AQI

March 27, 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
```

```
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]:
```
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]:
```
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.

[5]:
```
epa_data.get('annual_county_aqi').head()
```

[5]:

	State	County	Year	Days with AQI	Good Days	Moderate Days	\
0	Alabama	Baldwin	2020	269	250	19	
1	Alabama	Clay	2020	108	99	9	
2	Alabama	DeKalb	2020	364	350	14	
3	Alabama	Elmore	2020	197	197	0	
4	Alabama	Etowah	2020	278	260	18	

	Unhealthy for Sensitive Groups Days	Unhealthy Days	Very Unhealthy Days	\
0	0	0	0	
1	0	0	0	
2	0	0	0	
3	0	0	0	
4	0	0	0	

	Hazardous Days	Max AQI	90th Percentile AQI	Median AQI	Days CO	\
0	0	74	49	36	0	
1	0	86	49	26	0	
2	0	90	45	36	0	
3	0	47	41	31	0	
4	0	92	46	34	0	

	Days NO2	Days Ozone	Days SO2	Days PM2.5	Days PM10
0	0	198	0	71	0
1	0	0	0	108	0
2	0	331	0	33	0
3	0	197	0	0	0
4	0	204	0	74	0

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?

Because the max AQI might not be representitive of what the actual high end annual county aqi is because there could exist 1 county that has a much higher AQI compared to the other counties. If we only use max, then we wouldn't capture that. If 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, that infers there exist an outlier which a county has a very high AQI while the difference between the other counties are not as large.

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.

[6]:
```
daily_county = epa_data.get('daily_county_aqi')
daily_county['Day'] = pd.DatetimeIndex(daily_county['Date']).day
daily_county['Month'] = pd.DatetimeIndex(daily_county['Date']).month

daily_county.head()
```

[6]:

	State Name	county Name	State Code	County Code	Date	AQI	Category	Defining Parameter	Defining Site	Number of Sites Reporting	Day	Month
0	Alabama	Baldwin	1	3	2020-01-01	48	Good	PM2.5	01-003-0010	1	1	1
1	Alabama	Baldwin	1	3	2020-01-04	13	Good	PM2.5	01-003-0010	1	4	1
2	Alabama	Baldwin	1	3	2020-01-07	14	Good	PM2.5	01-003-0010	1	7	1
3	Alabama	Baldwin	1	3	2020-01-10	39	Good	PM2.5	01-003-0010	1	10	1
4	Alabama	Baldwin	1	3	2020-01-13	29	Good	PM2.5	01-003-0010	1	13	1

[7]:
```
grader.check("q1a")
```

[7]:
```
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`.

[8]:
```
epa_data_CA = epa_data.copy()
epa_data_CA.pop("annual_county_aqi")
epa_data_CA.pop("aqs_sites")

for key in epa_data_CA.keys():
    epa_data_CA[key] = epa_data_CA[key][epa_data_CA[key]["State Name"] ==␣
 ↪"California"]


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

[8]:
```
       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

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

[9]:
```
grader.check("q1b")
```

[9]:
```
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`

[10]:
```
epa_data_CA2 = epa_data.copy()
epa_data_CA2.pop("annual_county_aqi")

epa_data_CA2["aqs_sites"] =␣
 ↪epa_data_CA2["aqs_sites"][epa_data_CA2["aqs_sites"]["State Name"] ==␣
 ↪"California"]
epa_data_CA2["aqs_sites"]["Site Num"] = epa_data_CA2["aqs_sites"]["Site Number"]

cols = ['State Name', 'county Name', 'Month', 'Day', 'AQI', 'Category',␣
 ↪'Defining Site', 'Latitude', 'Longitude']
info = pd.DataFrame()
for key in epa_data_CA:
    if key != "daily_county_aqi":
```

```
        info = pd.concat([info, epa_data_CA[key]])
info = pd.concat([info, epa_data_CA2["aqs_sites"]])
info = info[['State Code', 'County Code', 'Site Num', 'Latitude', 'Longitude']]
info['State Code'] = info['State Code'].astype('str').str.zfill(2)
info['County Code'] = info['County Code'].astype('str').str.zfill(3)
info['Site Num'] = info['Site Num'].astype('str').str.zfill(4)
info['Defining Site'] = info['State Code'] + '-' + info['County Code'] + '-' +␣
 ↪info['Site Num']
info = info.drop_duplicates('Defining Site', keep = "first")
epa_data_CA_merged = epa_data_CA["daily_county_aqi"].merge(info, how = "left",␣
 ↪left_on = 'Defining Site', right_on = 'Defining Site')
epa_data_CA_merged = epa_data_CA_merged[cols]

epa_data_CA_merged.head()
```

[10]:

	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

[11]:
```
grader.check("q1c")
```

[11]:
```
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()`.

[12]:
```
traffic_data = pd.read_csv("data/Traffic_Volumes_AADT.csv")
traffic_needed = traffic_data[["District", "Route", "County", "Descriptn",␣
 ↪"Back_AADT", "Ahead_AADT", "Lon_S_or_W", "Lat_S_or_W"]]
traffic_needed["Back_AADT"] = traffic_needed["Back_AADT"].str.strip()
traffic_needed["Ahead_AADT"] = traffic_needed["Ahead_AADT"].str.strip()
traffic_needed["Back_AADT"] = traffic_needed["Back_AADT"].replace('','0')
traffic_needed["Ahead_AADT"] = traffic_needed["Ahead_AADT"].replace('','0')
traffic_needed = traffic_needed.astype({'Back_AADT': 'int64', 'Ahead_AADT':␣
 ↪'int64'})

traffic_needed["AADT"] = traffic_needed["Ahead_AADT"] +␣
 ↪traffic_needed["Back_AADT"]
traffic_needed = traffic_needed.drop(["Back_AADT", "Ahead_AADT"], axis = 1)
traffic_needed = traffic_needed.rename(columns={"Lon_S_or_W": "Longitude",␣
 ↪"Lat_S_or_W": "Latitude"})
traffic_data_cleaned = traffic_needed

traffic_data_cleaned.head()
```

[12]:

	District	Route	County	Descriptn	Longitude	Latitude	AADT
0	1	1	MEN	SONOMA/MENDOCINO COUNTY LINE	-123.5185026	38.75984264	4000
1	1	1	MEN	NORTH LIMITS GUALALA	-123.53189	38.77004592	7100
2	1	1	MEN	FISH ROCK ROAD	-123.585411	38.80354931	6200
3	1	1	MEN	POINT ARENA, SOUTH CITY LIMITS	-123.6915134	38.90397338	4600
4	1	1	MEN	POINT ARENA, RIVERSIDE DRIVE	-123.6924099	38.91091252	5000

[13]:
```
grader.check("q2a")
```

[13]:
```
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`.

[14]:
```
# cleaning the longitude and latitude data by deleting non-number rows and␣
 ↪casting the rest to float
traffic_data_cleaned['Latitude'] =␣
 ↪traffic_data_cleaned['Latitude'][traffic_data_cleaned['Latitude'] != 'Left␣
 ↪Skipped - Input PM on Right Ind. Alignment']
traffic_data_cleaned['Longitude'] =␣
 ↪traffic_data_cleaned['Longitude'][traffic_data_cleaned['Longitude'] != 'Left␣
 ↪Skipped - Input PM on Right Ind. Alignment']
traffic_data_cleaned.astype({'Latitude': 'float', "Longitude": "float"})
```

[14]:
```
      District  Route County                        Descriptn   Longitude  \
0            1      1    MEN     SONOMA/MENDOCINO COUNTY LINE -123.518503
1            1      1    MEN              NORTH LIMITS GUALALA -123.531890
2            1      1    MEN                    FISH ROCK ROAD -123.585411
3            1      1    MEN   POINT ARENA, SOUTH CITY LIMITS -123.691513
4            1      1    MEN     POINT ARENA, RIVERSIDE DRIVE -123.692410
…          …    …    …                                 …          …
7115        12    605    ORA           SEAL BEACH, JCT RTE 22 -118.091474
7116        12    605    ORA                    JCT. RTE. 405 -118.091768
7117        12    605    ORA     LOS ALAMITOS, KATELLA AVENUE -118.082030
7118        12    605    ORA   ORANGE/LOS ANGELES COUNTY LINE -118.081547
7119         3     99    SAC                   BREAK IN ROUTE -121.473649

       Latitude    AADT
0     38.759843    4000
1     38.770046    7100
2     38.803549    6200
3     38.903973    4600
4     38.910913    5000
…           …       …
7115  33.778633   46100
7116  33.784414  212200
7117  33.802799  326800
7118  33.806140  170000
7119  38.558838       0

[7120 rows x 7 columns]
```

[15]:
```
import geopandas as gpd
from shapely.geometry import Point

# turn dataframe to geopandas
g_epa_data_CA_merged = gpd.GeoDataFrame(
    epa_data_CA_merged.copy(), geometry=gpd.points_from_xy(x=epa_data_CA_merged.
 ↪Longitude, y=epa_data_CA_merged.Latitude)
)


g_traffic_data_cleaned = gpd.GeoDataFrame(
    traffic_data_cleaned, geometry=gpd.points_from_xy(x=traffic_data_cleaned.
 ↪Longitude, y=traffic_data_cleaned.Latitude)
)

# join tables
gpd_epa_traffic = gpd.sjoin_nearest(g_epa_data_CA_merged,␣
 ↪g_traffic_data_cleaned)

#rename columns and get the needed columns
gpd_epa_traffic = gpd_epa_traffic.rename(columns={"Latitude_left": "Site Lat",␣
 ↪"Longitude_left": "Site Long", "Longitude_right":"Traffic Long",␣
 ↪"Latitude_right":"Traffic Lat"})
gpd_epa_traffic = gpd_epa_traffic[["State Name", "county Name", "Month", "Day",␣
 ↪"AQI", "Category", "Defining Site", "Site Lat", "Site Long", "Traffic Lat",␣
 ↪"Traffic Long", "Descriptn", "AADT"]]

gpd_epa_traffic = gpd_epa_traffic.astype({'Traffic Long': 'float'})
gpd_epa_traffic.head()
```

[15]:

	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	Traffic Lat	Traffic Long	\
0	06-001-0009	37.743065	-122.169935	37.74435189	-122.170586	
24	06-001-0009	37.743065	-122.169935	37.74435189	-122.170586	
184	06-001-0009	37.743065	-122.169935	37.74435189	-122.170586	
185	06-001-0009	37.743065	-122.169935	37.74435189	-122.170586	
186	06-001-0009	37.743065	-122.169935	37.74435189	-122.170586	

	Descriptn	AADT
0	OAKLAND, 98TH AVENUE	48300
24	OAKLAND, 98TH AVENUE	48300

```
184  OAKLAND, 98TH AVENUE  48300
185  OAKLAND, 98TH AVENUE  48300
186  OAKLAND, 98TH AVENUE  48300
```

[16]:
```
grader.check("q2b")
```

[16]:
```
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**.

[17]:
```
annual_aqi = epa_data.get('annual_county_aqi')
annual_aqi
annual_aqi["State"].unique()
annual_aqi = annual_aqi[(annual_aqi["State"] != 'Country Of Mexico')]
annual_aqi = annual_aqi[(annual_aqi["State"] != 'Virgin Islands')]
annual_aqi = annual_aqi[(annual_aqi["State"] != 'Puerto Rico')]
annual_aqi = annual_aqi[(annual_aqi["State"] != 'District Of Columbia')]
worst_median_aqis = annual_aqi.groupby("State").agg(np.mean).
 ↪sort_values("Median AQI", ascending = False)["Median AQI"][:10]
worst_max_aqis = annual_aqi.groupby("State").agg(np.mean).sort_values("Max␣
 ↪AQI", ascending = False)["Max AQI"][: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
```

[17]:
```
array([430.35, 334.42, 286.98, 238.23, 197.86, 196.67, 196.67, 137.42,
       133.  , 124.75])
```

[18]:
```
grader.check("q3a")
```

[18]:
```
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, Conneticut, Nevada. California has a lot of people and a lot of cars, which is probably bad for air quality. It also has a lot of wildfires that can make it worse. Arizona has sandstormsthat probably make the quality bad, and it could also get air pollution from California. Conneticut is probably too close to coal power plants. Winds probably blow California's air pollution into Nevada.

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.

[19]:
```
daily_county_aqi = epa_data["daily_county_aqi"]
daily_county_aqi = daily_county_aqi[(daily_county_aqi["State Name"] != 'Country␣
 ↪Of Mexico')]
```

```
daily_county_aqi = daily_county_aqi[(daily_county_aqi["State Name"] != 'Virgin␣
 ↪Islands')]
daily_county_aqi = daily_county_aqi[(daily_county_aqi["State Name"] != 'Puerto␣
 ↪Rico')]
daily_county_aqi = daily_county_aqi[(daily_county_aqi["State Name"] !=␣
 ↪'District Of Columbia')]

days_count = daily_county_aqi.groupby("State Name").size().rename("Days with␣
 ↪AQI")
num_county = daily_county_aqi.groupby("State Name").nunique()["county Name"]

days_with_AQI = days_count/num_county
days_with_AQI = days_with_AQI.sort_values(ascending = True)
days_with_AQI.head()
```

[19]:
```
State Name
Alaska        235.222222
Arkansas      251.545455
New Mexico    264.062500
Virginia      265.303030
Colorado      278.892857
dtype: float64
```

[20]:
```
grader.check("q4a")
```

[20]:
```
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.

[21]:
```
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)
```

[21]:
```
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`.

[22]:
```
county_to_missing_dates = {}
# sort dataframe so that we can match county to days with aqi
temp_ca_annual_data = ca_annual_data[["County", 'Days with AQI']].
 ↪sort_values(by=['Days with AQI'])
# replace the index in the array with the names of the county
ca_aqi_days.index = temp_ca_annual_data["County"]
# pull data on days with epa data from epa_data_CA["daily_county_aqi"]
counties_and_dates = epa_data_CA["daily_county_aqi"][["county Name", "Date"]]



county_to_missing_dates = {}
# go through the array of total days of data
for index in ca_aqi_days.index:
    # if we're missing dates
    if ca_aqi_days[index] != 366:
        # add a list to our dictionary to hold missing dates
        county_to_missing_dates[index.strip()] = []
county_to_missing_dates



# import datetime
import datetime
# define range of dates I'll go over
d1 = datetime.date(2020, 1, 1)
d2 = datetime.date(2021, 1, 1)

# got through the dictionary
for index in county_to_missing_dates:
    # get a list of all available dates for a county
    dates = counties_and_dates[counties_and_dates["county Name"] == index]
    # get a list of all days in my range of days
    days = [d1 + datetime.timedelta(days=x) for x in range((d2-d1).days + 1)]
    # change that list of days to string form
    days = ["20" + day.strftime("%y-%m-%d") for day in days]
    # I had too many days, make number of days 366
    days = set(days[:366])
```

```
    # remove duplicate days
    for date in dates["Date"]:
        days.remove(date)
    # update the dictionary
    county_to_missing_dates[index] = list(days)

# sort all the lists
for index in county_to_missing_dates:
    county_to_missing_dates[index] = np.sort(county_to_missing_dates[index])
county_to_missing_dates
```

[22]:
```
{'Del Norte': array(['2020-01-15', '2020-01-16', '2020-01-17', '2020-01-18',
        '2020-01-20', '2020-01-21', '2020-01-23', '2020-03-14',
        '2020-03-15', '2020-04-22', '2020-04-23', '2020-04-25',
        '2020-04-26', '2020-04-28', '2020-04-29', '2020-05-01',
        '2020-05-02', '2020-05-04', '2020-05-05', '2020-05-07',
        '2020-05-08', '2020-05-10', '2020-05-11', '2020-05-13',
        '2020-05-14', '2020-05-16', '2020-05-17', '2020-05-19',
        '2020-05-20', '2020-05-22', '2020-05-23', '2020-05-25',
        '2020-05-26', '2020-05-28', '2020-05-29', '2020-05-31',
        '2020-06-01', '2020-06-02', '2020-06-03', '2020-06-04',
        '2020-06-05', '2020-06-06', '2020-06-07', '2020-06-08',
        '2020-06-09', '2020-06-10', '2020-06-11', '2020-06-12',
        '2020-06-13', '2020-06-14', '2020-06-15', '2020-06-16',
        '2020-06-17', '2020-06-18', '2020-06-19', '2020-06-20',
        '2020-06-21', '2020-06-22', '2020-06-23', '2020-06-24',
        '2020-06-25', '2020-06-26', '2020-06-27', '2020-06-28',
        '2020-06-29', '2020-06-30', '2020-07-01', '2020-07-02',
        '2020-07-03', '2020-07-04', '2020-07-05', '2020-07-06',
        '2020-07-07', '2020-07-08', '2020-07-09', '2020-07-10',
        '2020-07-11', '2020-07-12', '2020-07-13', '2020-07-14',
        '2020-07-15', '2020-07-16', '2020-07-17', '2020-07-18',
        '2020-07-19', '2020-07-20', '2020-07-21', '2020-09-13',
        '2020-09-14', '2020-09-15', '2020-12-02', '2020-12-03'],
       dtype='<U10'),
 'Trinity': array(['2020-01-29', '2020-02-28', '2020-02-29', '2020-03-02',
        '2020-03-03', '2020-07-01', '2020-07-02', '2020-07-03',
        '2020-07-04', '2020-07-05', '2020-07-06', '2020-07-08',
        '2020-07-09', '2020-07-10', '2020-07-11', '2020-07-12',
        '2020-07-13', '2020-07-14', '2020-07-23', '2020-07-24',
        '2020-07-25', '2020-07-26', '2020-07-27', '2020-07-28',
        '2020-07-29', '2020-07-30', '2020-07-31', '2020-08-01',
        '2020-08-02', '2020-08-03', '2020-08-13', '2020-09-27',
        '2020-09-28', '2020-12-01', '2020-12-02'], dtype='<U10'),
 'Lake': array(['2020-02-04', '2020-02-05', '2020-02-29', '2020-04-24',
        '2020-05-04', '2020-05-05', '2020-05-06', '2020-05-07',
        '2020-06-16', '2020-06-17', '2020-08-17', '2020-08-22',
```

```
       '2020-08-23', '2020-08-24', '2020-10-21'], dtype='<U10'),
 'Tuolumne': array(['2020-01-08', '2020-01-26', '2020-01-27', '2020-01-28',
       '2020-01-29', '2020-01-30', '2020-01-31', '2020-02-01',
       '2020-02-02', '2020-02-03', '2020-09-07', '2020-09-08',
       '2020-10-25'], dtype='<U10'),
 'Amador': array(['2020-01-04', '2020-01-05', '2020-01-06', '2020-01-07',
       '2020-08-24', '2020-10-17', '2020-10-18'], dtype='<U10'),
 'Plumas': array(['2020-01-06', '2020-02-16', '2020-02-25', '2020-02-26',
       '2020-02-28', '2020-02-29'], dtype='<U10'),
 'Calaveras': array(['2020-06-04', '2020-09-21'], dtype='<U10'),
 'Glenn': array(['2020-11-08', '2020-11-11'], dtype='<U10'),
 'Napa': array(['2020-11-04'], dtype='<U10')}
```

[23]:
```
grader.check("q4i")
```

[23]:
```
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?

For the most part, there doesn't seem to be any commonality for dates missed across the counties. Del Norte and Trinity county have the two most missing dates. We couldn't find many commonalities in when they both missed days, but in the first two weeks of November 2020, both counties didn't report their AQI. One explanation we came up with is that both counties experienced outages during the time and weren't able to collect data as a result.

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.

[24]:
```
aqi_per_month = daily_county.groupby("Month").agg(np.mean)["AQI"]
aqi_per_month_CA = epa_data_CA_merged.groupby("Month").agg(np.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
```

[25]:
```
grader.check("q5a")
```

[25]:
```
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?

The AQI per month value for September is really large! In early September 2020, there was a "record-breaking heat wave and strong katabatic winds" that caused one of the largest wildfire.

Source: https://en.wikipedia.org/wiki/2020_California_wildfires

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?

If the average AQI per year is the same, then I would just use a constant model $f(x) = \theta$ where θ is the average AQI per year. If we see a linear increase, then we should use a simple linear regression model $g(x) = \theta_0 + \theta_1 x$.

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.

[26]:
```
def bucket_data(table, aggfunc, resolution):
    copy = table.copy()
    long_buckets = np.linspace(copy['Longitude'].min(), copy['Longitude'].
 ↪max(), num=resolution, endpoint=False)
    lat_buckets = np.linspace(copy['Latitude'].min(), copy['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())

    copy["lat_bucket"] = copy["Latitude"].apply(get_lat_bucket_num)
    copy["long_bucket"] = copy["Longitude"].apply(get_long_bucket_num)

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

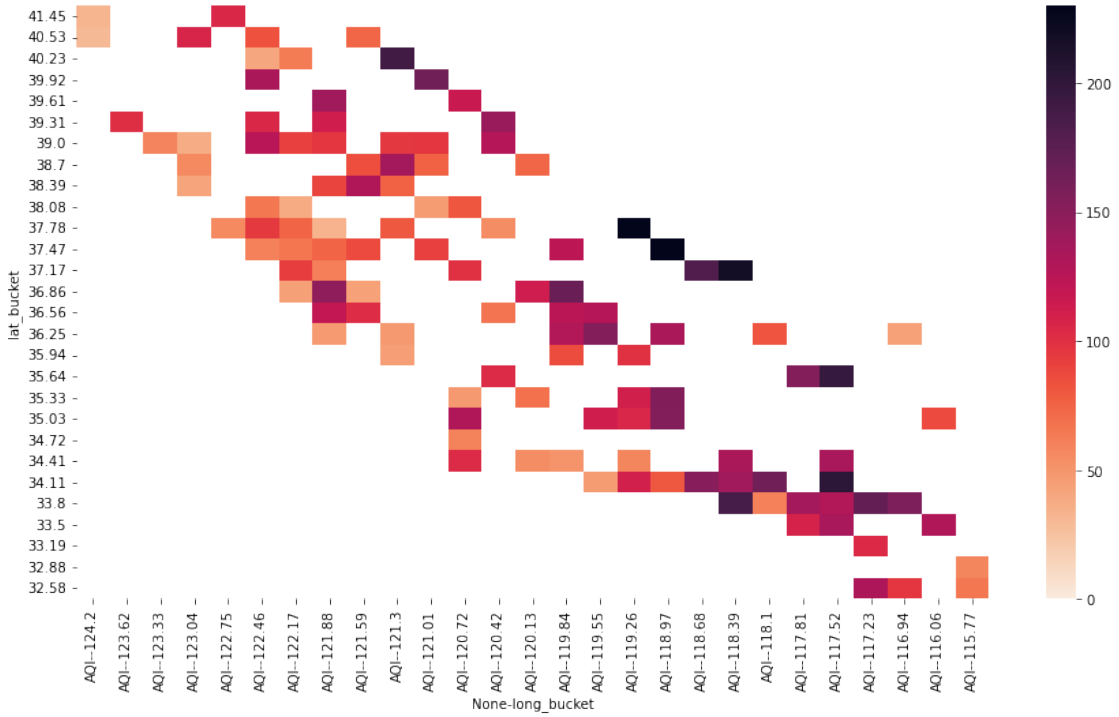
[27]:
```
grader.check("q6a")
```

[27]:
```
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.

[28]:
```
epa_data_CA_merged_sep = epa_data_CA_merged[epa_data_CA_merged["Month"]==9]
heatmap_data = bucket_data(epa_data_CA_merged_sep, np.median, 30)


#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()
```



[29]:
```
grader.check("q6b")
```

[29]:
```
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?

It makes sense after searching those dark regions on the google map. The black pixel at the bottomright corner with approximate Latitude 34.16, Longitude -117.3 locates at San Barnardino, which is near Angeles National Forest and San Bernardino National Forest. The black pixel with approximate Latitude 35.74, Longitude -117.6 locates near Sierra National Forest and Sequoia National Forest. The two black pixels at the middle, respectively with approximate Latitude 37.64, Longitude -119.1 and approximate Latitude 37.96, Longitude -119.4, are located within the Yosemite National Park. Therefore, forest fires can be the reason that make AQI levels in these regions especailly high.

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.

[30]:
```
filenames = ["daily_no2", "daily_temp", "daily_wind", "daily_ozone",␣
 ↪"daily_so2", "daily_co"]
epa_data_CA_replicate = epa_data_CA.copy()
final_df = epa_data_CA["daily_county_aqi"].copy()
final_df = final_df[["County Code", "Date", "county Name", "AQI"]]
final_df["County Code"] = final_df["County Code"].astype(str)
for dataframe in filenames:
    current = epa_data_CA_replicate[dataframe]
    current = current[["County Code", "Date Local", "Arithmetic Mean"]]
    current = current.groupby(["County Code", "Date Local"]).agg(np.mean).
 ↪reset_index()
    current = current.rename(columns={"Arithmetic Mean": dataframe + " Daily␣
 ↪Mean"})
    current["County Code"] = current["County Code"].astype(str)
```

```
    final_df = pd.merge(final_df, current, left_on = ["Date", "County Code"],␣
 ↪right_on = ["Date Local", "County Code"])
    print(len(final_df))
final_df = final_df.rename(columns={"daily_temp Daily Mean" : "daily_wind Daily␣
 ↪Mean", "daily_wind Daily Mean":"daily_temp Daily Mean"})
final_df = final_df.drop(["Date Local_x", "Date Local_y"],axis = 1)
final_df
```

```
12013
8262
8073
8070
3441
3049
```

[30]:

	County Code	Date	county Name	AQI	daily_no2 Daily Mean	\
0	19	2020-01-01	Fresno	130	12.088696	
1	19	2020-01-02	Fresno	102	12.826087	
2	19	2020-01-03	Fresno	88	15.493043	
3	19	2020-01-04	Fresno	66	9.514743	
4	19	2020-01-05	Fresno	55	5.443123	
…	…	…	…	…	…	
3044	85	2020-12-27	Santa Clara	47	11.807807	
3045	85	2020-12-28	Santa Clara	39	14.648222	
3046	85	2020-12-29	Santa Clara	53	18.970887	
3047	85	2020-12-30	Santa Clara	55	20.201373	
3048	85	2020-12-31	Santa Clara	56	15.589614	

	daily_wind Daily Mean	daily_temp Daily Mean	daily_ozone Daily Mean	\
0	82.127344	48.035417	0.014519	
1	74.927604	47.408333	0.010873	
2	59.380208	50.284375	0.013696	
3	115.795833	52.025000	0.019961	
4	116.238802	46.796354	0.017775	
…	…	…	…	
3044	80.337500	50.750000	0.021691	
3045	137.173530	50.470588	0.016103	
3046	81.280000	44.400000	0.013103	
3047	107.839583	47.000000	0.014544	
3048	146.066667	50.266667	0.021604	

	daily_so2 Daily Mean	daily_co Daily Mean
0	0.129166	0.563523
1	0.052084	0.520751
2	0.127083	0.531785
3	0.043750	0.342258
4	0.000000	0.233755

```
…                    …                    …
3044                 0.269048             0.513865
3045                 0.063095             0.454125
3046                 0.213095             0.643625
3047                 0.128421             0.655126
3048                 0.133823             0.468062

[3049 rows x 10 columns]
```

1.7.2 Question 7b - Visualization

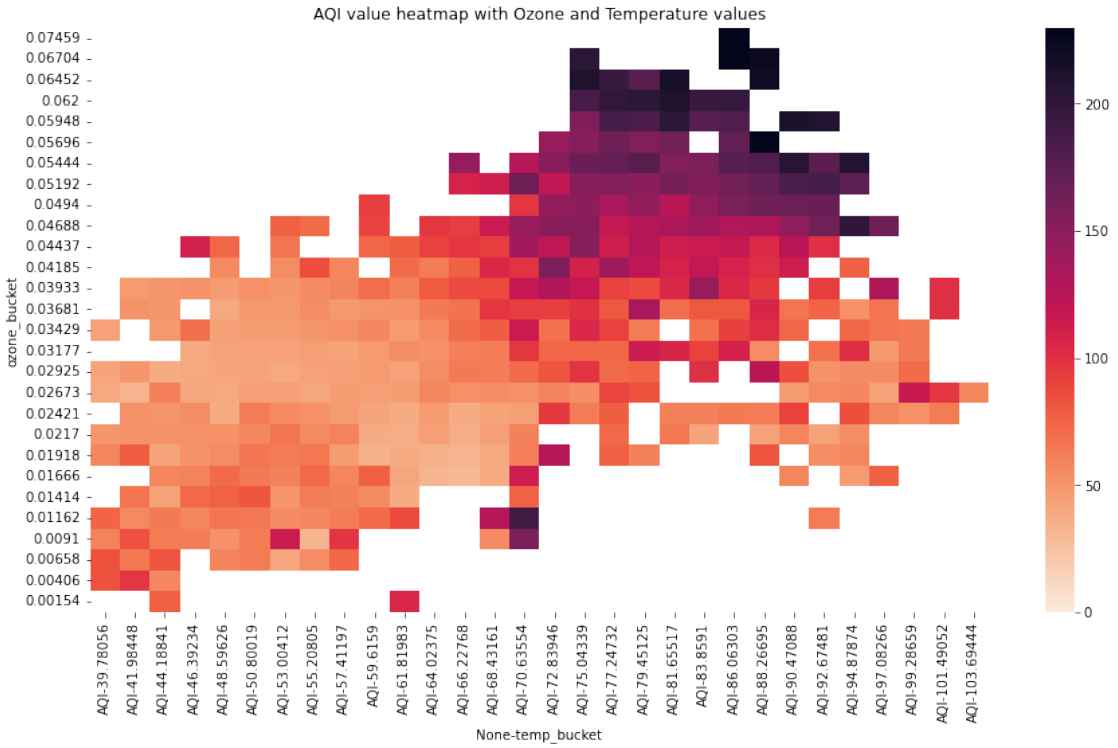
Please create **two** visualizations to summarize your analysis above. The only restrictions are that these visualizations **cannot** simply be scatterplots between two features in the dataset(s) and **cannot** be of the same type (dont make two bar graphs, for example).

[31]:
```
def bucket_data_for_ozone_and_temp(table, aggfunc, resolution):
    temp_buckets = np.linspace(table['daily_temp Daily Mean'].min(),␣
 ↪table['daily_temp Daily Mean'].max(), num=resolution, endpoint=True)
    ozone_buckets = np.linspace(table['daily_ozone Daily Mean'].min(),␣
 ↪table['daily_ozone Daily Mean'].max(), num=resolution, endpoint=True)
    temp_buckets_edges = np.append(temp_buckets, 10000)
    ozone_buckets_edges = np.append(ozone_buckets, 10000)
    table["temp_bucket"] = pd.cut(table["daily_temp Daily Mean"],␣
 ↪bins=temp_buckets_edges, labels = temp_buckets.round(5), right = False)
    table["ozone_bucket"] = pd.cut(table["daily_ozone Daily Mean"],␣
 ↪bins=ozone_buckets_edges, labels = ozone_buckets.round(5), right = False)
    pivot_result = table.pivot_table(columns = ["temp_bucket"], index =␣
 ↪["ozone_bucket"], values = ["AQI"], aggfunc = aggfunc)
    return pivot_result

bucket_data_for_ozone_and_temp(final_df, np.mean, 5)
```

[31]:
```
                    AQI
temp_bucket     39.78056    55.75903     71.7375    87.71597 103.69444
ozone_bucket
0.00154        66.257778   64.052632  143.250000   59.600000       NaN
0.01981        54.739130   57.752372   89.665399   66.760563      58.0
0.03807        59.205882   90.060000  130.162228  135.046875       NaN
0.05633              NaN  143.000000  181.057692  205.300000       NaN
0.07459              NaN         NaN         NaN  236.000000       NaN
```

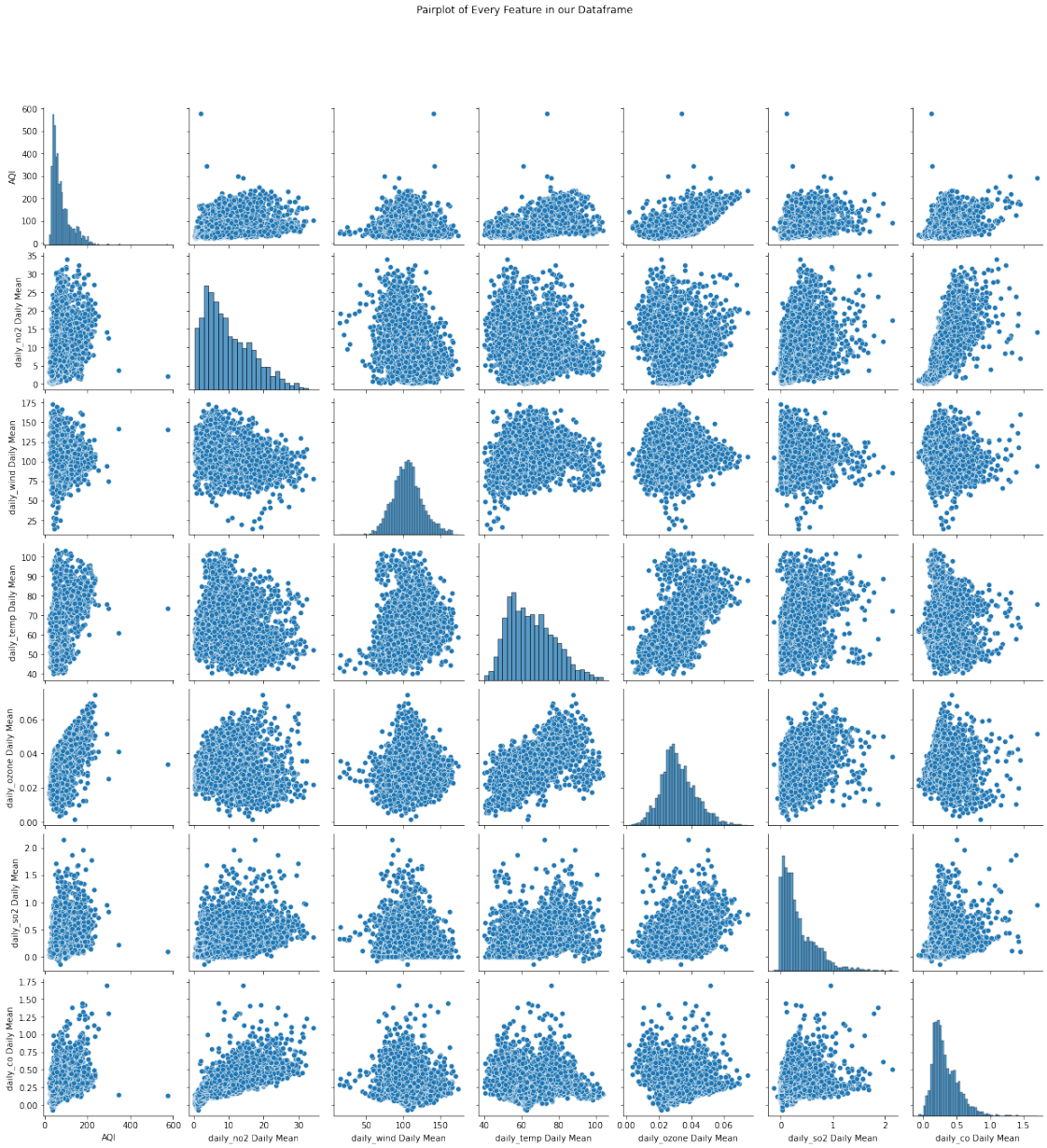
[32]:
```
# Open-ended EDA Heatmap
heatmap_ozone_temp = bucket_data_for_ozone_and_temp(final_df, np.median, 30)

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

```
plt.title("AQI value heatmap with Ozone and Temperature values")
plt.show()
```



[33]:
```
# Open-ended EDA Pairplot
g = sns.pairplot(final_df)
g.fig.suptitle("Pairplot of Every Feature in our Dataframe", y=1.08)
```

[33]: Text(0.5, 1.08, 'Pairplot of Every Feature in our Dataframe')



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.

We made a line plot that shows the changes in the average AQI on different months for both California and across all the states. We found it interesting that there seems to be a seasonal pattern to the mean AQI per month where the mean AQI generally increases from January to August, reaches its peak in September, and decreases from September to December. In addition to understanding the change in AQI for the different states and their location, we also want to

discover whether there is any relationship between AQI and the temperature, wind, particulate matter, and chemical concentrations. Therefore, we created a dataframe with the AQI data and the arithmetic mean for each of the possible features merged using the defining site and the dates. After creating the data frame, we made a pairplot that would show all the relationships between each of the variables against each other. We see a positive linear relationship between co and no2, ozone and temperature. We also see a negative linear relationship between ozone and co, ozone and no2. We also observed that there exist some weak to moderate positive correlations between AQI and so2, co, no2, or temperature; a weak negative linear correlation between AQI and wind. We also created a heatmap for ozone and temperature values to see their influence on AQI values. We find that both of them tend to have a postive relationship to AQI values. On the one hand, at the bottomleft corner, both ozone and temperature values are small, and the AQI values are small. On the other hand, the topright corner marks a cluster of dark pixels that suggest high AQI values when ozone and temperature values are high.

1.8 Part 4: Guided Modeling

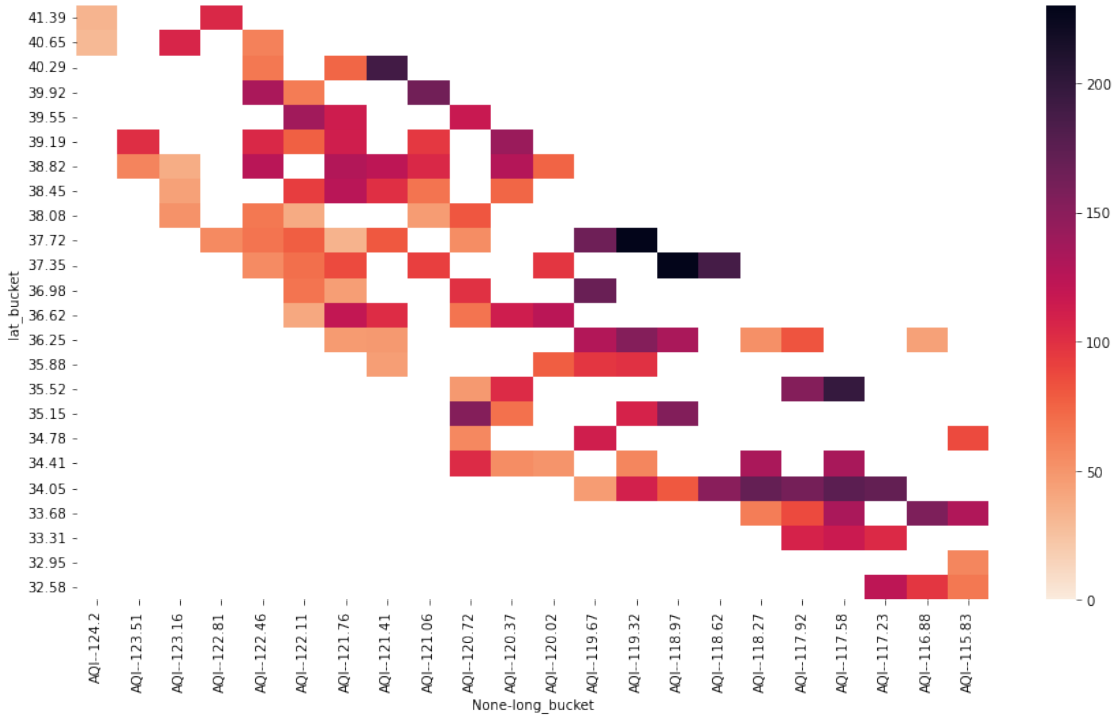
For this part, we will be looking at some open-ended modeling approaches to answering the question of predicting AQI given a location and a date.

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.

[34]:
```
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()
```



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.

[35]:
```
import math
def fill_bucket(pivot_table, lat_bucket, lon_bucket):
    interpolable = False
    has_left = False
    has_right = False
    has_above = False
    has_below = False
    closest_left = 0
    closest_right = 0
    closest_above = 0
    closest_below = 0
    num_closest = 0
    if not math.isnan(pivot_table.iloc[lat_bucket, lon_bucket]):
        interpolable = True
        return pivot_table.iloc[lat_bucket, lon_bucket]
    for i in np.arange(lon_bucket):
        if not math.isnan(pivot_table.iloc[lat_bucket, i]):
            has_left = True
            closest_left = pivot_table.iloc[lat_bucket, i]
    for i in np.arange(pivot_table.shape[1]-1, lon_bucket, -1):
        if not math.isnan(pivot_table.iloc[lat_bucket, i]):
            has_right = True
            closest_right = pivot_table.iloc[lat_bucket, i]
    for i in np.arange(lat_bucket):
        if not math.isnan(pivot_table.iloc[i, lon_bucket]):
            has_above = True
            closest_above = pivot_table.iloc[i, lon_bucket]
    for i in np.arange(pivot_table.shape[0]-1, lat_bucket, -1):
        if not math.isnan(pivot_table.iloc[i, lon_bucket]):
            has_below = True
            closest_below = pivot_table.iloc[i, lon_bucket]
    if [has_left, has_right, has_above, has_below].count(True) >= 3:
        interpolable  = True
    if not interpolable:
        return "NaN"
    else:
        num_closest = np.count_nonzero([closest_left, closest_right,␣
↪closest_above, closest_below])
        return np.sum([closest_left, closest_right, closest_above,␣
↪closest_below]) / num_closest
```

[36]:
```
grader.check("q8a")
```

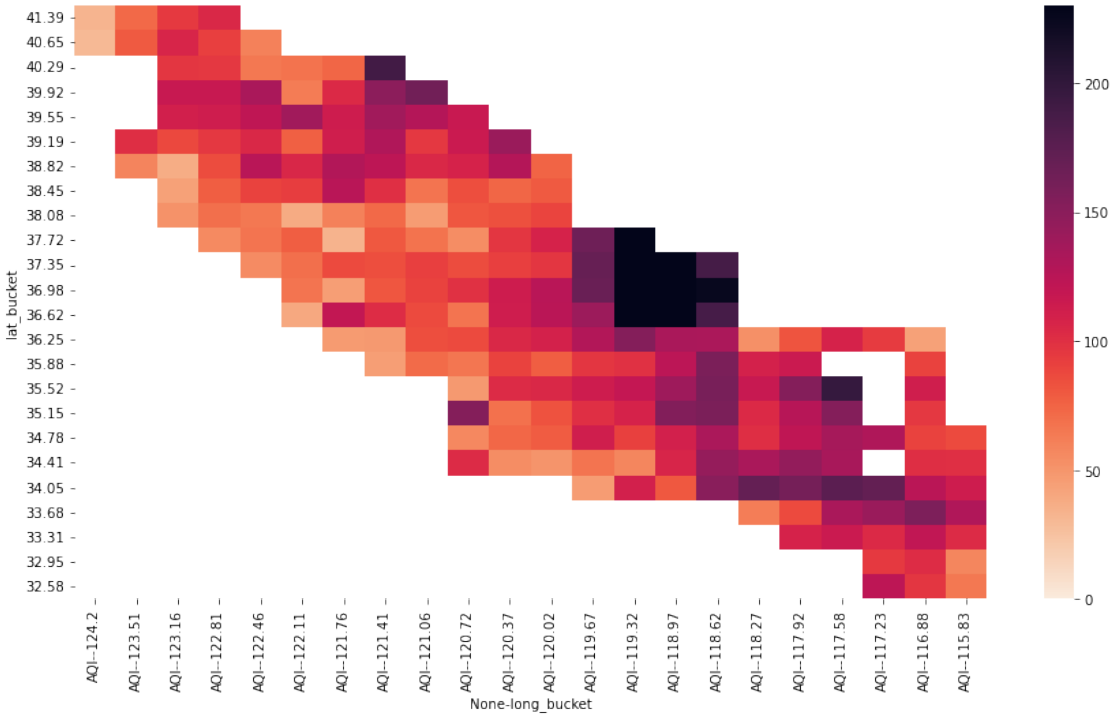
[36]:
```
q8a results: All test cases passed!
```

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.

[37]:
```
def fill_all(pivot_table):
    filled_table = pivot_table.copy()
    for i in np.arange(pivot_table.shape[0]):
        for j in np.arange(pivot_table.shape[1]):
            filled_table.iloc[i][j] = fill_bucket(filled_table, i, j)
    return filled_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()
```



[38]:
```
grader.check("q8b")
```

[38]:
```
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.

Another approach to interpolate could be described as follows:
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`. Every such cell has an attibute to record its distance to the target cell. The same goes for the rest of the directions. For the cases below, assuming that our current cell is called cell K. For each cell in the pivot table: 1. if cell K is not "NaN", just return the AQI at that given cell 2. only if there are at lest three valid directional cells, we call K interpolable. If K is interpolable, then interpolate K by assigning it an AQI value equal to m (m is equal to the average of l, r, a, b, while each of l, r, a, b is the weighted average of the cells in its category. For example, l can be calculated by the weighted average of all L cells, while the weight is inversely proportional to their distance to the target cell) in each of the valid directions. 3. If K is not interpolable, then do nothing and return "NaN".

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?

Since we want to penalize big mistakes more than small mistakes, we designed a custom loss function where predictions that fall farther from the actual category get penalized more. We calculate how far a prediction is by assigning numerical values to each category of AQI (good: 0, moderate: 1, unhealthy sensitive groups: 2, unhealthy: 3, very unhealthy: 4, hazardous: 5, these pairings are analogous to their index in a list sorted by severity of AQI, ascending). We calculate the loss as equal to the 4 raised to the absolute difference between the index of the predicted category and index of the actual category.

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 9 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.)?

We will use ozone, temperature, wind, and traffic AADT of each CA county to predict the California AQI. We believe these features have predictive power because ozone itself is a harmful air pollutant; air temperature and wind affects the movement of air, and thus the movement of air pollution; traffic AADT can be strong emission sources of polluted gases, which means higher traffic AADT should be related to worse AQI categories. To build our model, we will use a random forest model.

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.

[39]:
```
gpd_epa_traffic_reduced = gpd_epa_traffic[["county Name", "Month",␣
 ↪"Day","Defining Site", "AADT"]]
gpd_epa_traffic_reduced["Date Local"] = "2020-" +␣
 ↪gpd_epa_traffic_reduced["Month"].astype(str).str.zfill(2) + "-"+␣
 ↪gpd_epa_traffic_reduced["Day"].astype(str).str.zfill(2)
gpd_epa_traffic_reduced = gpd_epa_traffic_reduced.drop(["Month", "Day"], axis =␣
 ↪1)
gpd_epa_traffic_reduced



final_df = pd.merge(epa_data_CA["daily_county_aqi"].copy(),␣
 ↪gpd_epa_traffic_reduced, left_on = ["Date", "Defining Site"], right_on =␣
 ↪["Date Local", "Defining Site"])
final_df = final_df.rename(columns={"county Name_x": "county Name"})
final_df




filenames = ["daily_temp", "daily_wind", "daily_ozone"]
final_df = final_df[["County Code", "Date", "county Name", "AQI", "AADT"]]
final_df["County Code"] = final_df["County Code"].astype(str)
for dataframe in filenames:
    current = epa_data_CA[dataframe].copy()
    current = current[["County Code", "Date Local", "Arithmetic Mean",␣
 ↪"Observation Count"]]
    current = current.groupby(["County Code", "Date Local"]).agg(np.mean).
 ↪reset_index()
    current = current.rename(columns={"Arithmetic Mean": dataframe + " Daily␣
 ↪Mean"})
    current = current.rename(columns={"Observation Count": dataframe + " Daily␣
 ↪Count"})
    current["County Code"] = current["County Code"].astype(str)
    final_df = pd.merge(final_df, current, left_on = ["Date", "County Code"],␣
 ↪right_on = ["Date Local", "County Code"])
final_df = final_df.rename(columns={"daily_temp Daily Mean" : "daily_wind Daily␣
 ↪Mean", "daily_wind Daily Mean":"daily_temp Daily Mean"})
```

```
final_df = final_df.rename(columns={"daily_temp Daily Count" : "daily_wind␣
 ↪Daily Count", "daily_wind Daily Count":"daily_temp Daily Count"})
final_df = final_df.drop(["Date Local_x", "Date Local_y", "Date Local"],axis =␣
 ↪1)
final_df_cleaned = final_df.copy()
final_df_cleaned
```

```
#This is where we import external dataset and merge into our table, to pass the␣
 ↪autograder, we just leave it commented

#population_density = pd.read_csv("data/Population-Density-By-County.csv")

#population_density_ca = population_density[population_density["GEO.
 ↪display-label"]=="California"]

# get the county name without the "County"
#counties = population_density_ca["GCT_STUB.display-label"].str.split(pat="␣
 ↪County").str[0]

# replace county column
#population_density_ca["GCT_STUB.display-label"] = counties

#final_df_cleaned = final_df_cleaned.merge(population_density_ca, how='left',␣
 ↪left_on="county Name", right_on="GCT_STUB.display-label")
#final_df_cleaned.head()
```

```
final_df_cleaned['Month'] = pd.to_datetime(final_df_cleaned['Date']).dt.month
final_df_cleaned["date loc"] = final_df_cleaned["county Name"] +␣
 ↪final_df_cleaned["Month"].astype(str)
final_df_cleaned
```

```
t_features = final_df_cleaned[["daily_wind Daily Mean", "daily_temp Daily␣
 ↪Mean", "daily_ozone Daily Mean","AADT"]]
features = t_features
```

```
targets = final_df_cleaned["AQI"]
for i in range(len(targets)):
    if targets[i] <= 50:
        targets[i] = "good"
    elif targets[i] > 50 and targets[i] <=100:
        targets[i] = "moderate"
    elif targets[i] > 100 and targets[i] <= 150:
        targets[i] = "unhealthy sensitive groups"
    elif targets[i] > 150 and targets[i] <= 200:
        targets[i] = "unhealthy"
    elif targets[i] > 200 and targets[i] <= 300:
        targets[i] = "very unhealthy"
    else: targets[i] = "hazardous"
```

[40]:
```
grader.check("q10b")
```

[40]:
```
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.

[41]:
```
from sklearn.ensemble import RandomForestClassifier

def build_model(features, targets):
    rf = RandomForestClassifier()
    rf.fit(features, targets)
    return rf
```

[42]:
```
grader.check("q10c")
```

[42]:
```
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.

[43]:
```
```

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

def predict(model, features):
    y_pred_rf = model.predict(features)
    return y_pred_rf
```

[44]:
```
grader.check("q10d")
```

[44]:
```
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`.

[45]:
```
# Our designed loss function

aqi_to_index = dict()
for i in range(len(categories)):
    aqi_to_index[categories[i]] = i
aqi_to_index



def distance_Loss(y, yhat):
    ans = 0
    for i in range(len(y)):
        ans += 4**np.abs(aqi_to_index[y[i]] - aqi_to_index[yhat[i]])

    return ans
```

[46]:
```
# Model validation and performance

from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(features, targets,␣
 ↪test_size = 0.3)

model = build_model(X_train, y_train)
prediction = predict(model, X_test)


y_test_category = []
for i in y_test:
    y_test_category.append(i)

prediction = pd.Series(prediction)
y_test_category = pd.Series(y_test_category)
print(prediction)


binary_error = sum(prediction != y_test_category)/ len(prediction)
cv_error = distance_Loss(y_test_category, prediction)

print("cv_error: " + str(cv_error))
print("binary_error: " + str(binary_error))
```

```
0           good
1           good
2           good
3           good
4           good
          …
3746    moderate
3747    moderate
3748    moderate
3749    moderate
3750    moderate
Length: 3751, dtype: object
cv_error: 14452
binary_error: 0.26526259664089574
```

[47]:
```
grader.check("q10e")
```

[47]:
```
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.

[48]:
```
# importation of binary encoder ce for input transformation

import sys
!conda install --yes --prefix {sys.prefix} category_encoders
import category_encoders as ce
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 4.10.3
  latest version: 4.11.0

Please update conda by running

    $ conda update -n base conda



# All requested packages already installed.
```

[49]:
```
# Standardization function to standardize input values before modeling

def standardize(feature):
    for col_name in feature.columns:
        feature_mean = np.mean(feature[col_name])
        feature_std = np.std(feature[col_name])
        feature[col_name] = (feature[col_name]-feature_mean) / feature_std
    return feature
```

[50]:
```
# Work to merge data, build model, and perform cross validation on baseline␣
 ↪model

social_factors = pd.read_csv("data/social-ca.csv")
final_df_cleaned = final_df_cleaned.merge(social_factors, how='left',␣
 ↪left_on="county Name", right_on="County")
```

```
population_density = pd.read_csv("data/Population-Density-By-County.csv")

population_density_ca = population_density[population_density["GEO.
 ↪display-label"]=="California"]

# get the county name without the "County"
counties = population_density_ca["GCT_STUB.display-label"].str.split(pat="␣
 ↪County").str[0]

# replace county column
population_density_ca["GCT_STUB.display-label"] = counties

final_df_cleaned = final_df_cleaned.merge(population_density_ca, how='left',␣
 ↪left_on="county Name", right_on="GCT_STUB.display-label")

b_features = final_df_cleaned[["daily_wind Daily Mean", "daily_temp Daily␣
 ↪Mean", "daily_ozone Daily Mean"]]#, "Density per square mile of land area",␣
 ↪"total_AADT"]]
b_features = standardize(b_features)
b_targets = final_df_cleaned["AQI"]

X_train_binary, X_test_binary, y_train_binary, y_test_binary =␣
 ↪train_test_split(b_features, b_targets, test_size = 0.3)

from sklearn.model_selection import KFold
kf_base = KFold(n_splits=5)
binary_error_base = []
cv_error_base = []
for train_index, test_index in kf_base.split(X_train_binary):
    X_train, X_test = X_train_binary.to_numpy()[train_index], X_train_binary.
 ↪to_numpy()[test_index]
    y_train, y_test = y_train_binary.to_numpy()[train_index], y_train_binary.
 ↪to_numpy()[test_index]
    model = build_model(X_train, y_train)
    prediction = predict(model, X_test)
    y_test_category = []
    for i in y_test:
        y_test_category.append(i)
    prediction = pd.Series(prediction)
    y_test_category = pd.Series(y_test_category)
    binary_error_base.append(sum(prediction != y_test_category) /␣
 ↪len(prediction))
    cv_error_base.append(distance_Loss(y_test_category, prediction))

overall_model_base = build_model(X_train_binary, y_train_binary)
```

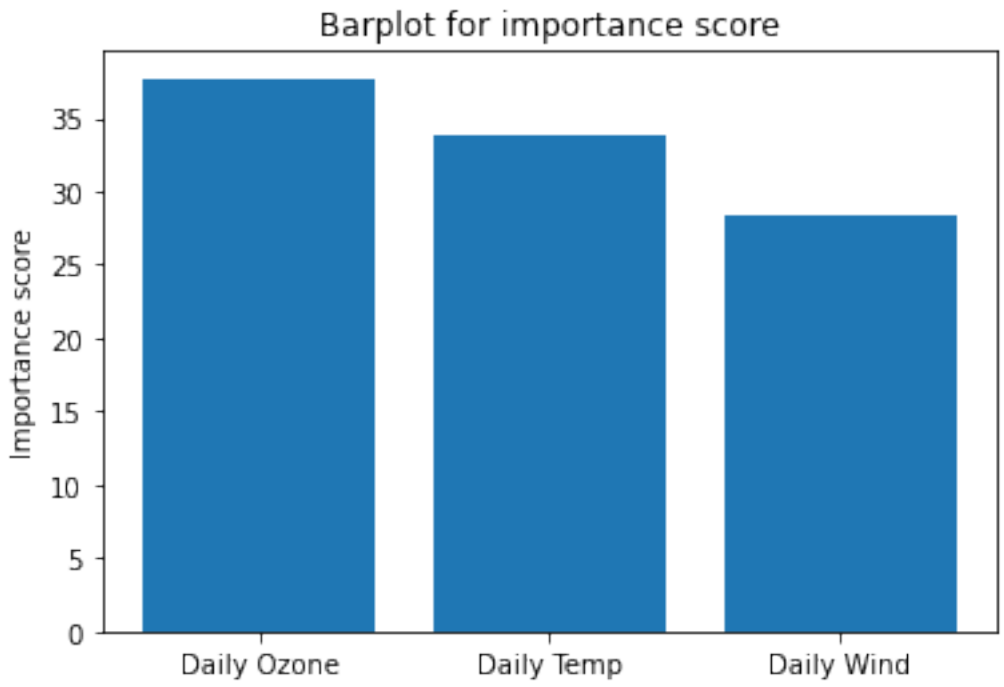
```
important_feature = pd.DataFrame({'Feature' : X_train_binary.columns,
          'Importance score': 100*overall_model_base.feature_importances_}).
 ↪round(1)
important_feature = important_feature.sort_values(['Importance score'],␣
 ↪ascending = False)


print("cv_error calculated: " + str(cv_error_base))
print("binary_error calculated: " + str(binary_error_base))
print("cv_error calculated mean: " + str(np.mean(cv_error_base)))
print("binary_error calculated mean: " + str(np.mean(binary_error_base)))
print("importance score of every feature: ")
important_feature
```

```
cv_error calculated: [6539, 6800, 7429, 6784, 6997]
binary_error calculated: [0.3632210165619646, 0.36950314106225013, 0.348,
0.36228571428571427, 0.35942857142857143]
cv_error calculated mean: 6909.8
binary_error calculated mean: 0.3604876886677001
importance score of every feature:
```

[50]:

	Feature	Importance score
2	daily_ozone Daily Mean	37.7
1	daily_temp Daily Mean	33.9
0	daily_wind Daily Mean	28.4

[51]:
```
# Barplot for baseline model

plt.bar(x=important_feature["Feature"], height=important_feature["Importance␣
 ↪score"], tick_label = ["Daily Ozone", "Daily Temp", "Daily Wind"])
plt.title("Barplot for importance score")
plt.ylabel('Importance score')
plt.show()
```



[52]:
```
# KDE Plot for baseline model binary errors and cv errors
# This took too long to run so we only did it once but the results are in the␣
 ↪design doc

"""
11a binary error distribution
feature_bin_3 = []
feature_cv_3 = []
for _ in range(100):
    X_train_binary, X_test_binary, y_train_binary, y_test_binary =␣
 ↪train_test_split(b_features, b_targets, test_size = 0.3)
    binary_error_base = []
    cv_error_base = []
    for train_index, test_index in kf_base.split(X_train_binary):
        X_train, X_test = X_train_binary.to_numpy()[train_index],␣
 ↪X_train_binary.to_numpy()[test_index]
        y_train, y_test = y_train_binary.to_numpy()[train_index],␣
 ↪y_train_binary.to_numpy()[test_index]
        model = build_model(X_train, y_train)
        prediction = predict(model, X_test)
        y_test_category = []
        for i in y_test:
            y_test_category.append(i)
        prediction = pd.Series(prediction)
```

```
        y_test_category = pd.Series(y_test_category)
        binary_error_base.append(sum(prediction != y_test_category) /␣
 ↪len(prediction))
        cv_error_base.append(distance_Loss(y_test_category, prediction))

    feature_bin_3.append(np.mean(binary_error_base))
    feature_cv_3.append(np.mean(cv_error_base))
print(np.mean(feature_bin_3))
print(np.mean(feature_cv_3))"""
```

[52]:
```
'\n11a binary error distribution\nfeature_bin_3 = []\nfeature_cv_3 = []\nfor _
in range(100):\n    X_train_binary, X_test_binary, y_train_binary, y_test_binary
= train_test_split(b_features, b_targets, test_size = 0.3)\n
binary_error_base = []\n    cv_error_base = []\n    for train_index, test_index
in kf_base.split(X_train_binary):\n        X_train, X_test =
X_train_binary.to_numpy()[train_index], X_train_binary.to_numpy()[test_index]\n
y_train, y_test = y_train_binary.to_numpy()[train_index],
y_train_binary.to_numpy()[test_index]\n        model = build_model(X_train,
y_train)\n        prediction = predict(model, X_test)\n        y_test_category =
[]\n        for i in y_test:\n            y_test_category.append(i)\n
prediction = pd.Series(prediction)\n        y_test_category =
pd.Series(y_test_category)\n        binary_error_base.append(sum(prediction !=
y_test_category) / len(prediction))\n
cv_error_base.append(distance_Loss(y_test_category, prediction))\n\n
feature_bin_3.append(np.mean(binary_error_base))\n    feature_cv_3.append(np.mea
n(cv_error_base))\nprint(np.mean(feature_bin_3))\nprint(np.mean(feature_cv_3))'
```

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.

In our baseline model, our general idea is to use a random forest model to predict the AQI category for every data point we find in California. The AQI category is calculated based on the standard sorting criteria of air quality in Q10. The inputs for our model are the daily mean value of wind speed, daily mean value of temperature, and daily mean value of ozone for all data points in a daily county level. They are organized into a dataframe that each column of the dataframe is represented by one variable. The outputs, according to our random forest model, should be a series of predicted AQI categories that we generated from the model. For the cross_validation error, we calculated the cross validation error for both cv_error and binary_error. We find that the cv_error and the binary_error are relatively high, with cv_error a value of about 7000 and binary_error of about 0.37. We also outputed an importance score table that reveals the importance for each of the features used in the model. We plan to change the baseline model by adding more useful features that are characteristics of locations from external datasets. Also, we plan to add a creative feature called dateloc it incorporates seasonal as well as locational influence. We will generate this variable through binary encoding and transform this categorical variable into numerical ones for our later

analysis.

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.

[53]:
```
# Work to build model and perform cross validation on improved model

i_features = final_df_cleaned[['daily_wind Daily Mean', "daily_temp Daily␣
 ↪Mean", "daily_ozone Daily Mean", "AADT", "date loc", 'Density per square␣
 ↪mile of land area', 'Smokers']]
i_targets = final_df_cleaned["AQI"]
i_encoder = ce.BinaryEncoder(cols=["date loc"], return_df=True)
i_features = i_encoder.fit_transform(i_features)

X_train_binary, X_test_binary, y_train_binary, y_test_binary =␣
 ↪train_test_split(i_features, i_targets, test_size = 0.3)

from sklearn.model_selection import KFold
kf_improved = KFold(n_splits=5)
binary_error_improved = []
cv_error_improved = []
for train_index, test_index in kf_base.split(X_train_binary):
    X_train, X_test = X_train_binary.to_numpy()[train_index], X_train_binary.
 ↪to_numpy()[test_index]
    y_train, y_test = y_train_binary.to_numpy()[train_index], y_train_binary.
 ↪to_numpy()[test_index]
    model = build_model(X_train, y_train)
    prediction = predict(model, X_test)
    y_test_category = []
    for i in y_test:
        y_test_category.append(i)
    prediction = pd.Series(prediction)
    y_test_category = pd.Series(y_test_category)
    binary_error_improved.append(sum(prediction != y_test_category) /␣
 ↪len(prediction))
    cv_error_improved.append(distance_Loss(y_test_category, prediction))

overall_model_improved = build_model(X_train_binary, y_train_binary)
important_feature = pd.DataFrame({'Feature' : X_train_binary.columns,
          'Importance score': 100*overall_model_improved.feature_importances_}).
 ↪round(1)
```

```
important_feature = important_feature.sort_values(['Importance score'],␣
 ↪ascending = False)
sum_date_loc = round(sum(important_feature.iloc[6:
 ↪len(important_feature)]["Importance score"]), 1)
important_feature.loc[12] = {"Feature": "date loc", "Importance score":␣
 ↪sum_date_loc}
important_feature = important_feature.iloc[:7]
important_feature = important_feature.sort_values(['Importance score'],␣
 ↪ascending = False)


print("cv_error calculated: " + str(cv_error_improved))
print("binary_error calculated: " + str(binary_error_improved))
print("cv_error calculated mean: " + str(np.mean(cv_error_improved)))
print("binary_error calculated mean: " + str(np.mean(binary_error_improved)))
print("importance score sum for meteorological features: " +␣
 ↪str(round(sum(important_feature[important_feature["Feature"].str.
 ↪contains("daily")]["Importance score"]), 1)))
print("importance score sum for locational features: " + str(round(100 -␣
 ↪sum(important_feature[important_feature["Feature"].str.
 ↪contains("daily")]["Importance score"]), 1)))
print("importance score of every feature: ")
important_feature
```

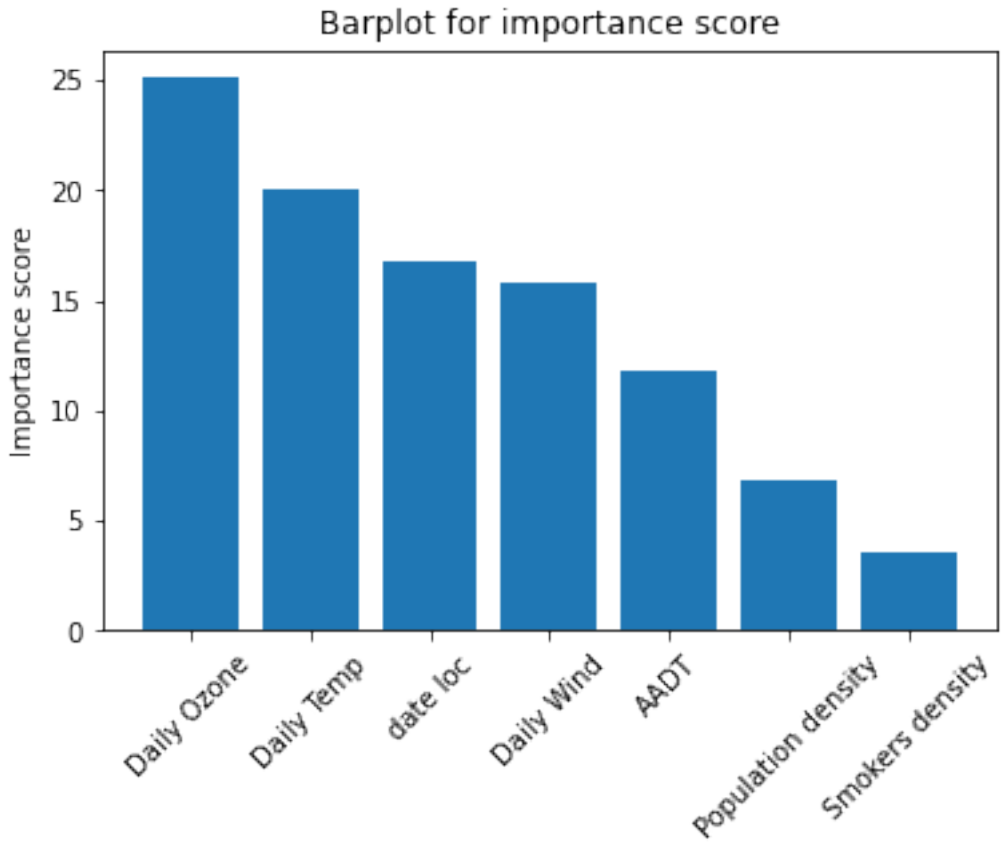
```
cv_error calculated: [4427, 4499, 5599, 4246, 5326]
binary_error calculated: [0.2238720731010851, 0.23757852655625358,
0.24857142857142858, 0.256, 0.23314285714285715]
cv_error calculated mean: 4819.4
binary_error calculated mean: 0.23983297707432488
importance score sum for meteorological features: 60.9
importance score sum for locational features: 39.1
importance score of every feature:
```

[53]:

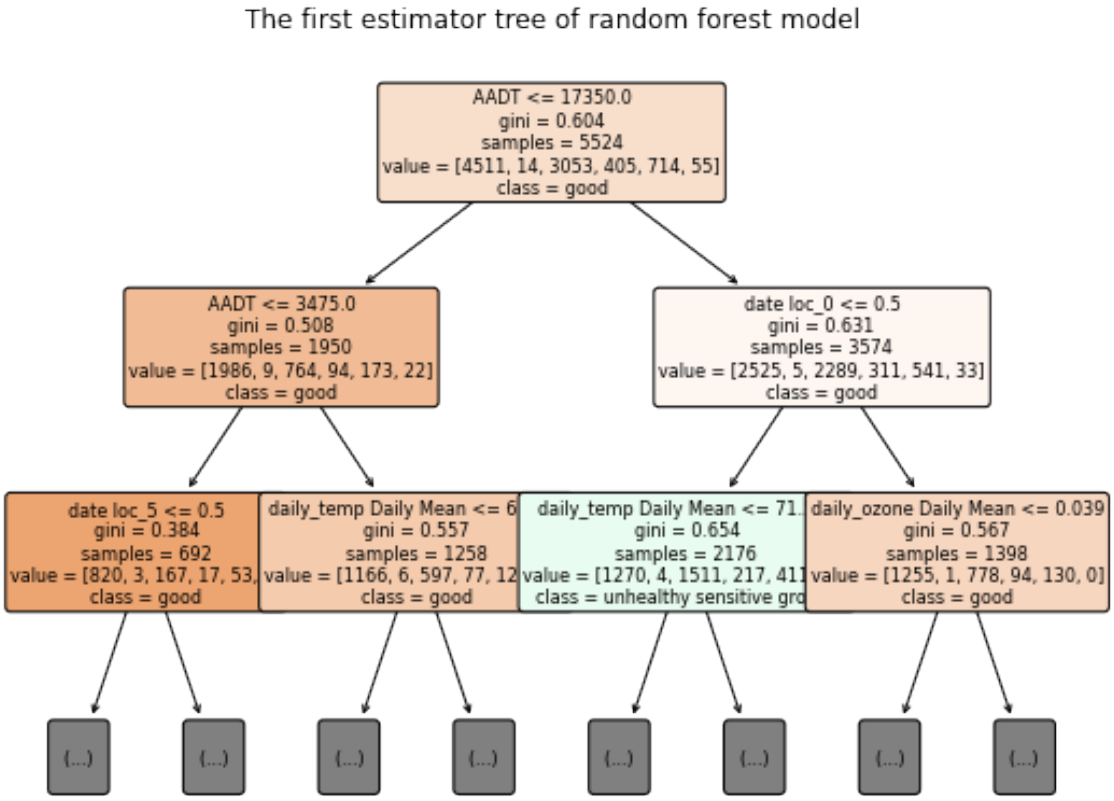
	Feature	Importance score
2	daily_ozone Daily Mean	25.1
1	daily_temp Daily Mean	20.0
12	date loc	16.8
0	daily_wind Daily Mean	15.8
3	AADT	11.8
13	Density per square mile of land area	6.9
14	Smokers	3.6

[54]:
```
# Barplot for baseline model

plt.bar(x=important_feature["Feature"], height=important_feature["Importance␣
 ↪score"], tick_label = ["Daily Ozone", "Daily Temp", "date loc", "Daily␣
 ↪Wind", "AADT", "Population density", "Smokers density"])
```

```
plt.title("Barplot for importance score")
plt.ylabel('Importance score')
plt.xticks(rotation = 45)
plt.show()
```



[55]:
```
# Plot of the first estimator tree

from sklearn.tree import plot_tree

plt.figure(figsize=(9,7))
plot_tree(overall_model_improved.estimators_[0],
          feature_names=i_features.columns,
          class_names=categories,
          filled=True,
          impurity=True,
          rounded=True,
          fontsize=8,
          max_depth=2)
plt.title("The first estimator tree of random forest model")
plt.show()
```



[56]:
```
# KDE Plot for baseline model binary errors and cv errors
# This took too long to run so we only did it once but the results are in the␣
 ↪design doc

"""
11c binary error distribution:
feature_bin_5 = []
feature_cv_5 = []
for _ in range(100):
    X_train_binary, X_test_binary, y_train_binary, y_test_binary =␣
 ↪train_test_split(i_features, i_targets, test_size = 0.3)
    binary_error_improved = []
    cv_error_improved = []
    for train_index, test_index in kf_base.split(X_train_binary):
        X_train, X_test = X_train_binary.to_numpy()[train_index],␣
 ↪X_train_binary.to_numpy()[test_index]
        y_train, y_test = y_train_binary.to_numpy()[train_index],␣
 ↪y_train_binary.to_numpy()[test_index]
        model = build_model(X_train, y_train)
```

```
        prediction = predict(model, X_test)
        y_test_category = []
        for i in y_test:
            y_test_category.append(i)
        prediction = pd.Series(prediction)
        y_test_category = pd.Series(y_test_category)
        binary_error_improved.append(sum(prediction != y_test_category) /␣
 ↪len(prediction))
        cv_error_improved.append(distance_Loss(y_test_category, prediction))
    feature_bin_5.append(np.mean(binary_error_improved))
    feature_cv_5.append(np.mean(cv_error_improved))
print(np.mean(feature_bin_5))
print(np.mean(feature_cv_5))
"""
```

[56]:
```
'\n11c binary error distribution:\nfeature_bin_5 = []\nfeature_cv_5 = []\nfor _
in range(100):\n    X_train_binary, X_test_binary, y_train_binary, y_test_binary
= train_test_split(i_features, i_targets, test_size = 0.3)\n
binary_error_improved = []\n    cv_error_improved = []\n    for train_index,
test_index in kf_base.split(X_train_binary):\n        X_train, X_test =
X_train_binary.to_numpy()[train_index], X_train_binary.to_numpy()[test_index]\n
y_train, y_test = y_train_binary.to_numpy()[train_index],
y_train_binary.to_numpy()[test_index]\n        model = build_model(X_train,
y_train)\n        prediction = predict(model, X_test)\n        y_test_category =
[]\n        for i in y_test:\n            y_test_category.append(i)\n
prediction = pd.Series(prediction)\n        y_test_category =
pd.Series(y_test_category)\n        binary_error_improved.append(sum(prediction
!= y_test_category) / len(prediction))\n
cv_error_improved.append(distance_Loss(y_test_category, prediction))\n
feature_bin_5.append(np.mean(binary_error_improved))\n    feature_cv_5.append(np
.mean(cv_error_improved))\nprint(np.mean(feature_bin_5))\nprint(np.mean(feature_
cv_5))\n'
```

[57]:
```
# Continuation of the last cell

"""
bins = {'11a': feature_bin_3, '11c': feature_bin_5}
cvs = {'11a': feature_cv_3, '11c': feature_cv_5}

import seaborn as sns
ax = sns.kdeplot(data=bins).set(xlabel='Binary Error', title="Distribution of␣
 ↪Average Binary Errors for Models from 11a and 11c after 100 Runs of␣
 ↪Cross-Validation")

sns.kdeplot(data=cvs).set(xlabel='Losses', title="Distribution of Average Loss␣
 ↪for Models from 11a and 11c after 100 Runs of Cross-Validation")
"""
```

[57]:
```
'\nbins = {\'11a\': feature_bin_3, \'11c\': feature_bin_5}\ncvs = {\'11a\':
feature_cv_3, \'11c\': feature_cv_5}\n\nimport seaborn as sns\nax =
sns.kdeplot(data=bins).set(xlabel=\'Binary Error\', title="Distribution of
Average Binary Errors for Models from 11a and 11c after 100 Runs of Cross-
Validation")\n\nsns.kdeplot(data=cvs).set(xlabel=\'Losses\', title="Distribution
of Average Loss for Models from 11a and 11c after 100 Runs of Cross-
Validation")\n'
```

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.

We added more features to our improved model. Aside from daily mean value of wind speed, daily mean value of temperature, and daily mean value of ozone, we have the feature dateloc, as described in 11b. We also have AADT (annual average traffic daily) that we draw from the traffic data, we also added the population density and smoker density both measured by density per square mile of land area. We can see significant improvements in the cross validation portion in both the cv_error and the binary_error after we add some new features with some of them as transformations of our original data and some of them drawn from external datasets. The cv_error now is about 4800 and the binary error now is about 0.23. From the baseline model, we have a cv_error of about 7000 and binary error of about 0.37, so the comparison is obvious here. We are able to test our hypothesis through this improved model. From our model and analysis, we can see that dateloc, the feature design to test the seasonality of AQI, has an importance score of $16.8 > 10$. And our improved model with newly added locational and social features has a 37.8% decrease in the binary error rate and a 34.5% decrease in average loss. However, we cannot conclude that the locational, social and time features are as important as the meteological parameters because there is a 35% difference in their importance scores, which is much larger than our 10% line. In conclusion, we cannot reject our null hypothesis. (see our report "How to Test" portion for more details)

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

[58]:
```
grader.check_all()
```

[58]:
```
q10b results: All test cases passed!

q10c results: All test cases passed!

q10d results: All test cases passed!

q10e results: All test cases passed!

q1a results: All test cases passed!

q1b results: All test cases passed!
```

```
q1c results: All test cases passed!

q2a results: All test cases passed!

q2b results: All test cases passed!

q3a results: All test cases passed!

q4a results: All test cases passed!

q4i results: All test cases passed!

q5a results: All test cases passed!

q6a results: All test cases passed!

q6b results: All test cases passed!

q8a results: All test cases passed!

q8b results: All test cases passed!
```

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()
```