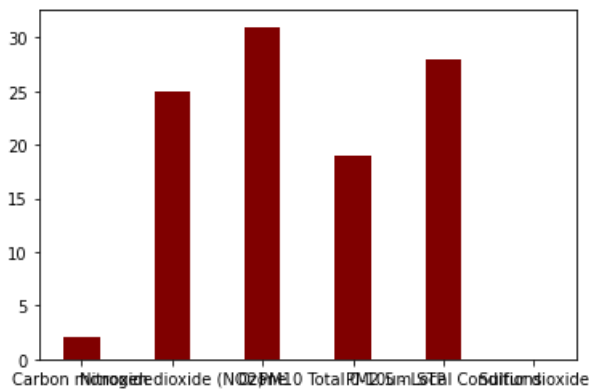
### This is your playground, do whatever test at here!
```

In [17]:

Grade cell: **cell-f2d80bf171373977**

Score: 8.0 / 8.0 (Top)

```
#@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself again! {display-mode: "form"}
with mles.clock_ticking(5):
    df_mode_by_pollution = mode_by_pollution(df_states, "AQI")
```



In [18]:

Grade cell: **cell-e492bc16927ab867**

Score: 2.0 / 3.0 (Top)

```
#@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself again! {display-mode: "form"}
assert df_states.HW1.mode_by_pollution("Mean", pltbar=False).equals( mode_by_pollution(df_states, "Mean", pltbar=False))
```

```
-----
AssertionError                                Traceback (most recent call last)
<ipython-input-18-21184b6d3ed2> in <module>
      1 #@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself again! {display-mode: "form"}
----> 2 assert df_states.HW1.mode_by_pollution("Mean", pltbar=False).equals( mode_by_pollution(df_states, "Mean", pltbar=False))

AssertionError:
```

## Task 3.2

Waiting for you

```
df_mean_by_state = mean_by_state(df_states,vname,pollutant,pltbar=True)
```

#### Args:

1. df\_states: DataFrame contains all daily state-level data
2. vname: the feature you are going to analyze (either "AQI" or "Mean")
3. pollutant: the pollutant you are going to analyze (Ozone etc.)
4. pltbar: whether Returns a figure

**Returns:** A DataFrames contains each state's mean value of specific features of selected pollution among all days.

#### This function will:

1. Calculate the mean value for a specific feature of the pollutant for each state.
2. Draw a horizontal bar plot to compare the feature of specific pollutants among all states. per user's request (default: True). Sort the bars by the mean values.

#### Note:

1. Don't use for loop.
2. The code amount should be less than 10 lines.
3. Execution time less than 5 sec

#### DEMO

```
df_mean_by_state = df_states.HW1.mean_by_state("AQI", "Ozone" ,pltbar=True)
```

In [19]:

Student's answer

(Top)

```
def mean_by_state(df_states,vname,pollutant,pltbar=True):
    temp = df_states[df_states['Parameter'] == pollutant].groupby(['State','Parameter'])[vname]
    .mean().reset_index()
    temp2 = temp.sort_values(vname,ascending = True).reset_index().drop(['index'],axis = 1)
    temp3 = temp2[vname].tolist()
    temp4 = temp2['State'].tolist()
    if(pltbar == True):
        plt.barh(temp4,temp3)
        plt.show()
    temp2 = temp2.sort_values(vname,ascending = False).reset_index().drop(['index'],axis = 1)
    return(temp2)
```

In [20]:

```
%%playground student
### This is your playground, do whatever test at here!

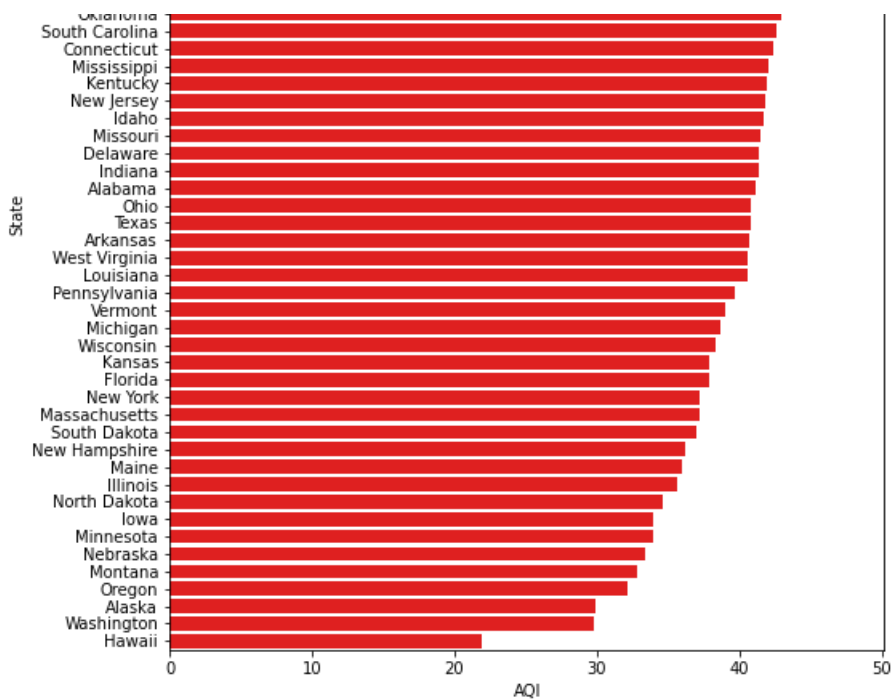
### This is your playground, do whatever test at here!
```

### Task 3.2 grading section, **DO NOT** edit me, answer in the above cell!

In [21]:

```
#@title Demo Cell, don't edit me, check your outcome with me! {display-mode: "form"}
df_mean_by_state = df_states.HW1.mean_by_state("AQI", "Ozone" ,pltbar=True)
```



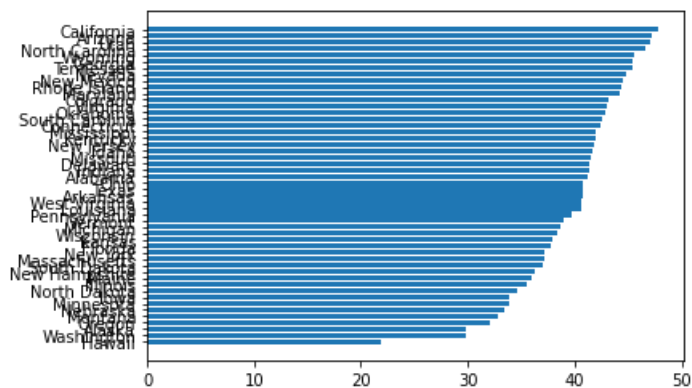


In [22]:

Grade cell: `cell-d466bace08d36807`

Score: 7.0 / 7.0 (Top)

```
##@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself again! {display-mode: "form"}
with mles.clock_ticking(5):
    df_mean_by_state = mean_by_state(df_states,"AQI","Ozone",pltbar=True)
```



In [23]:

Grade cell: `cell-ae0a2989e7692f8a`

Score: 2.0 / 6.0 (Top)

```
##@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself again! {display-mode: "form"}
assert df_states.HW1.mean_by_state("AQI", "Ozone", pltbar=False).equals(
    mean_by_state(df_states,"AQI", "Ozone", pltbar=False) )
assert df_states.HW1.mean_by_state("Mean", "Ozone", pltbar=False).equals(
    mean_by_state(df_states,"Mean", "Ozone", pltbar=False) )
```

```
-----
AssertionError                                Traceback (most recent call last)
<ipython-input-23-3fd422c58cd5> in <module>
      1 ##@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself again! {display-mode: "form"}
----> 2 assert df_states.HW1.mean_by_state("AQI", "Ozone", pltbar=False).equals(
      3                                     mean_by_state(df_states,"AQI", "Ozone", pltbar=False)
      4 assert df_states.HW1.mean_by_state("Mean", "Ozone", pltbar=False).equals(
      5                                     mean_by_state(df_states,"Mean", "Ozone", pltbar=False) )
```

)

AssertionError:

### Task 3.3

```
df_corr_between_st = corr_between_st(df_states,vname,pollutant,pltbar=True)
```

#### Args:

1. df\_states: DataFrame contains all daily state-level data
2. vname: feature you are going to analyze (either "AQI" or "Mean")
3. pollutant: the pollutant you are going to analyze (Ozone etc.)
4. pltbar: whether plot a figure

**Returns:** A DataFrames contains temporal correlations of the daily feature of a selected pollutant from all possible combinations of paired states.

#### This function will:

1. Compute temporal correlations of a specific feature from a selected pollutant among all pairs of the states.
2. Draw the correlation DataFrame on a single cluster heatmap using the range (vmin=-0.6, vmax=0.6), per user's request (default: True).

#### Note:

1. Don't use for loop.
2. The code amount should be less than 10 lines.
3. Execution time less than 5 sec

Ref lecture 12

#### DEMO

```
df_corr_between_st = df_states.HW1.corr_between_st("AQI","Ozone",pltbar=True)
```

In [24]:

Student's answer

(Top)

```
#lecture 12
def corr_between_st(df_states,vname,pollutant,pltbar=True):
    list = []
    if(vname == 'AQI'):
        list = ['Parameter','Mean']
    else:
        list = ['Parameter','AQI']
    df1 = df_states[df_states['Parameter']==pollutant].drop(columns=list)
    df2 = pd.pivot_table(df1,index=["Date"],columns = ["State"],values = [vname]).corr()
    ax = sns.heatmap(
        df2,
        vmin=-1, vmax=1, center=0,
        cmap=sns.diverging_palette(20, 220, n=200),
        square=True
    )
    ax.set_xticklabels(
        ax.get_xticklabels(),
        rotation=45,
        horizontalalignment='right'
    );
```

In [25]:

see assignment student

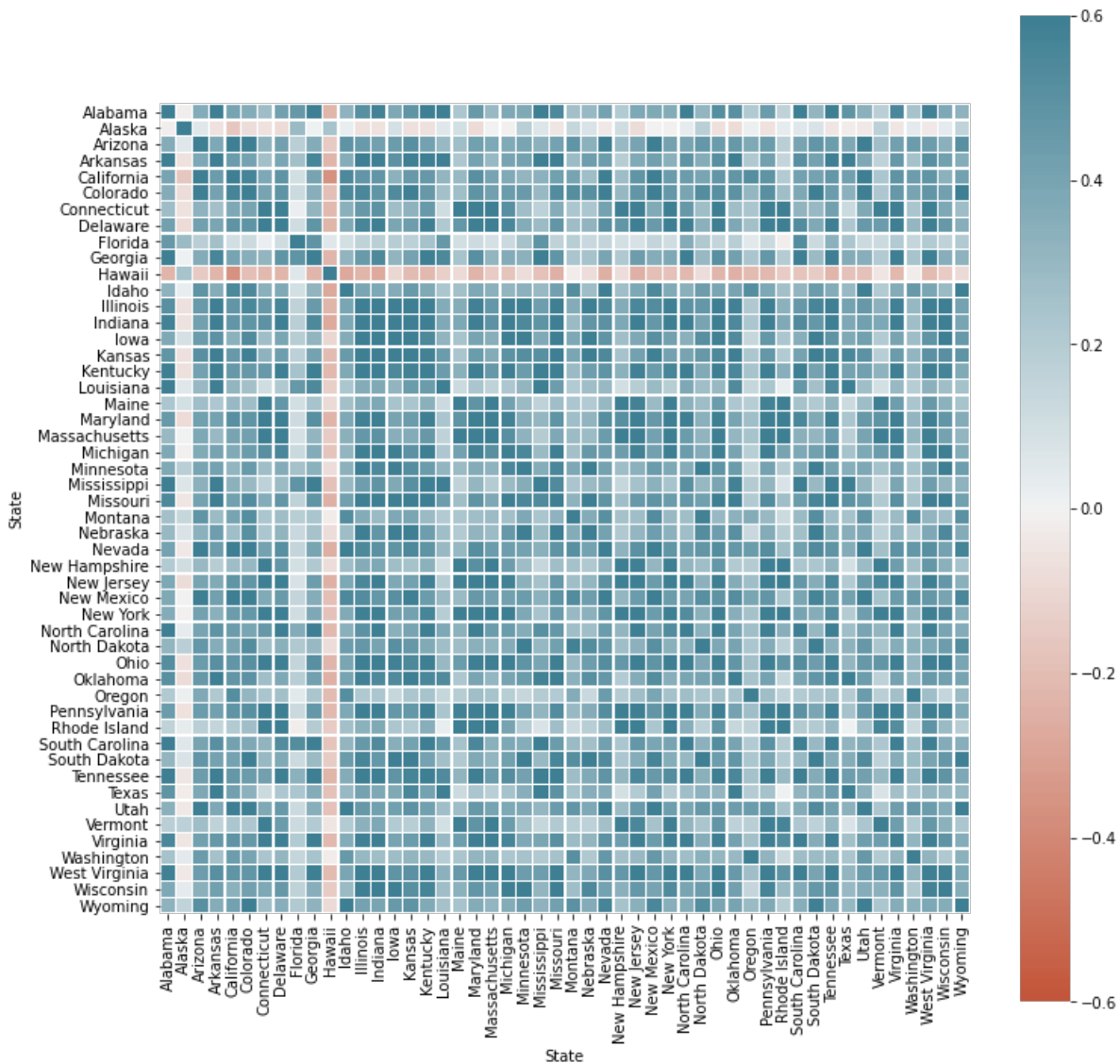
```
## playground student
### This is your playground, do whatever test at here!

### This is your playground, do whatever test at here!
```

### Task 3.3 grading section, **DO NOT** edit me, answer in the above cell!

In [26]:

```
#@title Demo Cell, don't edit me, check your outcome with me! {display-mode: "form"}
df_corr_between_st = df_states.HW1.corr_between_st("AQI", "Ozone", pltbar=True)
```

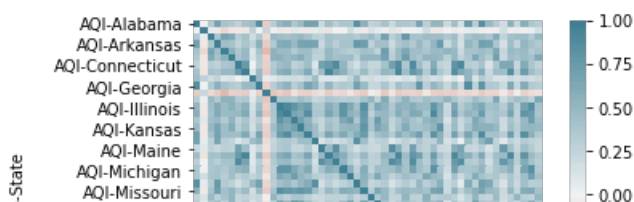


In [27]:

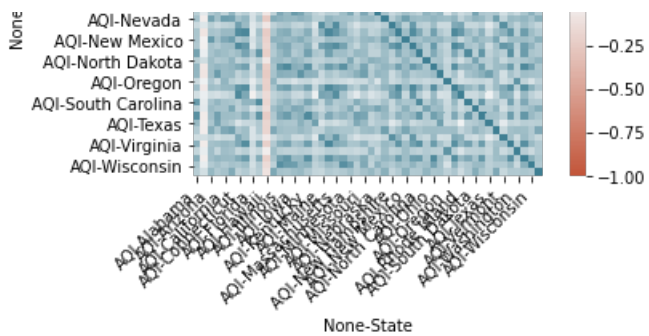
Grade cell: `cell-da0c533088c4d589`

Score: 8.0 / 8.0 (Top)

```
#@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself aga
in! {display-mode: "form"}
with mles.clock_ticking(5):
    df_corr_between_st = corr_between_st(df_states, "AQI", "Ozone", pltbar=True)
```







In [28]:

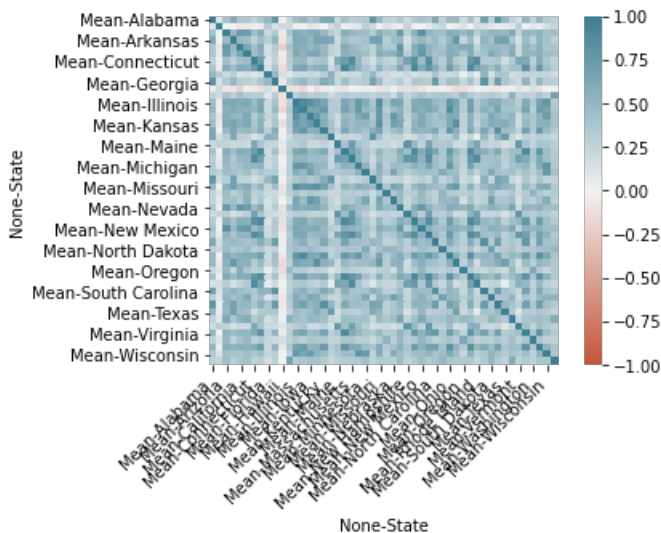
Grade cell: `cell-026875e9ee3195c3`

Score: 2.0 / 5.0 (Top)

```
##@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself again! {display-mode: "form"}
#df_corr_between_st = mles.corr_between_st(df_states,"AQI","Ozone",pltbar=True)
assert df_states.HW1.corr_between_st("Mean","Ozone",pltbar=False
).equals(corr_between_st(df_states,"Mean","Ozone",pltbar=False))
```

```
-----
AssertionError                                Traceback (most recent call last)
<ipython-input-28-bff513d697a5> in <module>
      1 ##@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself again! {display-mode: "form"}
      2 #df_corr_between_st = mles.corr_between_st(df_states,"AQI","Ozone",pltbar=True)
----> 3 assert df_states.HW1.corr_between_st("Mean","Ozone",pltbar=False
      4
      5 ).equals(corr_between_st(df_states,"Mean","Ozone",pltbar=False))
```

AssertionError:



## Task 4: Explore and Visualize Data

Do the following data and visualization analyses, and interpret what the results tell you in code comments.

### Task 4.1

```
df_anom = get_anom(df_states,pollutant,state_names,vname,pltline=True)
```

```
df_anom = get_anom(df_states,pollutant,statenames,vname,pltline=True)
```

#### Args:

1. df\_states: DataFrame contains all daily state-level data
2. pollutant: the pollutant you are going to analyze (Ozone etc.)
3. statename: the state you are going to analyze
4. vname: features to be calculated
5. pltline: whether to plot a figure (default: True)

**Returns:** A series contains the daily anomalies of a particular feature, here is the sample output:

Date	anom_daily
1986-01-01 00:00:00	2.95342
1986-01-02 00:00:00	-7.37114
1986-01-03 00:00:00	-3.53345
1986-01-04 00:00:00	-8.46124
1986-01-05 00:00:00	-2.87722

#### This function will:

1. Calculate the daily mean climatology [i.e. averaging over all years for the same calendar date July-4th] in the select state for the select pollutant.
2. Calculate the anomalies by subtracting the climatology from the raw data.
3. Per-user request, draw the line plot to compare daily original vs anomaly variations. The x-axis shall be marked with date information.

#### Note:

1. Don't use for loop.
2. The code amount should be less than 10 lines.
3. You need to properly handle the leap years.
4. Can you see the seasonality? Describe that in a code comment.
5. Daily mean climatology might be an unfamiliar conception to non-meteorology background students. You can think about each day as a unique ID (we call it Julian date) which is the number of days since the beginning of the year. Climatology daily means equal to the mean values of all the same ID days from all years, and then you subtract it from the raw data.
6. Execution time less than 5 sec.

#### DEMO

```
df_anom = df_states.HW1.get_anom("Ozone","Maryland",vname="AQI")
```

In [29]:

Student's answer

(Top)

```

from datetime import datetime
def get_anom(df_in,pollutant,statenames,vname,pltline=True):
    b = df_states[df_states['State']==statenames]
    a = b[b['Parameter'] == pollutant].groupby(['Date','State','Parameter'])[vname].mean().reset_index()
    print(a)
    a['Date'] = pd.to_datetime(a['Date'])
    a['dayofyear'] = a['Date'].dt.month.astype(str).str.zfill(2) + '-' + a['Date'].dt.day.astype(str).str.zfill(2)
    c = a.groupby(['dayofyear'])[vname].mean().reset_index()
    c.rename(columns ={'AQI':'AQI2'}, inplace = True)
    x = pd.Series(c['AQI2'].tolist(), index= c['dayofyear'].tolist())
    equiv = x.to_dict()
    a["aq2"] = a["dayofyear"].map(equiv)
    temp6 = a[vname].tolist()
    a['anom_daily'] = a[vname]-a['aq2']
    finall = a[['Date', 'anom_daily']]
    str_list = [t.strftime("%Y-%m-%d %H:%M:%S") for t in finall['Date'].tolist()]
    finall.set_index('Date',inplace = True)

    y = finall['anom_daily'].tolist()

    temp5 = [datetime.strptime(x,'%Y-%m-%d %H:%M:%S') for x in str_list]

    if(pltline == True):

        # first plot with X and Y data
        plt.plot_date(temp5, y, '-')

        # second plot with x1 and y1 data
        plt.plot_date(temp5, temp6, '-')

        plt.xlabel("Date")
        plt.show()
    return finall

```

In [30]:

```

%%playground student
### This is your playground, do whatever test at here!

### This is your playground, do whatever test at here!

```

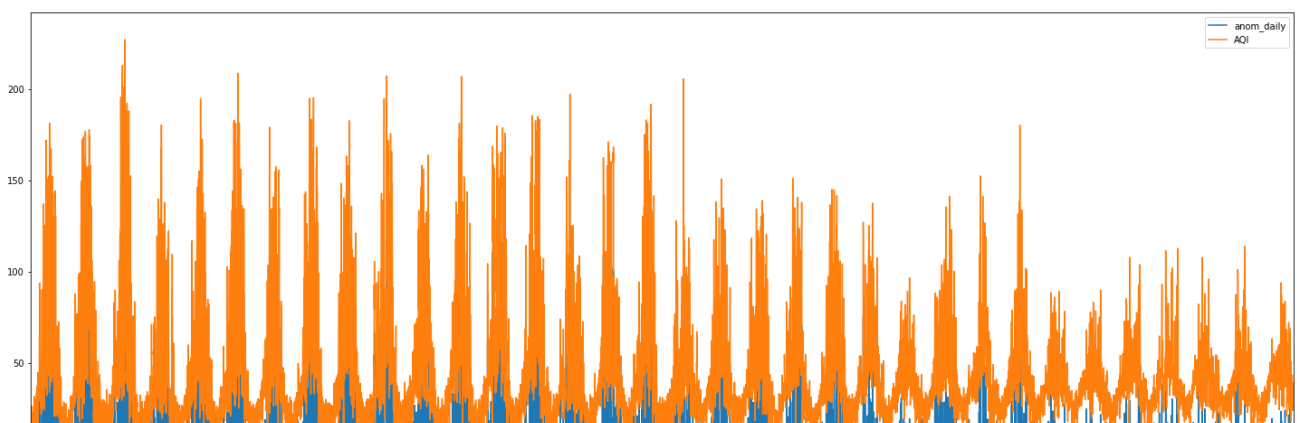
## Task 4.1 grading section, **DO NOT** edit me, answer in the above cell!

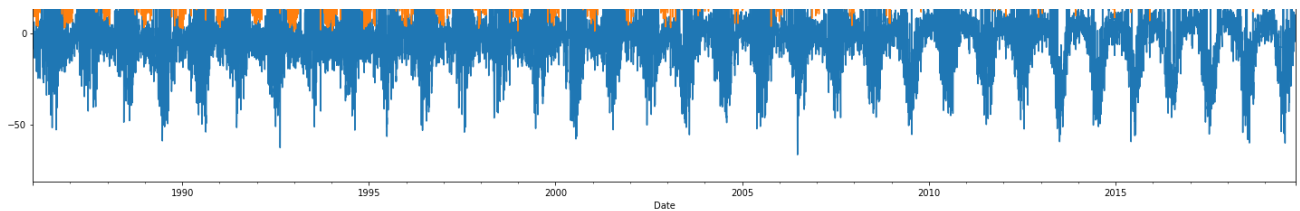
In [31]:

```

#@title Demo Cell, don't edit me, check your outcome with me! {display-mode: "form"}
df_anom = df_states.HW1.get_anom("Ozone","Maryland",vname="AQI")

```





In [32]:

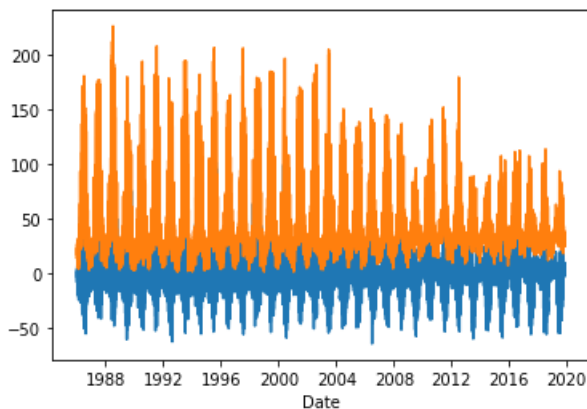
Grade cell: `cell-da2ee8f2907b4418`

Score: 10.0 / 10.0 (Top)

```
##@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself again! {display-mode: "form"}
with mles.clock_ticking(5):
    df_anom = get_anom(df_states, "Ozone", "Maryland", vname="AQI")
```

	Date	State	Parameter	AQI
0	1986-01-01	Maryland	Ozone	26.071429
1	1986-01-02	Maryland	Ozone	14.000000
2	1986-01-03	Maryland	Ozone	16.666667
3	1986-01-04	Maryland	Ozone	13.266667
4	1986-01-05	Maryland	Ozone	18.200000
...	...	...	...	...
12352	2019-10-27	Maryland	Ozone	31.375000
12353	2019-10-28	Maryland	Ozone	36.625000
12354	2019-10-29	Maryland	Ozone	30.875000
12355	2019-10-30	Maryland	Ozone	26.875000
12356	2019-10-31	Maryland	Ozone	25.000000

[12357 rows x 4 columns]



In [33]:

Grade cell: `cell-3591a20cc634cf09`

Score: 1.0 / 2.0 (Top)

```
##@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself again! {display-mode: "form"}
assert get_anom(df_states, "Ozone", "Maryland", vname="AQI", pltline=False)
        .equals(df_states.HW1.get_anom("Ozone", "Maryland", vname="AQI", pltline=False))
```

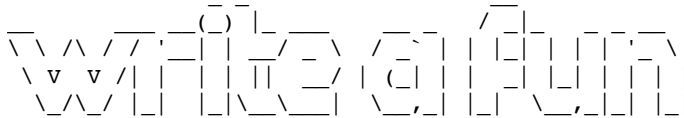
	Date	State	Parameter	AQI
0	1986-01-01	Maryland	Ozone	26.071429
1	1986-01-02	Maryland	Ozone	14.000000
2	1986-01-03	Maryland	Ozone	16.666667
3	1986-01-04	Maryland	Ozone	13.266667
4	1986-01-05	Maryland	Ozone	18.200000
...	...	...	...	...
12352	2019-10-27	Maryland	Ozone	31.375000
12353	2019-10-28	Maryland	Ozone	36.625000
12354	2019-10-29	Maryland	Ozone	30.875000
12355	2019-10-30	Maryland	Ozone	26.875000
12356	2019-10-31	Maryland	Ozone	25.000000

[12357 rows x 4 columns]

```
-----
AssertionError                                Traceback (most recent call last)
<ipython-input-33-639604fcd3cb> in <module>
      1 #@title Grading Cell, don't edit me, otherwise you would have to input all code by yourse
lf again! {display-mode: "form"}
----> 2 assert get_anom(df_states,"Ozone","Maryland",vname="AQI",pltline=False)
      3
      4 ).equals(df_states.HW1.get_anom("Ozone","Maryland",vname="AQI",pltline=False))

AssertionError:
```

## Task 4.2



```
Anom_Monthly_Contourf(df_states,pollutant,statenames,vname,clevel = range(-40,45,5))
```

### Args:

1. df\_states: DataFrame contains all daily state-level data
2. pollutant: the pollutant you are going to analyze (Ozone etc.)
3. statename: the state you are going to analysis
4. vname: features to be calculated
5. clevel: contour level

**Returns:** None

### This function will:

Draw a contoured field plot to illustrate the particular pollutant anomaly variations in terms of the month (y-axis) and year (x-axis). See sample outcome.

### Note

- Less than 10 lines of code, no for loop
- Execution time less than 5 sec

### DEMO

```
df_states.HW1.Anom_Monthly_Contourf("Ozone","Maryland","AQI")
```

In [34]:

Student's answer

(Top)

```
def Anom_Monthly_Contourf(df_states,pollutant,statename,vname,clevel = range(-40,45,5)):
    a =get_anom(df_states,pollutant,statename,vname,True).reset_index()
    a['dayofyear'] = a['Date'].dt.year.astype(str).str.zfill(2) + '-' + a['Date'].dt.month.astype(str).str.zfill(2)
    c = a.groupby(['dayofyear'])['anom_daily'].mean().reset_index()
    c['dayofyear'] = pd.to_datetime(c['dayofyear'])
    c['Year'] = c['dayofyear'].dt.year.astype(str).str.zfill(2)
    c['Month'] = c['dayofyear'].dt.month.astype(str).str.zfill(2)
    del c['dayofyear']

    df = c.pivot('Month','Year','anom_daily')

    X= np.array((df.columns.values),dtype = np.float64)

    Y=np.array((df.index.values),dtype = np.float64)

    Z=np.array((df.values),dtype = np.float64)

    x,y=np.meshgrid(X, Y)

    plt.contour(x, y, Z)
```

In [35]:

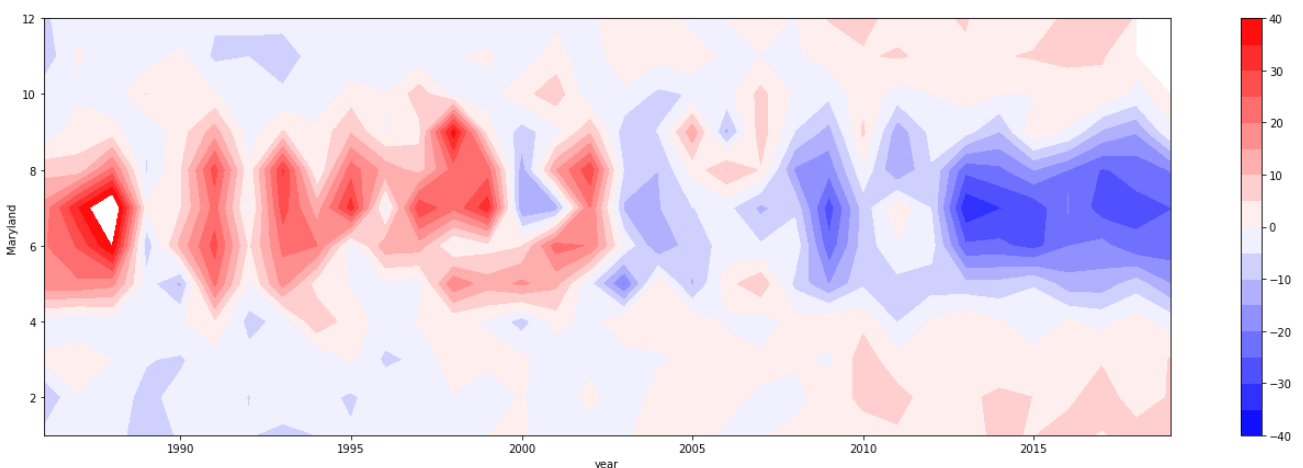
```
%%playground student
### This is your playground, do whatever test at here!

### This is your playground, do whatever test at here!
```

## Task 4.2 grading section, **DO NOT** edit me, answer in the above cell!

In [36]:

```
#@title Demo Cell, don't edit me, check your outcome with me! {display-mode: "form"}
f, ax = plt.subplots(figsize=(23, 7))
#mles.mycplot(df_anom,"Maryland" )
df_states.HW1.Anom_Monthly_Contourf("Ozone","Maryland","AQI")
```



In [37]:

```
%%playground student
### This is your playground, do whatever test at here!

### This is your playground, do whatever test at here!
```

In [38]:

Grade cell: cell-959660dafc018e6a

Score: 5.0 / 5.0 (Top)





7. Name: features to be calculated

Returns: None

This function will:

- Draw an error-bar plot whose mean value is the monthly mean of input data, and the error bar represents monthly standard deviation.
- See sample outcome

Note

- Discuss whether trends were visible.
- Less than 10 lines of code, only one for loop
- Execution time less than 5 sec

DEMO

```
df_states.HW1.Anom_Monthly_Errplot("Ozone","Maryland","AQI")
```

In [39]:

Student's answer

(Top)

```
def Anom_Monthly_Errplot(df_in,pollutant,statename,vname):
    b = df_states[df_states['State']==statename]
    a = b[b['Parameter'] == pollutant].groupby(['Date','State','Parameter'])[vname].mean().reset_index()
    a['Date'] = pd.to_datetime(a['Date'])
    a['dayofyear'] = a['Date'].dt.year.astype(str).zfill(2) + '-' + a['Date'].dt.month.astype(str).zfill(2)
    d = a[[vname,'dayofyear']].groupby('dayofyear').agg(np.std)

    c = a[[vname,'dayofyear']].groupby(['dayofyear'])[vname].mean().reset_index()

    c['dayofyear']=c['dayofyear'] + '-01'
    c['dayofyear'] = pd.to_datetime(c['dayofyear'], format='%Y/%m/%d')

    xval =np.array(c['dayofyear'])
    yval = c[vname].to_numpy()
    xerr = np.array(d[vname])

    plt.errorbar(xval, yval,xerr)
    plt.show()
```

Comments:

nice code!

In [40]:

```
%%playground student
### This is your playground, do whatever test at here!

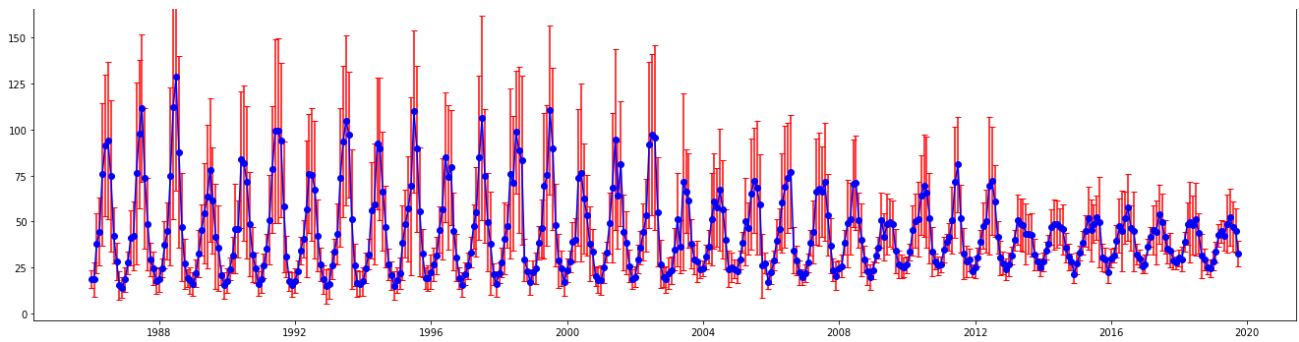
### This is your playground, do whatever test at here!
```

Task 4.3 grading section, **DO NOT** edit me, answer in the above cell!

In [41]:

```
#@title Demo Cell, don't edit me, check your outcome with me! {display-mode: "form"}

f, ax = plt.subplots(figsize=(23, 7))
#mles.myerrplot(df_anom,"Maryland")
df_states.HW1.Anom_Monthly_Errplot("Ozone","Maryland","AQI")
```



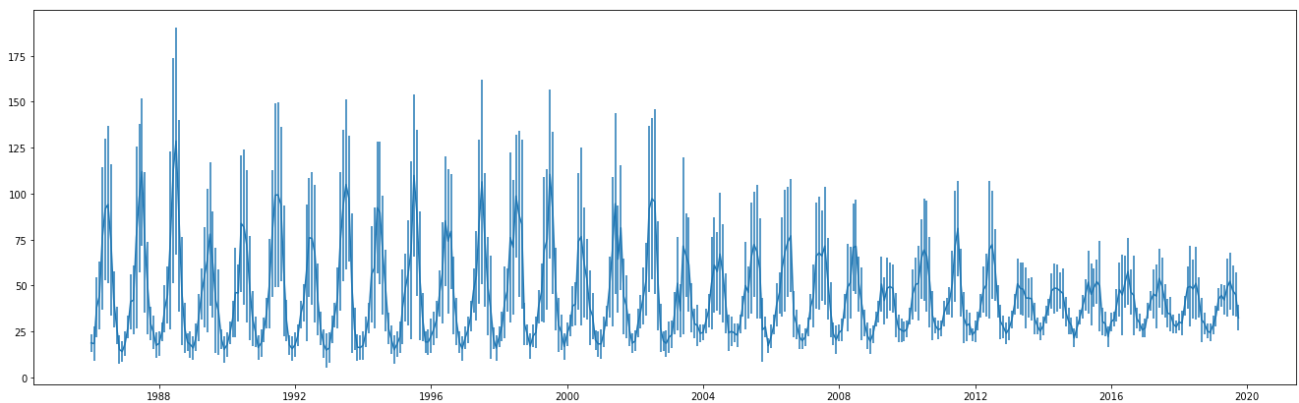
In [42]:

Grade cell: `cell-5fbff3a4c3bee3de`

Score: 10.0 / 8.0 (Top)

```
##@title Grading Cell, don't edit me, otherwise you would have to input all code by yourself again! {display-mode: "form"}
```

```
with mles.clock_ticking(5):
    f, ax = plt.subplots(figsize=(23, 7))
    Anom_Monthly_Errplot(df_states, "Ozone", "Maryland", "AQI")
```



## Submission & Survey Finally, please run the following cell and complete the survey to submit your assignment!

You will get a confirmation email after you submit it. For 447 students, the assignments number might be less, don't worry, just select the one shows up in your notebook. Thank you!

In [43]:

```
##@title RunMeAndCompleteTheSurveyToSubmit! {display-mode: "form"}
```

```
# REMOVE THIS WHEN READY TO SUBMIT
# raise NotImplementedError
```

```
from IPython.display import IFrame
IFrame(src="https://docs.google.com/forms/d/e/1FAIpQLSflpg_qbxqb54gv_RqAF3dzoAm8_g-Hw83MMmSmqob9yCvqgw/viewform?usp=sf_link", width='100%', height='500px')
```

Out[43]:

```
<IPython.lib.display.IFrame at 0x7fd90a681580>
```