#!/usr/bin/env python

# coding: utf-8

# In[47]:

import plotly.io as pio

pio.kaleido.scope.default\_format = "png"

# # 1. Data Collection

# - The data consists concentration of pollutants - Carbon Monixide (CO),PM2.5 and Ozone for different California counries for the years 2010 to 2021.

# - The raw data contains the columns:

# - <b>Date</b> : The date the observation was taken.

# - <b>Source</b> : Source of the observation.

# - <b>Site ID</b> : Site ID for the particular site in the California county.

# - <b>POC</b> : This is the "Parameter Occurrence Code" used to distinguish different instruments that measure the same parameter at the same site.

# - <b>Daily Mean Pollutant Concentration</b> : Mean concentration for the pollutant for the date.

# - <b>UNITS</b> : Units of measurement for the pollutant concentration.

# - <b>DAILY\_AQI\_VALUE</b> : Air Quality Index value for the date.

# - <b>Site Name</b> : Site Name within the County.

# - <b>DAILY\_OBS\_COUNT</b> : The number of values that comprise the daily data set.

# - <b>PERCENT\_COMPLETE</b> : The percentage of required observations (or scheduled days) made for the given assessment time period.

# - <b>AQS\_PARAMETER\_CODE</b> : The AQS code corresponding to the parameter measured by the monitor

# - <b>AQS\_PARAMETER\_DESC</b> : description assigned in AQS to the parameter measured by the monitor. Parameters may be pollutants or non-pollutants (e.g., wind speed).

# - <b>CBSA\_CODE</b> : The code of the core based statistical area (metropolitan area) where the monitoring site is located.

# - <b>CBSA\_NAME</b>: The short version of the OMB-assigned title for the core-based statistical area (CBSA).

# - <b>STATE\_CODE</b> : Code for the State in USA.

# - <b>STATE</b> : State Name

# - <b>COUNTY\_CODE</b> : County code within the State.

# - <b>COUNTY</b> : County name.

# - <b>SITE\_LATITUDE</b> : Latitude for the site.

# - <b>SITE\_LONGITUDE</b> :Longiture fir the site.

#

# - All the data pertaining to the pollutants has been stored in a public github repository : https://github.com/wadhawanabhishek/CrowdDoing

#

#

# - Each pollutant information has been stored in a separate folder within the repository as follows:-

#

# ![image.png](attachment:image.png)

# # 2. Data Transformation

# The data stored in the github repository is transformed for further analysis.

# - Data is transformed from a daily level of pollutant concentration to a monthly level.

# - Along with pollutant concentration, the AQI levels are also calculated

# - Only the data corresponding to years fed to function is transformed an further analysed.

# In[48]:

import pandas as pd

import re

import requests

from bs4 import BeautifulSoup

import plotly.express as px

# Currently we have data from 2010 - 2021

class Pollutant:

#

def \_\_init\_\_(self,pollutant:str,start\_year:int,end\_year:int):

self.\_\_pollutant = pollutant.upper()

self.\_\_start\_year = start\_year

self.\_\_end\_year = end\_year

self.\_\_master\_list = []

self.\_\_pollutant\_list = ["CO","PM2.5","OZONE"]

self.\_\_units = None

assert self.\_\_pollutant in self.\_\_pollutant\_list ,"Not a valid Pollutant"

assert isinstance(start\_year,int),"Start Year is not Integer!"

assert isinstance(end\_year,int),"End Year is not Integer!"

@property

def pollutant(self):

return self.\_\_pollutant

def \_\_get\_data(self):

#

try:

#

year\_regex = re.compile(r'\d\d\d\d')

github\_url = 'https://github.com/wadhawanabhishek/CrowdDoing/tree/main/'+self.\_\_pollutant+'\_data'

result = requests.get(github\_url)

soup = BeautifulSoup(result.text, 'html.parser')

csvfiles = soup.find\_all(title=re.compile("\.csv$"))

except:

#

print("Resuouce not Found!")

filename = [ ]

for i in csvfiles:

#

filename.append(i.extract().get\_text())

years=[]

for file in filename:

year = year\_regex.search(file)

years.append(year.group())

years =[int(i) for i in years]

# 2011-2016

if (self.\_\_start\_year < min(years)) or (self.\_\_start\_year > max(years)):

#

print("Invalid Year Range. The Start Year Does not Exist")

if (self.\_\_end\_year > max(years)) or (self.\_\_end\_year < self.\_\_start\_year) or (self.\_\_end\_year < min(years)):

#

print("Invalid Year Range. The End Year Does not Exist")

new\_lst = [i for i in range(self.\_\_start\_year,self.\_\_end\_year+1)]

check = all(item in years for item in new\_lst)

up\_file=[]

if check == True:

#

for file in filename:

for yr in new\_lst:

if str(yr) in file:

up\_file.append(file)

else:

raise Exception("The data for the given years not present")

github\_url = github\_url.replace("github.com",'raw.githubusercontent.com')

github\_url = github\_url.replace("tree/",'')

appended\_data =[]

for f in up\_file:

url = github\_url +'/'+ f

data = pd.read\_csv(url)

appended\_data.append(data)

final\_df = pd.concat(appended\_data)

self.\_\_units = final\_df['UNITS'].unique()[0]

return final\_df

def \_\_feature\_extraction(self,data\_df):

df = data\_df

df = df.iloc[:, [0,17,2,4,6,15]]

df = df[df['STATE']=='California']

return df

def \_\_get\_transformed\_data(self,f\_df):

initial\_df= f\_df

drop\_cols = ['Site ID','STATE']

for col in drop\_cols:

initial\_df = initial\_df.drop(col,axis = 1)

initial\_df = initial\_df.groupby(['COUNTY','Date']).sum()

initial\_df=initial\_df.reset\_index(['Date','COUNTY'])

initial\_df['Date']= pd.to\_datetime(initial\_df['Date'],format='%m/%d/%Y')

initial\_df['Year']= pd.to\_datetime(initial\_df['Date']).dt.to\_period('Y')

initial\_df['Month']= pd.to\_datetime(initial\_df['Date']).dt.to\_period('M')

initial\_df = initial\_df.drop("Date",axis = 1)

cols = [initial\_df.columns]

final\_df = initial\_df.groupby(['COUNTY','Year','Month']).mean()

final\_df = final\_df.reset\_index(['COUNTY','Year','Month'])

pollutant\_col = "Monthly Avg "+ self.\_\_pollutant+ " Concentration"

final\_df = final\_df.rename(columns={final\_df.columns[3]:pollutant\_col,"DAILY\_AQI\_VALUE":"Monthly\_Avg\_AQI\_VALUE"})

final\_df['Pollutant']= self.\_\_pollutant

final\_df['Units'] = self.\_\_units

return final\_df

def run(self):

data\_df = self.\_\_get\_data()

feature\_df = self.\_\_feature\_extraction(data\_df)

trans\_df = self.\_\_get\_transformed\_data(feature\_df)

return trans\_df

# In[49]:

df1 = Pollutant("co",2010,2021).run()

df2 = Pollutant("pm2.5",2010,2021).run()

df3 = Pollutant("ozone",2010,2021).run()

# In[50]:

df1.head()

# In[51]:

df2.head()

# In[52]:

df3.head()

# # 3. Concentration Trends for Different Counties Over the Years

# - After the data is transformed to the desired level, the pollutant concentration and AQI values are plotted for different years for different counties.

# - This is done to analyse whether the pollutant concentrations follow a particluar pattern in different counties and to understand the trend of pollutant concentration in different counties within California.

# - Further, The year in which the pollutant levels were maximum for most of the counties is analysed.

# - Here the function takes two arguments -

# 1. The transformed data for the pollutant analysed

# 2. conc / aqi : Whether the pollutant concentration needs to be plotted or the AQI levels.

# In[53]:

from plotly.subplots import make\_subplots

from math import ceil

import plotly.graph\_objects as go

class Plot\_Map():

def \_\_init\_\_(self,df,calc\_type:str):

self.df = df

self.typ = calc\_type.lower()

self.\_\_pollutant = None

self.units = None

assert self.typ in ['conc','aqi'], "Not a valid Calculation Type!"

def \_transform\_data(self):

self.\_\_pollutant = self.df.Pollutant.unique().tolist()[0]

units = self.df.Units.unique().tolist()[0]

self.units = units

trans\_df = self.df.drop(['Month','Pollutant','Units'],axis=1)

trans\_df= trans\_df.groupby(['COUNTY','Year']).mean()

trans\_df.reset\_index(['COUNTY','Year'],inplace=True)

trans\_df['Year'] = trans\_df.Year.apply(lambda x : str(x))

cols = trans\_df.columns.tolist()

trans\_df[cols[2]] = trans\_df[cols[2]].round(2)

trans\_df[cols[3]] = trans\_df[cols[3]].round(2)

pollutant\_col = "Yearly Avg "+ self.\_\_pollutant+ " Concentration"+"("+units+")"

aqi\_col = "Yearly Avg "+ self.\_\_pollutant+" AQI VALUE"

trans\_df = trans\_df.rename(columns={trans\_df.columns[2]:pollutant\_col,trans\_df.columns[3]:aqi\_col})

if self.typ == 'conc':

trans\_df.drop(trans\_df.columns[3],axis=1,inplace=True)

else:

trans\_df.drop(trans\_df.columns[2],axis=1,inplace=True)

return trans\_df

def \_\_getmap(self,trans\_df):

trans\_df = trans\_df

counties = trans\_df.COUNTY.unique()

# print(len(counties))

# cols = df\_x.columns

rows = ceil(len(counties)/4)

colus = 4

fig = make\_subplots(rows=rows, cols=colus,subplot\_titles=counties)

fig['layout'].update(height=3400, width=1800)

fig['layout'].update(title = self.\_\_pollutant+" Data Trend")

r = 1

c = 1

for county in counties:

cdf = trans\_df[trans\_df['COUNTY']==county]

# col = cdf.columns

fig.add\_trace(go.Scatter(x=cdf['Year'], y=cdf.iloc[:,2] ,name=county),row=r, col=c)

fig.update\_xaxes(title\_text = "Year")

fig.update\_yaxes(title\_text = "Pollutant Concentration "+"("+self.units+")")

c+=1

if c > 4:

r+=1

c=1

fig.show()

# return fig

def \_\_getmap\_analysis(self,trans\_df):

trans\_df = trans\_df

cols = trans\_df.columns.tolist()

dic= dict(trans\_df.groupby("COUNTY")[cols[2]].max())

lst =[]

for county in dic.keys():

val = trans\_df[(trans\_df["COUNTY"]==county) & (trans\_df[cols[2]]==dic[county])]["Year"].values

data = {"County": county,"Concentration":dic[county],"Year":val}

lst.append(data)

d = pd.DataFrame(lst)

d= d.explode(['Year'])

d\_n = pd.DataFrame(d['Year'].value\_counts()).reset\_index()

d\_n= d\_n.rename(columns = {"index":"Year","Year":"Count"})

f = px.bar(d\_n, x= "Year",y ="Count",text = "Count",text\_auto=True)

f['layout'].update(title = self.\_\_pollutant+" Maximum Pollution Years")

f.update\_yaxes(title\_text = "No. of Counties")

f.show()

def run(self):

trans\_data = self.\_transform\_data()

self.\_\_getmap(trans\_data)

self.\_\_getmap\_analysis(trans\_data)

return trans\_data

# In[54]:

df4 = Plot\_Map(df1,'conc').run()

# In[55]:

df5 = Plot\_Map(df2,'conc').run()

# In[56]:

df6 = Plot\_Map(df3,'conc').run()

# # 4. Regression Analysis

# - Regression Analysis (OLS method) is done to examine the relationship between Year and the pollutant concentration levels.

# - The relation between different years and pollutant concentration is examined for different counties. For each county, the regression line slope value,R squared value, P-value and the regression line equation is calculated. Further, the significance of each relationship is checked by analysing the calculated P-value at a 95% confidence level.

# In[57]:

from sklearn import linear\_model

import plotly.graph\_objects as go

import warnings

from scipy import stats

import numpy as np

from plotly.subplots import make\_subplots

warnings.filterwarnings("ignore")

class regression\_analysis(Plot\_Map):

def \_\_init\_\_(self,df,calc\_type):

self.\_\_pollutant = None

super().\_\_init\_\_(df,calc\_type)

def \_\_r\_trans\_df(self):

df = super().\_transform\_data()

cols = df.columns

self.\_\_pollutant = cols[2].split()[2]

df['Year'] = pd.to\_numeric(df['Year'])

return df

def \_\_reg\_analysis(self,initial\_df):

df = initial\_df

cols = df.columns

counties\_list = df.COUNTY.unique().tolist()

appended\_data =[]

for i,county in enumerate(counties\_list):

df2 = df[df['COUNTY']==county]

## Regression Analysis

X = df2['Year'].values.tolist()

y = df2[cols[2]].values.tolist()

X\_f = np.array(X, dtype=np.float32)

y\_f = np.array(y, dtype=np.float32)

slope, intercept, r\_value, p\_value, std\_err = stats.linregress(X\_f,y\_f)

line = str(round(slope,4)) + ' \* '+'Year '+ '+ ' + str(round(intercept,4))

sig=lambda p\_value: True if p\_value <= 0.05 else False

data = {'County':county,'Slope':slope,'R-Squared Value':r\_value\*\*2,'P-Value':p\_value,

'P-Value less than 0.05?':sig(p\_value),

'Line-Equation': line}

data\_df = pd.DataFrame(data,index = [i])

appended\_data.append(data\_df)

final\_df = pd.concat(appended\_data)

self.\_\_annotations = final\_df['Line-Equation'].values.tolist()

return final\_df

def \_\_getmap(self,trans\_df):

map\_df = trans\_df

map\_df['Year'] = pd.to\_numeric(map\_df['Year'])

cols = map\_df.columns.tolist()

fig = px.scatter(map\_df, x="Year", y=cols[2], color="COUNTY",trendline='ols',trendline\_color\_override='red')

fig.update\_layout(

title = self.\_\_pollutant+" Data " + "Regression Analysis",

updatemenus=[

{

"buttons": [

{

"label": m,

"method": "update",

"args": [

{

"visible": [

True if m == "All" else t.name == m for t in fig.data

]

}

],

}

for m in ["All"] + map\_df["COUNTY"].unique().tolist()

]

}

]

)

fig.show()

def run(self):

initial\_df = self.\_\_r\_trans\_df()

final\_df = self.\_\_reg\_analysis(initial\_df)

# f\_df = final\_df[['County','R-Squared\_value']]

# f\_df = f\_df.groupby(['County']).max()

# f\_df.reset\_index(inplace=True)

final\_df['Pollutant'] = self.\_\_pollutant

self.\_\_getmap(initial\_df)

return final\_df

# ## 4.1 Regression Analysis for Carbon Monoxide Pollutant

# In[58]:

df7 = regression\_analysis(df1,'conc').run()

df7

# ## 4.2 Regression Analysis for PM2.5 Pollutant

# In[59]:

df8 = regression\_analysis(df2,'conc').run()

df8

# ## 4.3 Regression Analysis for Ozone Pollutant

# In[60]:

df9 = regression\_analysis(df3,'conc').run()

df9

# # 5. Clustering The Counties

# - The Counties are now clustered based on their regression slope values

# - The counties are then further divided on the bases of their clusters into different pollution trends - Low, Medium, Increasing, Heavily Increasing

# In[61]:

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

from yellowbrick.cluster import KElbowVisualizer

class Clustering\_data:

def \_\_init\_\_(self,regression\_df,clusters = 4):

self.df = regression\_df

self.f\_clusters = clusters

def \_\_cluster\_number\_analysis(self):

self.df.sort\_values('Slope',ignore\_index=True,inplace=True)

x1 = np.array(self.df.index.values)

x2 = np.array(self.df['Slope'].values)

X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)

X=np.nan\_to\_num(X)

# distortions=[]

# for i in range(1, 11):

# km = KMeans(

# n\_clusters=i, init='k-means++',

# n\_init=10, max\_iter=300,

# tol=1e-04, random\_state=0

# )

# km.fit(X)

# distortions.append(km.inertia\_)

# # plot

# plt.plot(range(1, 11), distortions, marker='o')

# plt.xlabel('Number of clusters')

# plt.ylabel('Distortion')

# plt.title("Determining the number of clusters")

# plt.show()

model = KMeans()

visualizer = KElbowVisualizer(

model, k=(2,10))

visualizer.fit(X) # Fit the data to the visualizer

visualizer.poof()

return X

def \_\_assigning\_clusters(self,X):

km = KMeans(

n\_clusters=self.f\_clusters, init='k-means++',

n\_init=10, max\_iter=300,

tol=1e-04, random\_state=0)

y\_km = km.fit\_predict(X)

self.df['Cluster']=y\_km

self.df['group'] = self.df['Cluster'].ne(self.df['Cluster'].shift()).cumsum()

mapping = {1:'Low', 2:'Medium', 3:'Increasing',4:'Heavily Increasing'}

self.df['Pollution Trend']= self.df['group'].apply(lambda x : mapping[x])

self.df.drop('group',axis=1,inplace=True)

def \_\_plot\_clusters(self):

fig = px.scatter(self.df,x=self.df.index.values,y='Slope',color='Cluster',hover\_name='County',labels =

dict(x = "County Index", Slope = "Slope of Regression Line"))

title = 'Clustering for '+ self.df['Pollutant'].unique()[0]+' data'

fig.update\_layout(title=title)

fig.show()

def run(self):

X = self.\_\_cluster\_number\_analysis()

self.\_\_assigning\_clusters(X)

self.\_\_plot\_clusters()

final\_df = self.df

return final\_df

# ## 5.1 Clustering Counties for Carbon Monoxide Pollutant

# In[62]:

f1\_df = Clustering\_data(df7).run()

f1\_df

# ### 5.1.1 Checking for Outliers

# In[63]:

fig = px.box(f1\_df, y="Slope",hover\_name="County",title="CO Outliers")

fig.add\_annotation(x=0.05, y=0.18, #Q1

text="Alameda",

font=dict(size=12),

showarrow=False,

)

fig.add\_annotation(x=0.06, y=-0.32, #Q1

text="Los Angeles",

font=dict(size=12),

showarrow=False,

)

fig.show()

# - For Carbon Monoxide, it is quite evident that the counties Alamaeda and Los Angeles are the outliers. Alamaeda had an increasing trend for pollutant concentration over the years while the county of Los Angeles had a decreasing trend for pollutant concentration over the years.

# ## 5.2 Clustering Counties for PM2.5 Pollutant

# In[64]:

f2\_df=Clustering\_data(df8).run()

f2\_df

# ### 5.2.1 Checking for Outliers

# In[65]:

q1=f2\_df["Slope"].quantile(0.25)

q3=f2\_df["Slope"].quantile(0.75)

IQR=q3-q1

outliers = f2\_df[((f2\_df["Slope"]<(q1-1.5\*IQR)) | (f2\_df["Slope"]>(q3+1.5\*IQR)))]

fig = px.box(f2\_df, y="Slope",hover\_name="County",title="PM2.5 Outliers")

outliers\_annotations = outliers[["County","Slope"]].sort\_values(by ="Slope",ascending=False)

outliers\_lst = outliers\_annotations.values.tolist()

for county, slope in outliers\_lst:

fig.add\_annotation(x=0.05, y=slope, #Q1

text=county,

font=dict(size=10),

showarrow=False,

)

fig['layout'].update(height = 800,width =800)

fig.show()

# - From the outlier analysis of PM2.5 pollutant concentrations, it is evident that the counties - Riverside and San Bernardino are the extreme outliers

# ## 5.3 Clustering Counties for Ozone Pollutant

# In[66]:

f3\_df = Clustering\_data(df9).run()

f3\_df

# ### 5.3.1 Checking for Outliers

# In[67]:

q1=f3\_df["Slope"].quantile(0.25)

q3=f3\_df["Slope"].quantile(0.75)

IQR=q3-q1

outliers = f3\_df[((f3\_df["Slope"]<(q1-1.5\*IQR)) | (f3\_df["Slope"]>(q3+1.5\*IQR)))]

fig = px.box(f3\_df, y="Slope",hover\_name="County",title = "Ozone Outliers")

outliers\_annotations = outliers[["County","Slope"]].sort\_values(by ="Slope",ascending=False)

outliers\_lst = outliers\_annotations.values.tolist()

for county, slope in outliers\_lst:

fig.add\_annotation(x=0.06, y=slope, #Q1

text=county,

font=dict(size=10),

showarrow=False,

)

fig['layout'].update(height = 800,width =800)

fig.show()

# - From the outlier analysis of Ozone pollutant concentrations, it is evident that the counties - Santa Barbara and Inyo are the extreme outliers

# ## 5.4 Analysing Pollution Trends

# - Analysing which Counties had the Pollution Trend as <b>Heavily Increasing</b> for all three pollutants:

# In[68]:

##Getting counties with Heavily Increasing pollution trend for all 3 pollutants

co\_counties\_high = set(f1\_df[f1\_df['Pollution Trend']=='Heavily Increasing']['County'].values.tolist())

pm\_counties\_high = set(f2\_df[f2\_df['Pollution Trend']=='Heavily Increasing']['County'].values.tolist())

ozone\_counties\_high = set(f3\_df[f3\_df['Pollution Trend']=='Heavily Increasing']['County'].values.tolist())

common\_counties\_high = co\_counties\_high.intersection(pm\_counties\_high,ozone\_counties\_high)

common\_counties\_high

# - Analysing which Counties had the Pollution Trend as <b>Low</b> for all three pollutants:

# In[69]:

##Getting counties with low pollution trend for all 3 pollutants

co\_counties\_low = set(f1\_df[f1\_df['Pollution Trend']=='Low']['County'].values.tolist())

pm\_counties\_low = set(f2\_df[f2\_df['Pollution Trend']=='Low']['County'].values.tolist())

ozone\_counties\_low = set(f3\_df[f3\_df['Pollution Trend']=='Low']['County'].values.tolist())

common\_counties\_low = co\_counties\_low.intersection(pm\_counties\_low,ozone\_counties\_low)

common\_counties\_low

# In[ ]: