AQI

May 24, 2022

[1]:
```
# Initialize Otter
import otter
grader = otter.Notebook("AQI.ipynb")
```

1 Final Project: Air Quality Dataset

1.1 Analyzing and Predicting AQI Data through Modeling

1.2 Due Date: Thursday, December 17th, 11:59 PM

1.3 Collaboration Policy

Data science is a collaborative activity. While you may talk with other groups about the project, we ask that you **write your solutions individually**. If you do discuss the assignments with others outside of your group please **include their names** at the top of your notebook.

1.4 This Assignment

In this final project, we will investigate AQI data for the year 2020 from **USA EPA** data. All the data used for this project can be accessed from the EPA Website, which we will pull from directly in this notebook. This dataset contains geographical and time-series data on various factors that contribute to AQI from all government sites. The main goal at the end for you will be to understand how AQI varies both geographically and over time, and use your analysis (as well as other data that you can find) to be predict AQI at a certain point in time for various locations in California.

Through this final project, you will demonstrate your experience with: * EDA and merging on location using Pandas * Unsupervised and supervised learning techniques * Visualization and interpolation

This is **part 1** of the project, which includes the data cleaning, guided EDA and open-ended EDA components of the project. This will help you for part 2, where you will be completing the modeling component.

[2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import re
import geopandas as gpd
```

```python
import os
import requests, zipfile, io

import warnings
warnings.filterwarnings('ignore')
```

1.5 Section 1: Data Cleaning

As mentioned, we will be using the **US EPA** data from the EPA website. Below is a dataframe of the files we will be using for the project. The following two cells will download the data and put it into a dictionary called `epa_data`.

[3]:
```python
epa_weburl = "https://web.archive.org/web/20211118232504/https://aqs.epa.gov/
↪aqsweb/airdata/"
epa_filenames = pd.read_csv("data/epa_filenames.csv")
epa_filenames
```

[3]:

	name	epa_filename
0	annual_county_aqi	annual_aqi_by_county_2020
1	daily_county_aqi	daily_aqi_by_county_2020
2	daily_ozone	daily_44201_2020
3	daily_so2	daily_42401_2020
4	daily_co	daily_42101_2020
5	daily_no2	daily_42602_2020
6	daily_temp	daily_WIND_2020
7	daily_wind	daily_TEMP_2020
8	aqs_sites	aqs_sites

Below is code that we used to extract the code from the AQI website, which we encourage you to understand! This will pull directly from the website urls and put it into your `data/` folder.

[4]:
```python
epa_data = {}
for name, filename in zip(epa_filenames['name'], epa_filenames['epa_filename']):
    path_name = 'data/{}'.format(name)
    if not os.path.isdir(path_name):
        data_url = '{}{}.zip'.format(epa_weburl, filename)
        req = requests.get(data_url)
        z = zipfile.ZipFile(io.BytesIO(req.content))
        z.extractall(path_name)
    data = pd.read_csv(f'data/{name}/{filename}.csv')
    epa_data[name] = data
```

Use the below cell to explore each of the datasets, which can be accessed using the keys in the `name` column of `epa_filenames` above. Currently, the cell is viewing the `annual_county_aqi` dataset, but feel free to change it to whichever dataset you want to explore.

1.5.1 Question 0: Understanding the Data

Notice that for the table `annual_county_aqi`, the 90th percentile AQI is reported as a column. Why would the 90th percentile AQI be useful as opposed to the maximum? What does it mean when the difference between the 90th percentile AQI and Max AQI is very large compared to the difference between the 90th percentile AQI and the median AQI?

The 90th percentile AQI would be useful as opposed to the maximum because it is more representative of the typical AQI reported in a particular state & county throughout the year. In other words, it can serve as a basic summary statistic giving the reader an idea of typical AQI values obtained in the area. The maximimum AQI on the other hand would not give readers a good idea of typical AQI values since maximum values can sometimes be outliers in distributions of values. If the difference between the 90th percentile AQI and the Max AQI is very large, then the Max AQI is not a typical value seen in the distribution of the data. It could possibly be an outlier as well. If there is a large difference between the 90th percentile AQI and the median AQI, then the range of values or spread between these percentiles is likely large as well.

1.5.2 Question 1a: Creating Month and Day Columns

In the `daily_county_aqi` table in `epa_data`, add two new columns called `Day` and `Month` that denote the day and month, respectively, of the AQI reading. The day and month should both be reported as an **integer** as opposed to a string (`Jan`, `Feb`, etc.)

hint: `pd.to_datetime` may be useful.

[5]:
```
daily_county = epa_data.get('daily_county_aqi')
daily_county['Month'] = pd.to_datetime(daily_county['Date']).apply(lambda x: x.
↪month)
daily_county['Day'] = pd.to_datetime(daily_county['Date']).apply(lambda x: x.
↪day)


daily_county.head()
```

[5]:
```
  State Name county Name  State Code  County Code        Date  AQI Category  \
0    Alabama      Baldwin           1            3  2020-01-01   48     Good
1    Alabama      Baldwin           1            3  2020-01-04   13     Good
2    Alabama      Baldwin           1            3  2020-01-07   14     Good
3    Alabama      Baldwin           1            3  2020-01-10   39     Good
4    Alabama      Baldwin           1            3  2020-01-13   29     Good

  Defining Parameter Defining Site  Number of Sites Reporting  Month  Day
0              PM2.5   01-003-0010                          1      1    1
1              PM2.5   01-003-0010                          1      1    4
2              PM2.5   01-003-0010                          1      1    7
3              PM2.5   01-003-0010                          1      1   10
4              PM2.5   01-003-0010                          1      1   13
```

[6]:
```
grader.check("q1a")
```

[6]: q1a results: All test cases passed!

1.5.3 Question 1b: California Data

Currently, `epa_data` contains data for **all** counties in the United States. For the guided part of this project, we are specifically going to be focusing on AQI data for counties in California only. Your task is to assign `epa_data_CA` a dictionary mapping table names to dataframes. This map should have the same contents as `epa_data` but only tables that contain **daily data** in the state of `California`.

[7]:
```
epa_data_CA = {}
for name in epa_filenames['name'][1:8]:
    epa_data_CA[name] = epa_data.get(name).query("`State Name` == 'California'")


epa_data_CA.get('daily_county_aqi').head()
```

[7]:
```
       State Name county Name  State Code  County Code        Date  AQI  \
14003  California     Alameda           6            1  2020-01-01   53
14004  California     Alameda           6            1  2020-01-02   43
14005  California     Alameda           6            1  2020-01-03   74
14006  California     Alameda           6            1  2020-01-04   45
14007  California     Alameda           6            1  2020-01-05   33

       Category Defining Parameter Defining Site  Number of Sites Reporting  \
14003  Moderate              PM2.5   06-001-0009                          7
14004      Good              PM2.5   06-001-0013                          7
14005  Moderate              PM2.5   06-001-0013                          7
14006      Good              PM2.5   06-001-0007                          7
14007      Good              PM2.5   06-001-0007                          7

       Month  Day
14003      1    1
14004      1    2
14005      1    3
14006      1    4
14007      1    5
```

[8]:
```
grader.check("q1b")
```

[8]: q1b results: All test cases passed!

1.5.4 Question 1c: Merging Site Information

Now take a look at this link and look under "Site ID". For later analysis, we want to first get the latitude and longitudes of each of the measurements in the `daily_county_aqi` table by merging two or more tables in `epa_data_CA` (one of the tables is `daily_county_aqi`).

Our final merged table should be assigned to `epa_data_CA_merged` and the result should contain the following columns: `State Name`, `county Name`, `Month`, `Day`, `AQI`, `Category`, `Defining Site`, `Latitude`, and `Longitude`

[9]:
```
cols = ['State Name', 'county Name', 'Month', 'Day', 'AQI', 'Category',␣
 ↪'Defining Site', 'Latitude', 'Longitude']
aqs_sites = epa_data.get('aqs_sites').drop('State Name', axis = 'columns')
#aqs_sites['State Code'] = aqs_sites['State Code'].astype(int)
daily_county_aqi = epa_data_CA.get('daily_county_aqi')

daily_county_aqi['State Code'] = daily_county_aqi['Defining Site'].str[0:2]#.
 ↪astype(int)
daily_county_aqi['County Code'] = daily_county_aqi['Defining Site'].str[3:6].
 ↪astype(int)
daily_county_aqi['Site Number'] = daily_county_aqi['Defining Site'].str[7:].
 ↪astype(int)

epa_data_CA_merged = daily_county_aqi.merge(aqs_sites,
                                        how = "left",
                                        on = ['State Code', 'County Code',␣
 ↪'Site Number'])


epa_data_CA_merged = epa_data_CA_merged[cols]
epa_data_CA_merged.head()
```

[9]:
```
  State Name county Name  Month  Day  AQI  Category Defining Site   Latitude  \
0  California     Alameda      1    1   53  Moderate   06-001-0009  37.743065
1  California     Alameda      1    2   43      Good   06-001-0013  37.864767
2  California     Alameda      1    3   74  Moderate   06-001-0013  37.864767
3  California     Alameda      1    4   45      Good   06-001-0007  37.687526
4  California     Alameda      1    5   33      Good   06-001-0007  37.687526

    Longitude
0 -122.169935
1 -122.302741
2 -122.302741
3 -121.784217
4 -121.784217
```

[10]:
```
grader.check("q1c")
```

[10]:
```
q1c results: All test cases passed!
```

1.5.5 Question 2a - Cleaning Traffic Data

Throughout this project, you will be using other datasets to assist with analysis and predictions. Traditionally, to join dataframes we need to join on a specific column with shared values. However, when it comes to locations, exact latitudes and longitudes are hard to come by since it is a continuous space. First, lets look at such a dataset that we may want to merge on with `epa_data_CA_merged`.

In the below cell, we have loaded in the `traffic_data` dataset, which contains traffic data for various locations in California. Your task is to clean this table so that it includes only the following columns (you may have to rename some): `District`, `Route`, `County`, `Descriptn`, `AADT`, `Latitude`, `Longitude`, where `AADT` is found by taking the sum of the back and ahead `AADT`s (you may run into some issues with cleaning the data in order to add these columns - `.str` functions may help with this). The metric AADT, annual average daily traffic, is calculated as the sum of the traffic north of the route (ahead AADT) and south of the route (back AADT). You also need to make sure to clean and remove any illegal values from the dataframe (hint: check `Latitude` and `Longitude`).

Hint: `str` functions you will likely use: `.strip()`, `.replace()`.

[11]:
```python
traffic_data = pd.read_csv("data/Traffic_Volumes_AADT.csv")
traffic_data_cleaned = traffic_data[['District', 'Route', 'County',␣
↪'Descriptn']]
traffic_data_cleaned['AADT'] = traffic_data['Ahead_AADT'].apply(lambda x: x.
↪strip()).replace('', 0).astype(int) + traffic_data['Back_AADT'].apply(lambda␣
↪x: x.strip()).replace('', 0).astype(int)
traffic_data_cleaned['Latitude'] = traffic_data['Lat_S_or_W']
traffic_data_cleaned['Longitude'] = traffic_data['Lon_S_or_W']
traffic_data_cleaned['Latitude'] = traffic_data_cleaned['Latitude'].
↪apply(lambda x: pd.to_numeric(x, errors='coerce'))
traffic_data_cleaned['Longitude'] = traffic_data_cleaned['Longitude'].
↪apply(lambda x: pd.to_numeric(x, errors='coerce'))
traffic_data_cleaned.dropna(inplace = True)
traffic_data_cleaned.query("41.75613 > Latitude and Latitude > 32.57816 and␣
↪Longitude > -124.20347 and Longitude < -114.60209")

traffic_data_cleaned
```

[11]:

	District	Route	County	Descriptn	AADT	\
0	1	1	MEN	SONOMA/MENDOCINO COUNTY LINE	4000	
1	1	1	MEN	NORTH LIMITS GUALALA	7100	
2	1	1	MEN	FISH ROCK ROAD	6200	
3	1	1	MEN	POINT ARENA, SOUTH CITY LIMITS	4600	
4	1	1	MEN	POINT ARENA, RIVERSIDE DRIVE	5000	
…	…	…	…	…	…	
7115	12	605	ORA	SEAL BEACH, JCT RTE 22	46100	
7116	12	605	ORA	JCT. RTE. 405	212200	
7117	12	605	ORA	LOS ALAMITOS, KATELLA AVENUE	326800	
7118	12	605	ORA	ORANGE/LOS ANGELES COUNTY LINE	170000	
7119	3	99	SAC	BREAK IN ROUTE	0	

```
      Latitude   Longitude
0     38.759843 -123.518503
1     38.770046 -123.531890
2     38.803549 -123.585411
3     38.903973 -123.691513
4     38.910913 -123.692410
…          …          …
7115  33.778633 -118.091474
7116  33.784414 -118.091768
7117  33.802799 -118.082030
7118  33.806140 -118.081547
7119  38.558838 -121.473649

[6913 rows x 7 columns]
```

[12]:
```python
grader.check("q2a")
```

[12]: q2a results: All test cases passed!

1.5.6 Question 2b - Merging on Traffic Data

Traditionally, we could employ some sort of join where we join `epa_data_CA_merged` rows with the row in `traffic_data` that it is the "closest" to, as measured by euclidean distance. As you can imagine, this can be quite tedious so instead we will use a special type of join called a **spatial join**, which can be done using the package `geopandas`, which is imported as `gpd`. The documentation for `geopandas` is linked here. Please use this as a resource to do the following tasks:

- turn `traffic_data_cleaned` and `epa_data_CA_merged` into a geopandas dataframe using the latitude and longitude.
- Use a spatial join (which function is this in the documentation?) to match the correct traffic row information to each entry in `epa_data_CA_merged`.

Your final dataframe should be assigned to `gpd_epa_traffic` with the following columns: `State Name`, `county Name`, `Month`, `Day`, `AQI`, `Category`, `Defining Site`, `Site Lat`, `Site Long`, `Traffic Lat`, `Traffic Long`, `Descriptn`, and `AADT`.

[13]:
```python
epa_data_CA_merged_gdf = gpd.GeoDataFrame(epa_data_CA_merged, geometry = gpd.
 ↪points_from_xy(x = epa_data_CA_merged.Longitude, y = epa_data_CA_merged.
 ↪Latitude))
traffic_data_cleaned_gdf = gpd.GeoDataFrame(traffic_data_cleaned, geometry =␣
 ↪gpd.points_from_xy(x = traffic_data_cleaned.Longitude, y =␣
 ↪traffic_data_cleaned.Latitude))
gpd_epa_traffic = gpd.sjoin_nearest(left_df = epa_data_CA_merged_gdf,
                                    right_df = traffic_data_cleaned_gdf,
                                    how = "inner")
```

```
gpd_epa_traffic = gpd_epa_traffic.rename(columns = {"Latitude_left" : "Site␣
↪Lat", "Longitude_left" : "Site Long", "Latitude_right" : "Traffic Lat",␣
↪"Longitude_right" : "Traffic Long"})
gpd_epa_traffic.drop(labels = ['geometry', 'index_right', 'District', 'Route',␣
↪'County'], axis = "columns")

gpd_epa_traffic.head()
```

[13]:

```
    State Name county Name  Month  Day  AQI                        Category  \
0   California     Alameda      1    1   53                        Moderate
24  California     Alameda      1   25   40                            Good
184 California     Alameda      7    3   48                            Good
185 California     Alameda      7    4  115  Unhealthy for Sensitive Groups
186 California     Alameda      7    5   78                        Moderate

   Defining Site   Site Lat   Site Long                       geometry  \
0     06-001-0009  37.743065 -122.169935  POINT (-122.16993 37.74307)
24    06-001-0009  37.743065 -122.169935  POINT (-122.16993 37.74307)
184   06-001-0009  37.743065 -122.169935  POINT (-122.16993 37.74307)
185   06-001-0009  37.743065 -122.169935  POINT (-122.16993 37.74307)
186   06-001-0009  37.743065 -122.169935  POINT (-122.16993 37.74307)

    index_right  District  Route County            Descriptn   AADT  \
0          2370         4    185    ALA  OAKLAND, 98TH AVENUE  48300
24         2370         4    185    ALA  OAKLAND, 98TH AVENUE  48300
184        2370         4    185    ALA  OAKLAND, 98TH AVENUE  48300
185        2370         4    185    ALA  OAKLAND, 98TH AVENUE  48300
186        2370         4    185    ALA  OAKLAND, 98TH AVENUE  48300

    Traffic Lat  Traffic Long
0     37.744352   -122.170586
24    37.744352   -122.170586
184   37.744352   -122.170586
185   37.744352   -122.170586
186   37.744352   -122.170586
```

[14]:

```
grader.check("q2b")
```

[14]:

```
q2b results: All test cases passed!
```

1.6 Section 2: Guided EDA

1.6.1 Question 3a: Initial AQI Analysis

Assign a `pd.Series` object to `worst_median_aqis` that contains the states with the top 10 worst median AQIs throughout the year 2020, as measured by the average median AQIs across all counties for a single state. Your result should have index `state`, the column value should be labelled `Average Median AQI`, and it should be arranged in descending order.

Now, assign the same thing to `worst_max_aqis`, except instead of aggregating the average median AQIs across all counties, aggregate the average **max AQIs** across all counties. Your result should have the same shape and labels as before, except the column value should be labelled `Average Max AQI`.

Note: you may have to remove a few regions in your tables. Make sure every entry in your output is a **US State**.

[15]:
```
anual_aqi = epa_data.get('annual_county_aqi')
worst_median_aqis = anual_aqi.groupby('State').mean().drop(['Country Of␣
↪Mexico', 'District Of Columbia', 'Puerto Rico', 'Virgin Islands']).
↪sort_values('Median AQI' , ascending = False)['Median AQI'][0:10]
worst_max_aqis = anual_aqi.groupby('State').mean().drop(['Country Of Mexico',␣
↪'District Of Columbia', 'Puerto Rico', 'Virgin Islands']).sort_values('Max␣
↪AQI' , ascending = False)['Max AQI'][0:10]


print("Worst Median AQI : \n{}\n".format(worst_median_aqis))
print("Worst Max AQI : \n{}".format(worst_max_aqis))

np.round(list(worst_max_aqis), 2)
```

```
Worst Median AQI :
State
California       48.018868
Arizona          47.307692
Utah             41.066667
Connecticut      39.125000
Delaware         38.000000
Mississippi      37.200000
New Jersey       36.937500
Massachusetts    36.538462
Nevada           36.222222
Pennsylvania     35.756098
Name: Median AQI, dtype: float64

Worst Max AQI :
State
Oregon          430.347826
Washington      334.419355
California      286.981132
Arizona         238.230769
Idaho           197.857143
Wyoming         196.666667
Nevada          196.666667
Montana         137.421053
```

```
Rhode Island     133.000000
Connecticut      124.750000
Name: Max AQI, dtype: float64
```

[15]:
```
array([430.35, 334.42, 286.98, 238.23, 197.86, 196.67, 196.67, 137.42,
       133.  , 124.75])
```

[16]:
```
grader.check("q3a")
```

[16]:
```
q3a results: All test cases passed!
```

1.6.2 Question 3b: Worst AQI States

What are the states that are in both of the top 10 lists? Why do you think most of these states are on both of the lists?

California, Arizona, Connecticut, Nevada are in the top ten for both. three of the four states are in the southwestern part of the United States which is dry due to the topographical differences that prevent moisture from reaching these regions which leads to lower humidity and creates a larger risk of forest fires which negatively affect AQI.

1.6.3 Question 4a: Missing AQI Data

We want to see the accessibility of the AQI data across states. In the following cell, assign `days_with_AQI` to a series that contains the state as the index and the average number of days with AQI entries across all counties in that state as the value. Make sure to label the series as `Days with AQI` and sort in ascending order (smallest average number of days at the top). As before, make sure to remove the regions that are not **US States** from your series.

[17]:
```
anual_aqi = epa_data.get('annual_county_aqi')
days_with_AQI = anual_aqi.groupby('State').mean().drop(['Country Of Mexico',␣
 ↪'District Of Columbia', 'Puerto Rico', 'Virgin Islands']).sort_values('Days␣
 ↪with AQI' , ascending = True)['Days with AQI']


days_with_AQI.head()
```

[17]:
```
State
Alaska         235.222222
Arkansas       251.545455
New Mexico     264.062500
Virginia       265.303030
Colorado       278.892857
Name: Days with AQI, dtype: float64
```

[18]:
```
grader.check("q4a")
```

[18]:
```
q4a results: All test cases passed!
```

1.6.4 Question 4b: What are the missing dates?

In the following cell, we create the series `ca_aqi_days` that outputs a series with each county in California mapped to the number of days that they have AQI data on. Notice that there exists a few counties without the full year of data, which is what you will be taking a closer look at in the following two parts.

[19]:
```
ca_annual_data = epa_data.get('annual_county_aqi')[epa_data.
 ↪get('annual_county_aqi')['State'] == 'California']
ca_aqi_days = ca_annual_data['Days with AQI'].sort_values()
ca_aqi_days.head(10)
```

[19]:
```
54    274
96    331
63    351
98    353
49    359
76    360
51    364
57    364
72    365
79    366
Name: Days with AQI, dtype: int64
```

Question 4bi: Missing Days Assign `county_to_missing_dates` to a dictionary that maps each county with less than the full year of data to the dates that have missing AQI data. Make sure that your keys are just the county name (no whitespace around it or `, California` appended to it) and the values are of the format `yyyy-mm-dd`.

[20]:
```
county_data = epa_data_CA.get('daily_county_aqi')[['county Name', 'AQI',␣
 ↪'Date']]

list_of_dates = county_data.query("`county Name` == 'Alameda'")['Date']
counties_with_missing_dates = ca_annual_data.query("`Days with AQI` < 366").
 ↪County

county_to_missing_dates = {}
for county in counties_with_missing_dates:
    current_county_data = county_data[county_data['county Name'] == county]
    current_county_dates = current_county_data['Date']
    missing_dates = list_of_dates.loc[~list_of_dates.isin(current_county_dates)]
    county_to_missing_dates[county] = list(missing_dates)
```

[21]:
```
grader.check("q4i")
```

[21]:
```
q4i results: All test cases passed!
```

Question 4bii: Missing Days Are there any key missing dates in common between the counties that have missing AQI data? What two counties have the most missing days and why do you think they do?

Some key missing dates in common between the counties include February 29th which is a leap year day and January 6th. Some counties may have chosen not to take the AQI on that day since it is only a day that is recorded every four years. January 6th was the day of the capital riots in 2020. The shock and startling news may have something to do with the missing AQI data. The 2 counties with the most missing days are 'Del Norte' and 'Trinity'. This is likely the case because Trinity County has no incorporated cities and Del Norte County only has one. Because of this, there likely exists less administration taking AQI measurements. Counties with more cities or larger populations likely have more groups of people taking these daily AQI measurements. As a result, any gaps for missing days can easily be filled in since it is likely that people from other cities took measurements on a particular day.

1.6.5 Question 5a: AQI over Time

Assign `aqi_per_month` to a series of the average aqi per month across all US states and `aqi_per_month_CA` to a series of the average AQI per month across California.

[22]:
```
aqi_per_month = daily_county.groupby('Month').mean()['AQI']
aqi_per_month_CA = daily_county[daily_county['State Code'] == 6].
 ↪groupby('Month').mean()['AQI']

print("AQI per Month: \n{}\n".format(aqi_per_month))
print("AQI per Month California : \n{}".format(aqi_per_month_CA))
```

```
AQI per Month:
Month
1      31.032050
2      32.258621
3      34.509181
4      37.287264
5      36.273464
6      40.533681
7      40.070404
8      41.252281
9      43.290611
10     35.285558
11     34.184020
12     34.990632
Name: AQI, dtype: float64

AQI per Month California :
Month
1       46.346888
2       47.110236
3       40.114094
4       41.443462
```

```
5       49.538319
6       47.996146
7       56.069375
8       79.960220
9      107.020228
10      75.491763
11      52.070573
12      53.645516
Name: AQI, dtype: float64
```

[23]:
```
grader.check("q5a")
```

[23]: q5a results: All test cases passed!

1.6.6 Question 5b: AQI over Time Analysis

Is there anything interesting that you notice in `aqi_per_month_CA`? If so, why do you think that is?

August, September, and October are the three months with the worst AQI and we believe this is so because the autumn season is beginning which causes hotter, drier, and windier conditions in California which is the perfect recipe for wildfires that have an increasingly negative effect on AQI.

1.6.7 Question 5c: Modeling AQI over Time

Based on the AQI pattern in the year 2020, if we were to model AQI over the last 10 years, with the average AQI per year being the same, what sort of parametric function $f(x)$ would you use? Let us say that we see a linear increase in the average AQI per year over the last 10 years instead, then what parametric function $g(x)$ would you use?

Based on the AQI pattern in the year 2020, if we were to model AQI over the last 10 years, with the average AQI per year being the same, what sort of parametric function ( ) would you use? Let us say that we see a linear increase in the average AQI per year over the last 10 years instead, then what parametric function ( ) would you use?

1.6.8 Question 6a: Create Heatmap Buckets

Now we want to create a function called `bucket_data`, which takes in the following parameters: `table`, `resolution`. It outputs a pivot table with the latitude bucket (smallest latitude for that grid point) on the index and the longitude bucket (smallest longitude for that grid point) on the columns. The values in the pivot table should be the average AQI of the monitor sites inside that respective rectangle grid of latitudes and longitudes. The following should be the output of `bucket_data(epa_data_CA_merged, np.mean, 5)`:

The `resolution` parameter describes the number of buckets that the latitudes and longitudes are divided into on the heatmap. As an example, let us say that the range of longitudes for site monitors are between [100, 110]; make sure that the start of the range is exactly the **minimum** of all longitude values of your site monitors and the end of the range is the exactly the **maximum** of all longitude values of your site monitors. Let us say that we have a resolution of 10. Then we have the buckets

$$([100, 101], [101, 102], ..., [109, 110])$$

The column and row labels of this dataframe should be labelled as the **start** of the bucket. In the case of the example above, the names of the buckets should be $ 100, 101, …109 $. Note that we are just looking at the longitude dimension in this example, and you have to do do the same for the latitude dimension along the rows in order to build the pivot table.

Finally, make sure the row and column labels of your pivot table are **exactly** the same as the example given above.

[24]:
```
def bucket_data(table, aggfunc, resolution):
    long_buckets = np.sort(np.linspace(table['Longitude'].min(),␣
 ↪table['Longitude'].max(), num=resolution, endpoint = False))
    lat_buckets = np.sort(np.linspace(table['Latitude'].min(),␣
 ↪table['Latitude'].max(), num=resolution, endpoint = False))

    long_buckets_map = dict(list(zip(long_buckets, np.around(long_buckets,␣
 ↪decimals=2))))
    lat_buckets_map = dict(list(zip(lat_buckets, np.around(lat_buckets,␣
 ↪decimals=2))))

    get_lat_bucket_num = lambda loc : lat_buckets_map.
 ↪get(lat_buckets[lat_buckets <= loc].max())
    get_long_bucket_num = lambda loc : long_buckets_map.
 ↪get(long_buckets[long_buckets <= loc].max())

    table['lat_bucket'] = table['Latitude'].apply(get_lat_bucket_num)
    table['long_bucket'] = table['Longitude'].apply(get_long_bucket_num)

    pivot_cols = ['lat_bucket', 'long_bucket', 'AQI']
    return pd.pivot_table(table[pivot_cols], index = 'lat_bucket', columns =␣
 ↪'long_bucket', aggfunc = aggfunc)
```

[25]:
```
grader.check("q6a")
```

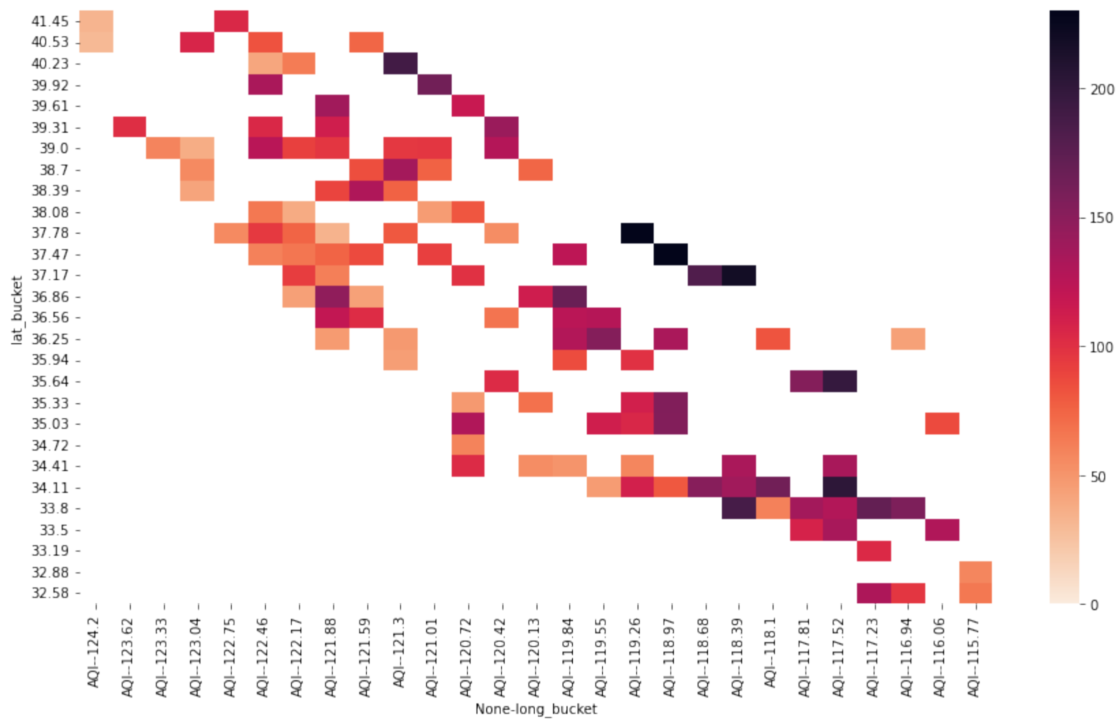
[25]:
```
q6a results: All test cases passed!
```

1.6.9 Question 6b: Visualize Heatmap

Assign `heatmap_data` to a heatmap bucket pivot table aggregated by median with resolution 30 for California AQI for the month of september. The code in the following cell will plot this heatmap for you.

[26]:
```
epa_data_CA_merged_sep = epa_data_CA_merged.query("Month == 9")
heatmap_data = bucket_data(epa_data_CA_merged_sep, np.median, 30)
```

```python
#create visualization
plt.figure(figsize=(15, 8))
ax = sns.heatmap(heatmap_data, vmin=0, vmax=230, cmap = sns.cm.rocket_r)
ax.invert_yaxis()
plt.show()

heatmap_data
```




```
[26]:                     AQI                                                          \
long_bucket -124.20 -123.62 -123.33 -123.04 -122.75 -122.46 -122.17 -121.88
lat_bucket
32.58           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
32.88           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
33.19           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
33.50           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
33.80           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
34.11           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
34.41           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
34.72           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
35.03           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
35.33           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
35.64           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
35.94           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
```

36.25	NaN	NaN	NaN	NaN	NaN	NaN	NaN	47.0
36.56	NaN	NaN	NaN	NaN	NaN	NaN	NaN	120.0
36.86	NaN	NaN	NaN	NaN	NaN	NaN	44.5	147.0
37.17	NaN	NaN	NaN	NaN	NaN	NaN	93.0	61.5
37.47	NaN	NaN	NaN	NaN	NaN	61.0	66.0	75.0
37.78	NaN	NaN	NaN	NaN	56.0	95.0	75.0	33.5
38.08	NaN	NaN	NaN	NaN	NaN	65.5	38.5	NaN
38.39	NaN	NaN	NaN	42.0	NaN	NaN	NaN	89.5
38.70	NaN	NaN	NaN	56.0	NaN	NaN	NaN	NaN
39.00	NaN	NaN	59.0	37.0	NaN	125.0	91.0	96.5
39.31	NaN	101.0	NaN	NaN	NaN	104.5	NaN	112.0
39.61	NaN	NaN	NaN	NaN	NaN	NaN	NaN	138.0
39.92	NaN	NaN	NaN	NaN	NaN	133.5	NaN	NaN
40.23	NaN	NaN	NaN	NaN	NaN	41.0	63.0	NaN
40.53	30.0	NaN	NaN	106.5	NaN	83.5	NaN	NaN
41.45	33.0	NaN	NaN	NaN	105.0	NaN	NaN	NaN

			…						\
long_bucket	-121.59	-121.30	…	-118.97	-118.68	-118.39	-118.10	-117.81	
lat_bucket			…						
32.58	NaN	NaN	…	NaN	NaN	NaN	NaN	NaN	
32.88	NaN	NaN	…	NaN	NaN	NaN	NaN	NaN	
33.19	NaN	NaN	…	NaN	NaN	NaN	NaN	NaN	
33.50	NaN	NaN	…	NaN	NaN	NaN	NaN	108.0	
33.80	NaN	NaN	…	NaN	NaN	187.5	61.0	137.0	
34.11	NaN	NaN	…	80.0	151.0	138.0	163.0	NaN	
34.41	NaN	NaN	…	NaN	NaN	133.0	NaN	NaN	
34.72	NaN	NaN	…	NaN	NaN	NaN	NaN	NaN	
35.03	NaN	NaN	…	154.0	NaN	NaN	NaN	NaN	
35.33	NaN	NaN	…	155.0	NaN	NaN	NaN	NaN	
35.64	NaN	NaN	…	NaN	NaN	NaN	NaN	153.5	
35.94	NaN	45.0	…	NaN	NaN	NaN	NaN	NaN	
36.25	NaN	48.0	…	133.0	NaN	NaN	82.0	NaN	
36.56	102.0	NaN	…	NaN	NaN	NaN	NaN	NaN	
36.86	44.0	NaN	…	NaN	NaN	NaN	NaN	NaN	
37.17	NaN	NaN	…	NaN	182.0	218.5	NaN	NaN	
37.47	87.0	NaN	…	249.0	NaN	NaN	NaN	NaN	
37.78	NaN	80.0	…	NaN	NaN	NaN	NaN	NaN	
38.08	NaN	NaN	…	NaN	NaN	NaN	NaN	NaN	
38.39	131.0	76.0	…	NaN	NaN	NaN	NaN	NaN	
38.70	85.0	136.0	…	NaN	NaN	NaN	NaN	NaN	
39.00	NaN	95.5	…	NaN	NaN	NaN	NaN	NaN	
39.31	NaN	NaN	…	NaN	NaN	NaN	NaN	NaN	
39.61	NaN	NaN	…	NaN	NaN	NaN	NaN	NaN	
39.92	NaN	NaN	…	NaN	NaN	NaN	NaN	NaN	
40.23	NaN	190.0	…	NaN	NaN	NaN	NaN	NaN	
40.53	74.0	NaN	…	NaN	NaN	NaN	NaN	NaN	

```
41.45          NaN    NaN  ...    NaN    NaN    NaN    NaN    NaN


long_bucket -117.52 -117.23 -116.94 -116.06 -115.77
lat_bucket
32.58           NaN   132.0    97.0     NaN    65.0
32.88           NaN     NaN     NaN     NaN    58.0
33.19           NaN   104.0     NaN     NaN     NaN
33.50         134.5     NaN     NaN   129.5     NaN
33.80         129.0   171.5   157.0     NaN     NaN
34.11         203.0     NaN     NaN     NaN     NaN
34.41         134.0     NaN     NaN     NaN     NaN
34.72           NaN     NaN     NaN     NaN     NaN
35.03           NaN     NaN     NaN    87.0     NaN
35.33           NaN     NaN     NaN     NaN     NaN
35.64         197.0     NaN     NaN     NaN     NaN
35.94           NaN     NaN     NaN     NaN     NaN
36.25           NaN     NaN    44.0     NaN     NaN
36.56           NaN     NaN     NaN     NaN     NaN
36.86           NaN     NaN     NaN     NaN     NaN
37.17           NaN     NaN     NaN     NaN     NaN
37.47           NaN     NaN     NaN     NaN     NaN
37.78           NaN     NaN     NaN     NaN     NaN
38.08           NaN     NaN     NaN     NaN     NaN
38.39           NaN     NaN     NaN     NaN     NaN
38.70           NaN     NaN     NaN     NaN     NaN
39.00           NaN     NaN     NaN     NaN     NaN
39.31           NaN     NaN     NaN     NaN     NaN
39.61           NaN     NaN     NaN     NaN     NaN
39.92           NaN     NaN     NaN     NaN     NaN
40.23           NaN     NaN     NaN     NaN     NaN
40.53           NaN     NaN     NaN     NaN     NaN
41.45           NaN     NaN     NaN     NaN     NaN

[28 rows x 27 columns]
```

[27]:
```python
grader.check("q6b")
```

[27]:
```
q6b results: All test cases passed!
```

1.6.10 Question 6c: Analyze Heatmap

Look up where the dark regions correspond to. Does this heatmap make sense?

This heatmap makes sense. Many of the dark regions correspond to areas with a lot of agricultural industry. This industry can contribute to an increase in AQI because it can create PM2.5 particles. These are directly produced when farmers till fields or burn crops before harvest. Additionally, they can come in the form of dust kicked up by livestock.

1.7 Part 3: Open-Ended EDA

Not that we have explored the data both spatially and temporally, we want to be able to look at what other indicators there are for air quality in California. Through the previous few questions we have discussed that wilfire data as well as temperature may be good indicators, but we can explitly look at correlations via the temperature to verify our hypothesis. Like temperature, there are other columns of data such as particulate matter, chemical concentrations, wind data, etc. Your open-ended EDA will be useful for filling in missing points in the heatmap that you created in question 4b.

Your goal in this question is to find relationships between AQI and other features in the current datasets, across time and space. Your exploration can include, but is not limited to: - Looking at correlations between AQI and various columns of interest in `epa_data_CA`. - This will require some merging, which you can look at question 1 for reference. - Performing clustering and/or other unsupervised learning methods such as PCA to discover clusters or useful (combinations of) features in the data. - Merging and exploring other external datasets that you may think are useful.

1.7.1 Question 7a - Code and Analysis

Please complete all of your analysis in the **single cell** below based on the prompt above.

[28]:
```python
annual_county_aqi = epa_data.get('annual_county_aqi')
daily_wind = epa_data.get('daily_temp')
daily_co = epa_data.get('daily_co')
daily_ozone = epa_data.get('daily_ozone')

daily_ozone = daily_ozone[daily_ozone['State Code'] == 6]
daily_co = daily_co[daily_co['State Code'] == 6]
daily_wind = daily_wind[daily_wind['State Code'] == 6][daily_wind["Units of␣
↪Measure"] == "Knots"]

daily_wind = daily_wind.merge(annual_county_aqi, how = 'left', left_on =␣
↪'County Name', right_on = 'County').rename(columns = {'Arithmetic Mean' :␣
↪'Wind Speed Mean'})
daily_co = daily_co.merge(annual_county_aqi, how = 'left', left_on = 'County␣
↪Name', right_on = 'County').rename(columns = {'Arithmetic Mean' : 'Mean CO'})
daily_ozone = daily_ozone.merge(annual_county_aqi, how = 'left', left_on =␣
↪'County Name', right_on = 'County').rename(columns = {'Arithmetic Mean' :␣
↪'Mean Ozone'})
daily_ozone = daily_ozone.groupby('Site Num').mean()

daily_ozone = daily_ozone[daily_ozone['Max AQI'] <400]

heat_map_ozone = daily_ozone[['Max AQI', 'Mean Ozone']]
heat_map_ozone['Max AQI'] = heat_map_ozone['Max AQI'].apply(lambda x : np.
↪round(x))
```

```
heat_map_ozone = heat_map_ozone.set_index("Max AQI").sort_values('Max AQI')
heat_map_ozone
```

[28]:

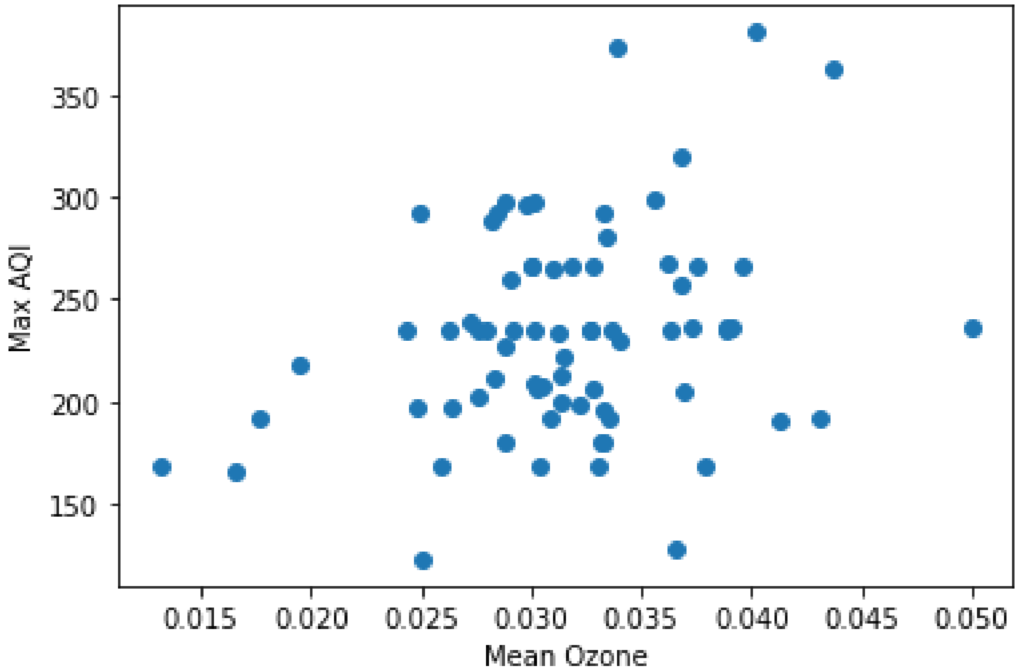
```
         Mean Ozone
Max AQI
122.0      0.024998
128.0      0.036562
166.0      0.016580
168.0      0.037908
168.0      0.013105
…               …
299.0      0.035607
320.0      0.036796
363.0      0.043754
374.0      0.033863
381.0      0.040210

[73 rows x 1 columns]
```

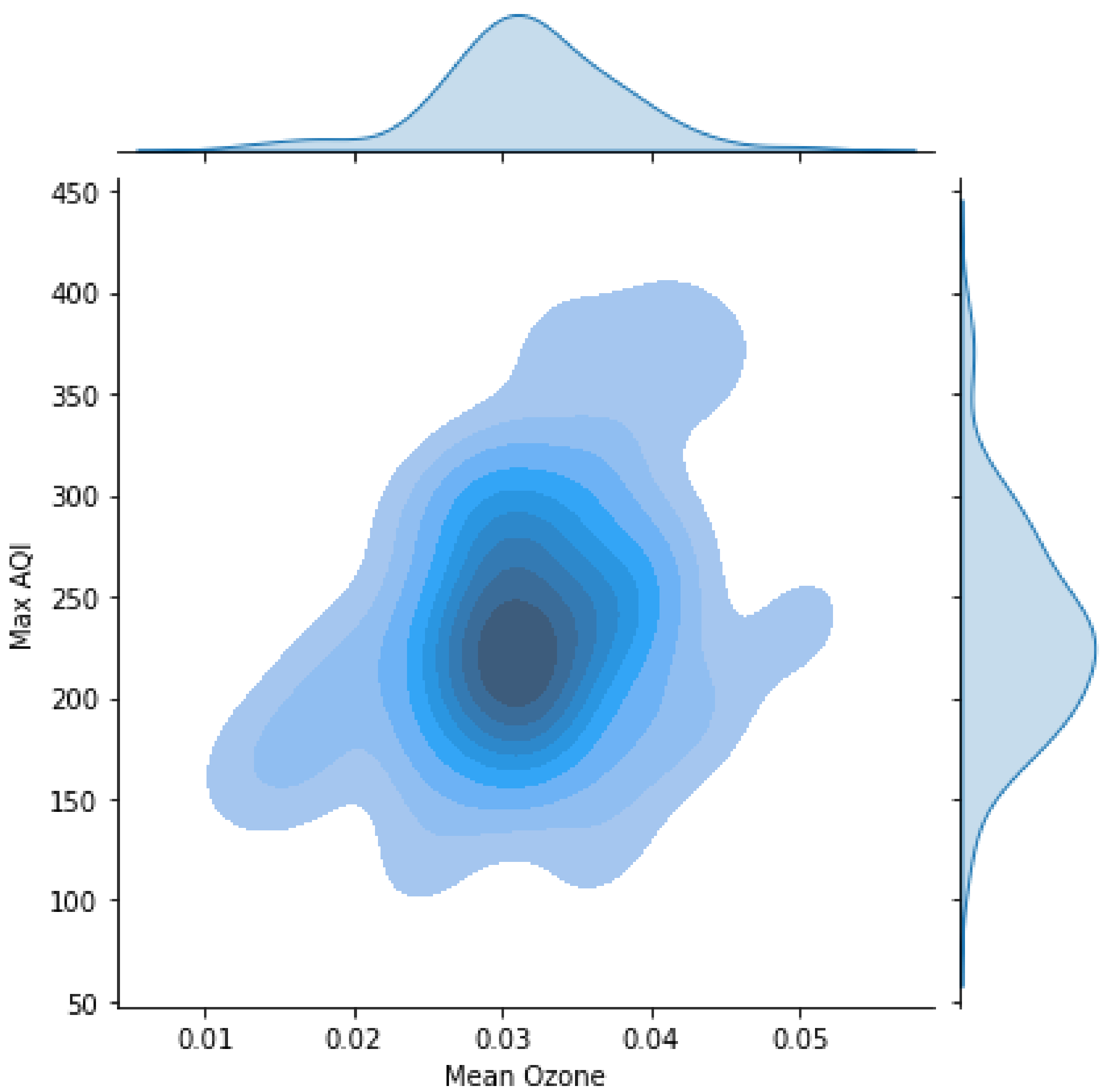
1.7.2 Question 7b - Visualization

Please create **two** visualizations to summarize your analysis above. The only restrictions are that these visualizations **cannot** simply be scatterplots between two features in the dataset(s) and **cannot** be of the same type (dont make two bar graphs, for example).

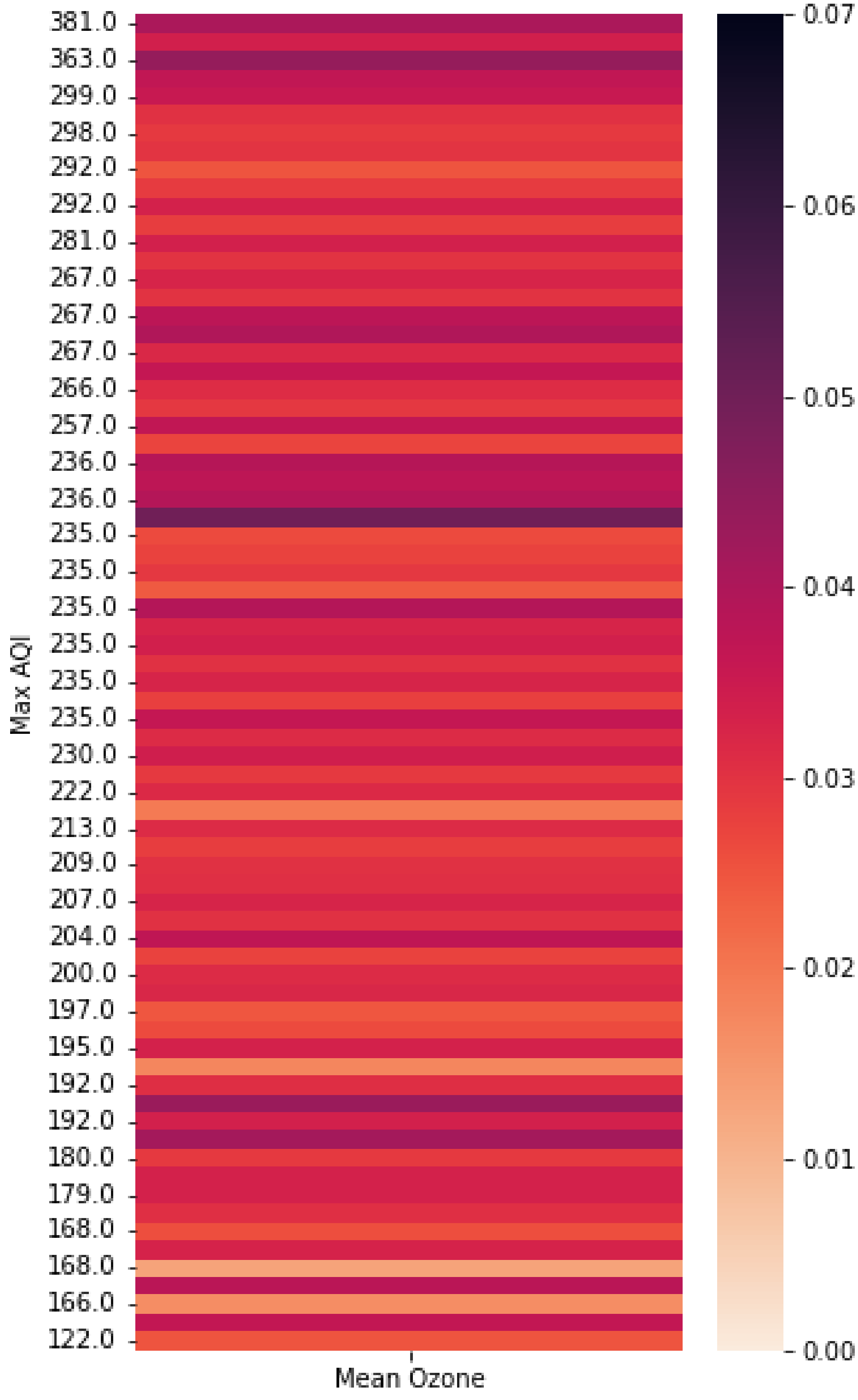
[29]:

```
plt.scatter(daily_ozone['Mean Ozone'], daily_ozone['Max AQI'])
plt.xlabel("Mean Ozone")
plt.ylabel("Max AQI")

sns.jointplot(data=daily_ozone, x="Mean Ozone", y="Max AQI",␣
 ↪kind="kde",fill=True)




plt.figure(figsize=(5, 10))
ax = sns.heatmap(heat_map_ozone, vmin=0, vmax=.07, cmap = sns.cm.rocket_r)
ax.invert_yaxis()
plt.show()
```


350
300
250
200
150
Max AQI
0.015
0.020
0.025
0.030
0.035
0.040
0.045
0.050
Mean Ozone



450
400
350
300
250
200
150
100
50
Max AQI
0.01
0.02
0.03
0.04
0.05
Mean Ozone


381.0
363.0
299.0
298.0
292.0
292.0
281.0
267.0
267.0
267.0
266.0
257.0
236.0
236.0
235.0
235.0
235.0
235.0
235.0
235.0
230.0
222.0
213.0
209.0
207.0
204.0
200.0
197.0
195.0
192.0
192.0
180.0
179.0
168.0
168.0
166.0
122.0
Max AQI
Mean Ozone
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00

1.7.3 Question 7c - Summary

In a paragraph, summarize the your findings and visualizations and explain how they will be useful for predicting AQI. Make sure that your answer is thoughtful and detailed in that it describes what you did and how you reached your conclusion.

Much of our exploratory data analysis involved merging various tables within epa_data_CA. We then proceeded to look for correlations between AQI and various columns of interest. The correlation was measured by drawing various scatter plots until we found particular features that showed moderate to strong linearity with AQI. We tested correlation between AQI and wind speed mean, wind speed max, co mean levels, co max levels, ozone mean, and ozone max. These were tested against median AQI and max AQI. The strongest correlation was shown between ozone mean and max AQI. In other words, as ozone mean levels increased, the max AQIs found across testing sites within counties also increased. We removed outliers by filtering out results with a max AQI greater than 400. We created a contour plot as well which shows higher density contours as the mean ozone increased along with the max AQI. Each axis of the contour plot also has a KDE plot showing the densities of the axes values. They mostly increase in density as the axes values increase up until the Max AQI and the Mean Ozone start getting into uncommon values (values that are very hazardous and untypical). The heat map below shows Max AQI versus Mean Ozone. Darker shades of color signify higher Mean Ozone levels. We found a general trend going from lighter to darker shades as the Max AQI increased. This again shows the correlation we were exploring in the two plots above.

1.8 Part 4: Guided Modeling

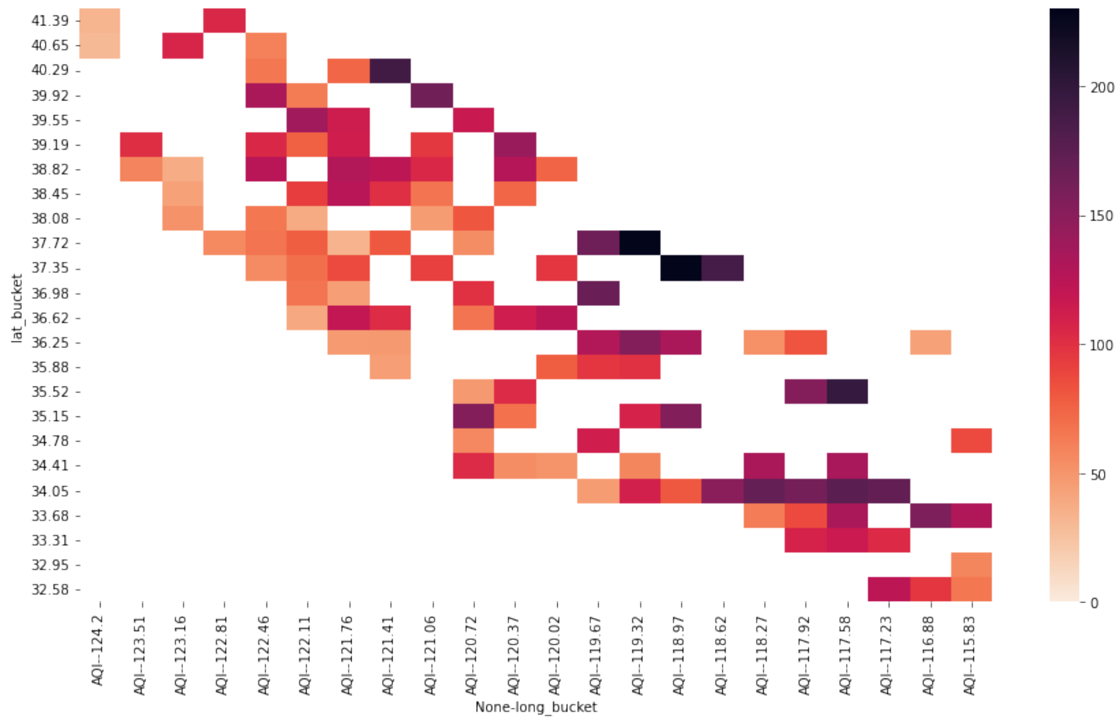
For this part, we will be looking at some open-ended modeling approaches to answering the question of predicting AQI given a location and a date.

1.8.1 Question 8 - Interpolation

For this part, we will be using a simple interpolation to find the missing grid values for AQI on the heatmap visualization that you produced in part 1. Simple linear interpolation just takes the locations' values and averages them to produce an estimate of the current location. Though this is not as predictive (we are not predicting based on features about the location itself), it will give you a sense of the task at hand for the remainder of the project.

As a reminder, the heatmap produced after running the cell below is the one you produced for question 6b when creating a visualization for the AQI in California for the month of september. It produces white spaces where there exist `NaN` values in the pivot table.

[30]:
```
table_sep = epa_data_CA_merged[epa_data_CA_merged['Month'] == 9]
heatmap_data = bucket_data(table_sep, np.median, 25)

plt.figure(figsize=(15, 8))
ax = sns.heatmap(heatmap_data, vmin=0, vmax=230, cmap = sns.cm.rocket_r)
ax.invert_yaxis()
plt.show()
```



[31]:
```
heatmap_data.head()
```

[31]:
```
                   AQI                                                          \
long_bucket -124.20 -123.51 -123.16 -122.81 -122.46 -122.11 -121.76 -121.41
lat_bucket
32.58           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
32.95           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
33.31           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
33.68           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
34.05           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN


                             …                                          \
long_bucket -121.06 -120.72  … -119.67 -119.32 -118.97 -118.62 -118.27
lat_bucket                   …
32.58           NaN     NaN  …     NaN     NaN     NaN     NaN     NaN
32.95           NaN     NaN  …     NaN     NaN     NaN     NaN     NaN
33.31           NaN     NaN  …     NaN     NaN     NaN     NaN     NaN
33.68           NaN     NaN  …     NaN     NaN     NaN     NaN    62.5
34.05           NaN     NaN  …    46.0   110.5    80.0   150.5   170.0


long_bucket -117.92 -117.58 -117.23 -116.88 -115.83
lat_bucket
32.58           NaN     NaN   122.5    97.0    65.0
```

```
32.95           NaN    NaN    NaN    NaN   58.0
33.31         108.0  115.0  104.0    NaN    NaN
33.68          87.0  133.0    NaN  157.0  129.5
34.05         161.0  175.5  171.5    NaN    NaN

[5 rows x 22 columns]
```

[32]:
```
heatmap_data.iloc[0, 0]
```

[32]:
```
nan
```

[33]:
```
pd.isna(heatmap_data.iloc[0, 0])
```

[33]:
```
True
```

[34]:
```
len(heatmap_data) #number of rows
```

[34]:
```
24
```

[35]:
```
len(heatmap_data.columns) #number of columns
```

[35]:
```
22
```

1.8.2 Question 8a - Simple Linear Interpolation

As previously mentioned, interpolation is a technique that is used to predict labels in a dataset by forming a model out of the data that is already labelled. In this case, we have a pivot table that we use to create a heatmap, but there contains many `NaN` values that we want to fill in.

- Create the function `fill_bucket` that takes in the following parameters:
  - `pivot_table`: the pivot table that we are providing.
  - `lat_bucket`: the bucket number that the latitude is in, indexed by zero. ex. if there are 25 buckets, they are numbered \$ 0, 2, …24 \$, from lowest to highest value latitudes.
  - `lon_bucket`: the bucket number that the longitude is in, indexed by zero. ex. if there are 25 buckets, they are numbered \$ 0, 2, …24 \$. from lowest to highest value longitudes.
- In the pivot table, every value has cells above (A cells), cells below (B cells), cells to the left (L cells), and cells to the right (R cells). We will say that a direction (R for example) is valid if and only if there exists a cell **anywhere** to its right that is not `NaN`. The closest such cell will be called the "closest R cell". The same goes for the rest of the directions. For the cases below, assuming that our current cell is called cell K.
  - If cell K is not `NaN`, then simply return the AQI at that given cell.
  - **Only** if there are **at least** three valid directional cells (ex. has A, B, and L valid but not R valid), we will call K *interpolable*. If K is *interpolable*, then interpolate K by assigning it an AQI value equal to the average of the closest cell AQIs in each of the valid directions.
  - If K is *not interpolable*, then do not do anything and simply return `NaN`.
- The return value of `fill_bucket` should be the the value assigned to K. **DO NOT** mutate the cell K in the pivot table yet.

[36]:
```python
def fill_bucket(pivot_table, lat_bucket, lon_bucket):
    if not pd.isna(pivot_table.iloc[lat_bucket, lon_bucket]):
        return pivot_table.iloc[lat_bucket, lon_bucket]

    numOfValidDirections = 0
    sumOfClosestValues = 0

    #closest R value
    column_index = lon_bucket + 1
    while (column_index < len(pivot_table.columns)):
        if not pd.isna(pivot_table.iloc[lat_bucket, column_index]):
            sumOfClosestValues = sumOfClosestValues + pivot_table.
 ↪iloc[lat_bucket, column_index]
            numOfValidDirections = numOfValidDirections + 1
            break
        else:
            column_index = column_index + 1

    #closest L value
    column_index = lon_bucket - 1
    while (column_index >= 0):
        if not pd.isna(pivot_table.iloc[lat_bucket, column_index]):
            sumOfClosestValues = sumOfClosestValues + pivot_table.
 ↪iloc[lat_bucket, column_index]
            numOfValidDirections = numOfValidDirections + 1
            break
        else:
            column_index = column_index - 1

    #closest A value
    row_index = lat_bucket - 1
    while (row_index >= 0):
        if not pd.isna(pivot_table.iloc[row_index, lon_bucket]):
            sumOfClosestValues = sumOfClosestValues + pivot_table.
 ↪iloc[row_index, lon_bucket]
            numOfValidDirections = numOfValidDirections + 1
            break
        else:
            row_index = row_index - 1

    #closest B value
    row_index = lat_bucket + 1
    while (row_index < len(pivot_table)):
        if not pd.isna(pivot_table.iloc[row_index, lon_bucket]):
            sumOfClosestValues = sumOfClosestValues + pivot_table.
 ↪iloc[row_index, lon_bucket]
            numOfValidDirections = numOfValidDirections + 1
```

```
            break
        else:
            row_index = row_index + 1

    #if K is interpolable
    if numOfValidDirections >= 3:
        return sumOfClosestValues/numOfValidDirections
    else:
        return float('NaN')
```

[37]:
```
grader.check("q8a")
```

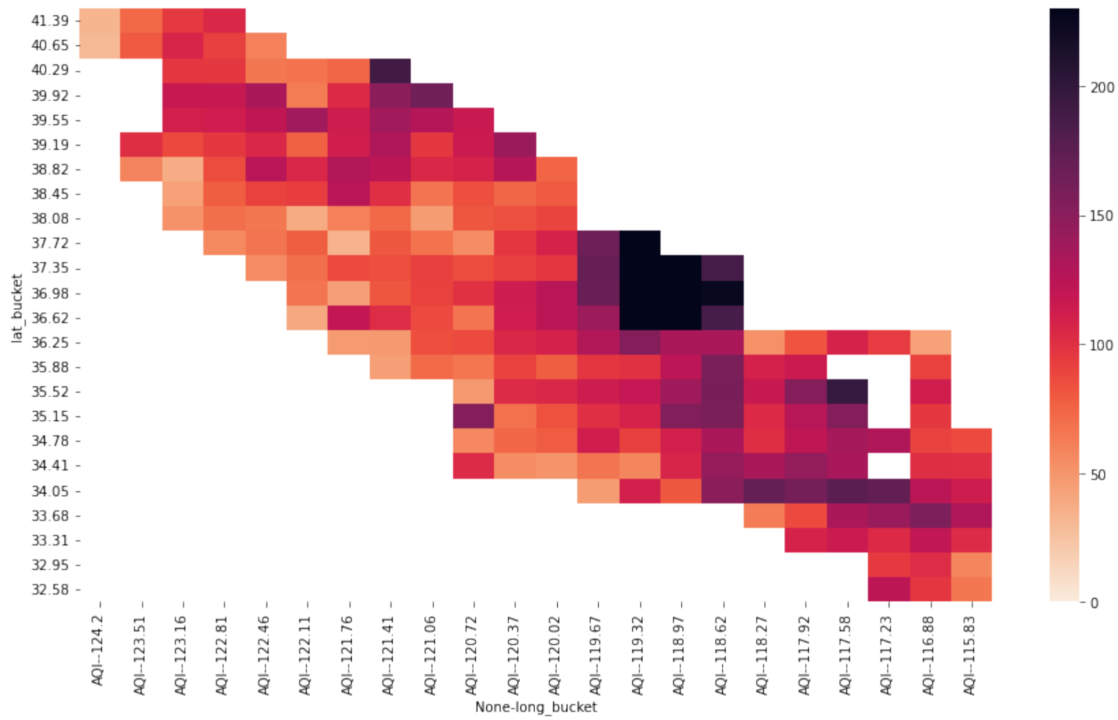
[37]: q8a results: All test cases passed!

1.8.3 Question 8b - Create Filled Heatmap

Now that you have created the `fill_bucket` function, we want to actually use it to fill in the values in `heatmap_data`. Complete the function `fill_all` that takes in the pivot table and fills in all the values and produces a pivot table with the updated values. **DO NOT** mutate the original pivot table. Instead, produce a new pivot table that that contains the filled values.

One point to note is that when we update a cell here, we do not use any surrounding *interpolated* cells to do our interpolation on any given cell. As a result, we will always use the **original** pivot table to find surrounding cells and interpolate.

[38]:
```
def fill_all(pivot_table):
    new_table = pivot_table.copy()
    for lat in range(len(new_table.index)):
        for lon in range(len(new_table.columns)):
            new_table.iat[lat,lon] = fill_bucket(new_table, lat, lon)

    return new_table



filled_heatmap_data = fill_all(heatmap_data)

plt.figure(figsize=(15, 8))
ax = sns.heatmap(filled_heatmap_data, vmin=0, vmax=230, cmap = sns.cm.rocket_r)
ax.invert_yaxis()
plt.show()
```



[39]:
```
grader.check("q8b")
```

[39]: q8b results: All test cases passed!

1.8.4 Question 8c - Other Interpolation Ideas

Instead of just interpolating in a simple fashion as we did above, suggest one other way to interpolate (that actually works so do not just say “put the average of all cells in every `NaN` cell). For example, you can take into account of the distance of the surrounding cells, the number of cells you use, and more.

Include closest diagonal values aswell as Vertical and Horizontal, then make an average of the closest 2 values to set for the K value.

1.8.5 Question 9 - Choosing your Loss Function

Let us say that you are trying to define a loss function $L(x_i, y_i)$ to use for model, where x_i is the input and the y_i is a qualitative variable that that model outputs, consisting of the following five groups: good, moderate, unhealthy for sensitive groups, unhealthy, very unhealthy, or hazardous. How would you design your loss function to evaluate your model?

1.8.6 Question 10: Creating your own Model!

Now that you have an idea of how to interpolate values, we will be using something more predictive. In this part, your final goal is to be creating a model and function that uses **at least four** features, with at least one of those four features being from an external dataset that you bring in and process yourself. Here are some rules on the model that you should follow:

- Using your open-ended EDA analysis, use at least three features in the dataset provided to come up with some sort of predictive model for the AQI for remaining locations not predicted in the heatmap. You are **NOT** allowed to use any more than **one** of the particulate matter features for this model i.e. ozone or CO2 concentrations for example.
  - The reason behind this is that AQI is directly based on these values, so there will be in some sense a near 100% correlation between AQI and these features under some transformations.
- Use at least one feature that comes from an external dataset of choice. Some examples are geographical region (categorical), elevation (quantitative), or wilfire data.
  - Reference question 2c of this project to see how to merge external data with the current EPA data.
- Your model should, at the end, predict one of the following broad categories for the AQI: good, moderate, unhealthy for sensitive groups, unhealthy, very unhealthy, or hazardous. Note that this specification is different from `fill_bucket` in the sense that instead of returning a value, you will be returning a string for a category.
  - As a result, you can either directly predict the category, or the AQI (ex. through regression) and then convert to the category. Category ranges for AQI can be found online.
- The final model should be validated with some data that you hold out. You decide how to do this but there should be some model validation accuracy reported. You should be using the loss function that you designed in question 3 in order to do this.

Deliverables **`features`**: This should be a `pd.DataFrame` object that represents the design matrix that will be fed in as input to your model. Each row represents a data point and each column represents a feature. Essentially your X matrix.

`targets`: This should be a numpy array that where each value corresponds to the AQI value or AQI category for each of the data points in `features`. Essentially your y vector.

`build_model`: This function should have two parameters: `features` that will be used as input into your model as a `pd.DataFrame` object, and `targets` should be a numpy array of AQI values OR AQI categories. It should return a *function* or *object* that represents your model.

`predict`: This function should have two parameters: `model`, the model that you build from the previous function `build_model`, and `features` that represent the design matrix for the test values that we want to predict. It should return the **AQI category** (not a value) that the model predicts for these inputs.

1.8.7 Question 10a: Choose Features and Model

First, decide on the features that you will be using for your model. How predictive do you think each of the features that you chose will be of the AQI category? Then, how will you choose to make your model (multiple regression, decision trees, etc.)?

The features that our model will be using are the mean ozone, mean temp, mean wind speed, and mean AADT at particular coordinates. We think that temperature and wind speed will be relatively good indicators of AQI because wind speeds can determine how much particles are picked up and dispersed through the air. These particles can negatively affect AQI. Higher wind speeds are also associated with less humidity which can increase AQI. Higher temperatures also seem to have a correlation with AQI likely due to the fact that temperature affects the movement of air and thus the movement of air pollution. Mean ozone is a particulate matter which directly affects AQI. Traffic is a large source of pollution in many areas which will likely make AADT a good predictive feature for our model. We will choose to make our model using multiple regression.

1.8.8 Question 10b: Build Features

Create the `build_features` function as described at the beginning of this part. You should also do any cleaning or merging of internal or external datasets in this part. Make sure to read the specifications of the function very carefully. The autograder will provide some sanity checks on your output.

[40]:
```
traffic_data = traffic_data[['Ahead_AADT', 'Lon_S_or_W', 'Lat_S_or_W']].
 ↪rename(columns = {'Ahead_AADT' : 'AADT', 'Lon_S_or_W' : 'Longitude',␣
 ↪'Lat_S_or_W' : 'Latitude'})
traffic_data = traffic_data.apply(pd.to_numeric, errors = 'coerce').dropna(axis␣
 ↪= 0, how = 'any')

epa_data_CA_merged = epa_data_CA_merged[['Latitude', 'Longitude', 'AQI']]
daily_ozone = epa_data_CA.get('daily_ozone')[['Latitude', 'Longitude',␣
 ↪'Arithmetic Mean']]
daily_temp = epa_data_CA.get('daily_wind')[['Latitude', 'Longitude',␣
 ↪'Arithmetic Mean', 'Units of Measure']]
daily_wind = epa_data_CA.get('daily_temp')[['Latitude', 'Longitude',␣
 ↪'Arithmetic Mean', 'Units of Measure']].query("`Units of Measure` ==␣
 ↪'Knots'")

epa_data_CA_merged = epa_data_CA_merged.groupby(['Latitude', 'Longitude']).
 ↪mean()
daily_ozone = daily_ozone.groupby(['Latitude', 'Longitude']).mean()
daily_wind = daily_wind.groupby(['Latitude', 'Longitude']).mean()
daily_temp = daily_temp.groupby(['Latitude', 'Longitude']).mean()
traffic_data = traffic_data.groupby(['Latitude', 'Longitude']).mean()
traffic_data = traffic_data.reset_index()

epa_data_CA_merged = epa_data_CA_merged.merge(daily_ozone, how = 'left',␣
 ↪left_on = ['Latitude', 'Longitude'], right_on = ['Latitude', 'Longitude']).
 ↪rename(columns = {'Arithmetic Mean' : 'Mean Ozone'})
epa_data_CA_merged = epa_data_CA_merged.reset_index()

epa_data_CA_merged = epa_data_CA_merged.merge(daily_temp, how = 'left', left_on␣
 ↪= ['Latitude', 'Longitude'], right_on = ['Latitude', 'Longitude']).
 ↪rename(columns = {'Arithmetic Mean' : 'Mean Temp'})
```

```
epa_data_CA_merged = epa_data_CA_merged.merge(daily_wind, how = 'left', left_on␣
↪= ['Latitude', 'Longitude'], right_on = ['Latitude', 'Longitude']).
↪rename(columns = {'Arithmetic Mean' : 'Mean Wind Speed'})

epa_data_CA_merged = epa_data_CA_merged.merge(traffic_data, how = 'cross').
↪rename(columns = {'AADT' : 'Mean AADT'})

epa_data_CA_merged['keep'] = (abs(epa_data_CA_merged['Latitude_y'] -␣
↪epa_data_CA_merged['Latitude_x']) <= 1)
epa_data_CA_merged['keep_2'] = (abs(epa_data_CA_merged['Longitude_y'] -␣
↪epa_data_CA_merged['Longitude_x']) <= 1)

epa_data_CA_merged = epa_data_CA_merged.query("keep == True and keep_2 ==␣
↪True").groupby(['Latitude_x', 'Longitude_x']).mean().reset_index().
↪drop(columns = ['keep', 'keep_2', 'Latitude_y', 'Longitude_y'])
epa_data_CA_merged = epa_data_CA_merged.rename(columns = {'Latitude_x' :␣
↪'Latitude', 'Longitude_x' : 'Longitude'}).dropna(axis = 0, how = 'any')

features = epa_data_CA_merged[['Mean Ozone', 'Mean Temp', 'Mean Wind Speed',␣
↪'Mean AADT']]
targets = epa_data_CA_merged[['AQI']]


features.head()
```

[40]:

	Mean Ozone	Mean Temp	Mean Wind Speed	Mean AADT
0	0.033288	64.443212	3.796735	107590.986193
1	0.032622	65.768566	3.448596	13043.612565
2	0.030819	64.204273	3.114406	114765.768566
3	0.033480	64.913998	2.287289	114458.922610
4	0.031616	95.610960	3.506522	15219.245283

[41]:
```
grader.check("q10b")
```

[41]: q10b results: All test cases passed!

1.8.9 Question 10c: Build Your Model!

Create the `build_model` function as described at the beginning of this part. Make sure to read the specifications of the function very carefully. The autograder will provide some sanity checks on your output.

[42]:
```
from sklearn.linear_model import LinearRegression

def build_model(features, targets):
    model = LinearRegression()
    model.fit(features, targets)
```

```
    return model
```

[43]:
```
grader.check("q10c")
```

[43]: q10c results: All test cases passed!

1.8.10 Question 10d: Predict Points

Create the `predict` function as described at the beginning of this part. Make sure to read the specifications of the function very carefully. The autograder will provide some sanity checks on your output.

[44]:
```
def category_mapper(AQI):
    if AQI < 51:
        return "good"
    elif AQI < 101:
        return "moderate"
    elif AQI < 151:
        return "unhealthy sensitive groups"
    elif AQI < 201:
        return "unhealthy"
    elif AQI < 301:
        return "very unhealthy"
    else:
        return "hazardous"
```

[45]:
```
categories = ["good", "moderate", "unhealthy sensitive groups", "unhealthy",␣
 ↪"very unhealthy", "hazardous"]

def predict(model, features):
    y_hat = model.predict(features)
    y_hat = pd.Series(y_hat.flatten()).apply(category_mapper)
    return y_hat
```

[46]:
```
grader.check("q10d")
```

[46]: q10d results: All test cases passed!

1.8.11 Question 10e: Model Validation and Performance

Now that you have finished making your model, we want to see how well it performs on our data. In this question, use the following cell to split your data into training and validation sets. You should partition 70% of your data to be used as your training set, and the remaining to be used as your validation set.

Assign `binary_error` to be the **fraction of inputs on your validation set that the your `predict` function classifies incorrectly.** Note that this is a binary loss in some sense as it assigns a loss of 1 to those points predicted incorrectly, and a loss of 0 to those points predicted correctly.

Assign `cv_error` to be the the error on the validation set produced by the loss function $ L $ that you designed in question 3.

Hint: you can use `train_test_split` from `sklearn`.

[70]:
```
def quantifier(category):
    if category == "good":
        return 1
    elif category == "moderate":
        return 2
    elif category == "unhealthy sensitive groups":
        return 3
    elif category == "unhealthy":
        return 4
    elif category == "very unhealthy":
        return 5
    elif category == "hazardous":
        return 6
```

[83]:
```
from sklearn.model_selection import train_test_split

training_data, validation_data = train_test_split(epa_data_CA_merged,␣
↪test_size=0.3, random_state=83)

validation_features = validation_data[['Mean Ozone', 'Mean Temp', 'Mean Wind␣
↪Speed', 'Mean AADT']]
true_values = validation_data['AQI'].apply(category_mapper).reset_index()['AQI']

training_features = training_data[['Mean Ozone', 'Mean Temp', 'Mean Wind␣
↪Speed', 'Mean AADT']]
targets = training_data[['AQI']]

model = build_model(training_features, targets)
predictions = predict(model, validation_features)

binary_error = sum(true_values != predictions)/len(predictions)

predictions_quantified = predictions.apply(quantifier)
true_values_quantified = true_values.apply(quantifier)
cv_error = abs(true_values_quantified - predictions_quantified).mean()
```

[84]:
```
grader.check("q10e")
```

[84]:
```
q10e results: All test cases passed!
```

1.9 Part 5: Open-Ended Modeling

Now that you have had some experience with creating the a model from scratch using the existing data, you are now ready to explore other questions, such as the ones in your design document. In this section, you will use the tools that we developed in the previous parts to answer the hypothesis of your choice! Note that breaking your model-building and analysis process into modularized functions as you did above will make your code more interpretable and less error-prone.

1.9.1 Question 11a

Train a baseline model of your choice using any supervised learning approach we have studied to answer your hypothesis and predict something related to AQI; you are not limited to a linear model. However, you may use a maximum of **three features** for this part. After training, evaluate it on some validation data that you hold out yourself.

[227]:
```
...
```

1.9.2 Question 11b

Explain and summarize the model that you used. In your summary, make sure to include the model description, the inputs, the outputs, as well as the cross-validation error. Additionally, talk a little bit about what you would change to your baseline model to improve it. The expected length of your summary should be 8-12 sentences.

Type your answer here, replacing this text.

1.9.3 Question 11c

Improve your model from part 11a based on the improvements that you suggested in part 11b. This could be the addition of more features, performing additional transformations on your features, increasing/decreasing the complexity of the model itself, or really anything else. You have no limitation on the number of features you can use, but you are required to use at least **one external dataset** that you process and merge in yourself.

[228]:
```
...
```

1.9.4 Question 11d

Compare and contrast your baseline model and (hopefully) improved model. Make sure to compare their validation errors. Were you able to successfully answer your research question and evaluate your hypothesis? Summarize in a few sentences the conclusions that you can draw from your model and predictions. The expected length of your response should be 8-10 sentences.

Type your answer here, replacing this text.

To double-check your work, the cell below will rerun all of the autograder tests.

[ ]:
```
grader.check_all()
```

1.10 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

[ ]:
```
# Save your notebook first, then run this cell to export your submission.
grader.export()
```