Air Quality Project

September 26, 2021

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import cufflinks as cf
import warnings
import itertools
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
plt.style.use('fivethirtyeight')
import statsmodels.api as sm
import matplotlib
matplotlib.rcParams['axes.labelsize'] = 14
matplotlib.rcParams['xtick.labelsize'] = 12
matplotlib.rcParams['ytick.labelsize'] = 12
matplotlib.rcParams['text.color'] = 'k'


data1=pd.read_csv("Data_2017.csv")
data2=pd.read_csv("Data_2018.csv")
data3=pd.read_csv("Data_2019.csv")
data4=pd.read_csv("Data_2020.csv")
data5=pd.read_csv("Data_PM_2017.csv")
data6=pd.read_csv("Data_PM_2018.csv")
data7=pd.read_csv("Data_PM_2019.csv")
p_id1=data1['pollutant_id'].to_list()
date1=data1['date_time'].to_list()
p_value1=data1['pollutant_value'].to_list()
s_code1=data1['station_code'].to_list()

p_id2=data2['pollutant_id'].to_list()
date2=data2['date_time'].to_list()
p_value2=data2['pollutant_value'].to_list()
s_code2=data2['station_code'].to_list()


p_id3=data3['pollutant_id'].to_list()
date3=data3['date_time'].to_list()
```

```
p_value3=data3['pollutant_value'].to_list()
s_code3=data3['station_code'].to_list()

p_id4=data4['pollutant_id'].to_list()
date4=data4['date_time'].to_list()
p_value4=data4['pollutant_value'].to_list()


p_id=p_id1+p_id2+p_id3
date=date1+date2+date3
p_value=p_value1+p_value2+p_value3
s_code=s_code1+s_code2+s_code3

#Particulate matter
p_id5=data5['Pollutant_Id'].to_list()
#date5=data5['date_time'].to_list()
p_value5=data5['Pollutant_Value'].to_list()
s_code5=data5['Station_code'].to_list()

p_id6=data6['pollutant_id'].to_list()
#date6=data6['date_time'].to_list()
p_value6=data6['pollutant_value'].to_list()
s_code6=data6['station_code'].to_list()


p_id7=data7['pollutant_id'].to_list()
#date7=data7['date_time'].to_list()
p_value7=data7['pollutant_value'].to_list()
s_code7=data7['station_code'].to_list()

P_id=p_id5+p_id6+p_id7
P_value=p_value5+p_value6+p_value7
S_code=s_code5+s_code6+s_code7
```

NICOSIA TRAFFIC

[2]:
```
date_NT=[date[i] for i in range(len(p_id)) if p_id[i]==6 and s_code[i]==1]
NO_2=[p_value[i] for i in range(len(p_id)) if p_id[i]==2 and  s_code[i]==1]
SO_2=[p_value[i] for i in range(len(p_id)) if(p_id[i]==4 and s_code[i]==1)]
O_3=[p_value[i] for i in range(len(p_id)) if(p_id[i]==5 and s_code[i]==1)]
CO=[p_value[i] for i in range(len(p_id)) if(p_id[i]==6 and s_code[i]==1)]
for i in range(len(CO)):
    if (CO[i]=='BDL'):
        CO[i]='nan'
    elif NO_2[i]=='BDL':
        NO_2[i]='nan'
    elif SO_2[i]=='BDL':
```

```
        SO_2[i]='nan'
    elif O_3[i]=="BDL":
        O_3[i]='nan'
    else:
        pass

for i in range(len(CO)):
    if (CO[i]=='nan'):
        CO[i]='0'
    elif SO_2[i]=='nan':
        SO_2[i]='0'
    elif NO_2[i]=='nan':
        NO_2[i]='0'
    elif O_3[i]=='nan':
        O_3[i]='0'
    else:
        pass

for i in range(len(SO_2)):
    if SO_2[i]=='BDL':
        SO_2[i]='nan'
for i in range(len(O_3)):
    if O_3[i]=='BDL':
        O_3[i]='nan'
for i in range(len(SO_2)):
    if (SO_2[i]=='nan'):
        SO_2[i]='0'
for i in range(len(O_3)):
    if (O_3[i]=='nan'):
        O_3[i]='0'


data_NT=[[date_NT[i],NO_2[i],SO_2[i],O_3[i], CO[i]] for i in␣
 ↪range(len(date_NT))]
data_NT= pd.DataFrame(data_NT, columns = ['Date','NO_2','SO_2','O_3', 'CO'])
data_NT["CO"] = pd.to_numeric(data_NT["CO"], downcast="float")
data_NT["NO_2"] = pd.to_numeric(data_NT["NO_2"], downcast="float")
data_NT["SO_2"] = pd.to_numeric(data_NT["SO_2"], downcast="float")
data_NT["O_3"] = pd.to_numeric(data_NT["O_3"], downcast="float")

column_mean1 = data_NT['CO'].mean()
column_mean2 = data_NT['NO_2'].mean()
column_mean3 = data_NT['SO_2'].mean()
column_mean4 = data_NT['O_3'].mean()

data_NT['CO'].fillna(column_mean1)
data_NT['NO_2'].fillna(column_mean2)
```

```
data_NT['SO_2'].fillna(column_mean3)
data_NT['O_3'].fillna(column_mean4)

data_NT['Date'] = pd.to_datetime(data_NT['Date'])
data_NT=data_NT.set_index('Date')
data_NT_O=data_NT

data_NT=data_NT.resample('D').mean()
#Nicosia Traffic
PM_2_5_NT=[P_value[i] for i in range(len(P_id)) if(P_id[i]==26 and␣
 ↪S_code[i]==1)]
PM_10_NT=[P_value[i] for i in range(len(P_id)) if(P_id[i]==25 and S_code[i]==1)]
data_NT['PM_2.5']=PM_2_5_NT
data_NT['PM_10']=PM_10_NT
column_mean9 = data_NT['PM_2.5'].mean()
column_mean10 = data_NT['PM_10'].mean()

data_NT['PM_2.5'].fillna(column_mean9)
data_NT['PM_10'].fillna(column_mean10)
data_NT.head()
```

[2]:

Date	NO_2	SO_2	O_3	CO	PM_2.5	PM_10
2017-01-01	22.656250	1.409583	31.888332	632.104187	17.4	30.5
2017-01-02	53.323479	1.470435	34.365417	888.864807	16.1	21.1
2017-01-03	40.632275	2.226364	46.317081	535.313354	30.4	80.7
2017-01-04	20.976250	0.873750	77.764168	315.857483	35.1	86.3
2017-01-05	18.100000	3.159167	89.449165	270.007507	28.7	70.2

[3]:
```
data_NT.describe()
```

[3]:

	NO_2	SO_2	O_3	CO	PM_2.5	PM_10
count	1091.000000	1093.000000	1094.000000	1092.000000	1020.000000	1067.000000
mean	30.552916	2.724416	56.590080	503.460266	17.622353	43.386223
std	12.430579	1.524104	23.430660	236.155304	6.828044	28.948460
min	4.437500	0.111667	11.179167	161.383179	3.000000	10.600000
25%	20.973541	1.520833	35.293701	337.561981	13.000000	30.000000
50%	29.250000	2.521667	59.215401	418.735962	16.500000	
75%	39.214584	3.591667	76.068756	611.567383	20.800000	
max	75.320831	9.293750	110.616669	2130.129150	62.100000	

```
50%      37.500000
75%      47.100000
max     569.700000
```

1 Variance Inflator factor(VIF)

A value of 1 indicates there is no correlation between a given explanatory variable and any other explanatory variables in the model.A value between 1 and 5 indicates moderate correlation between a given explanatory variable and other explanatory variables in the model, but this is often not severe enough to require attention.A value greater than 5 indicates potentially severe correlation between a given explanatory variable and other explanatory variables in the model. In this case, the coefficient estimates and p-values in the regression output are likely unreliable.

[4]:
```
from patsy import dmatrices
from statsmodels.stats.outliers_influence import variance_inflation_factor
y, X = dmatrices('NO_2 ~ SO_2+O_3+CO', data=data_NT, return_type='dataframe')
#calculate VIF for each explanatory variable
vif = pd.DataFrame()
vif['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
vif['variable'] = X.columns
vif
```

[4]:

	VIF	variable
0	44.429354	Intercept
1	1.180361	SO_2
2	2.324447	O_3
3	2.481323	CO

1.1 NICOSIA RESIDENCE

[5]:
```
date_NR=[date[i] for i in range(len(p_id)) if p_id[i]==6 and s_code[i]==2]
NO_2=[p_value[i] for i in range(len(p_id)) if p_id[i]==2 and  s_code[i]==2]
SO_2=[p_value[i] for i in range(len(p_id)) if(p_id[i]==4 and s_code[i]==2)]
O_3=[p_value[i] for i in range(len(p_id)) if(p_id[i]==5 and s_code[i]==2)]
CO=[p_value[i] for i in range(len(p_id)) if(p_id[i]==6 and s_code[i]==2)]
for i in range(len(CO)):
    if (CO[i]=='BDL'):
        CO[i]='nan'
    elif NO_2[i]=='BDL':
        NO_2[i]='nan'
    elif SO_2[i]=='BDL':
        SO_2[i]='nan'
    elif O_3[i]=="BDL":
        O_3[i]='nan'
```

```
    else:
        pass

for i in range(len(CO)):
    if (CO[i]=='nan'):
        CO[i]='0'
    elif SO_2[i]=='nan':
        SO_2[i]='0'
    elif NO_2[i]=='nan':
        NO_2[i]='0'
    elif O_3[i]=='nan':
        O_3[i]='0'
    else:
        pass

for i in range(len(SO_2)):
    if SO_2[i]=='BDL':
        SO_2[i]='nan'
for i in range(len(O_3)):
    if O_3[i]=='BDL':
        O_3[i]='nan'
for i in range(len(SO_2)):
    if (SO_2[i]=='nan'):
        SO_2[i]='0'
for i in range(len(O_3)):
    if (O_3[i]=='nan'):
        O_3[i]='0'


data_NR=[[date_NR[i],NO_2[i],SO_2[i],O_3[i], CO[i]] for i in␣
 ↪range(len(date_NR))]
data_NR= pd.DataFrame(data_NR, columns = ['Date','NO_2','SO_2','O_3', 'CO'])
data_NR["CO"] = pd.to_numeric(data_NR["CO"], downcast="float")
data_NR["NO_2"] = pd.to_numeric(data_NR["NO_2"], downcast="float")
data_NR["SO_2"] = pd.to_numeric(data_NR["SO_2"], downcast="float")
data_NR["O_3"] = pd.to_numeric(data_NR["O_3"], downcast="float")

column_mean5 = data_NR['CO'].mean()
column_mean6 = data_NR['NO_2'].mean()
column_mean7 = data_NR['SO_2'].mean()
column_mean8 = data_NR['O_3'].mean()

data_NR['CO'].fillna(column_mean5)
data_NR['NO_2'].fillna(column_mean6)
data_NR['SO_2'].fillna(column_mean7)
data_NR['O_3'].fillna(column_mean8)
```

```
data_NR['Date'] = pd.to_datetime(data_NR['Date'])
data_NR=data_NR.set_index('Date')
data_NR_O=data_NR
data_NR=data_NR.resample('D').mean()
PM_2_5_NR=[P_value[i] for i in range(len(P_id)) if(P_id[i]==26 and␣
 ↪S_code[i]==2)]
PM_10_NR=[P_value[i] for i in range(len(P_id)) if(P_id[i]==25 and S_code[i]==2)]
data_NR['PM_2.5']=PM_2_5_NR
data_NR['PM_10']=PM_10_NR
column_mean11 = data_NR['PM_2.5'].mean()
column_mean12 = data_NR['PM_10'].mean()

data_NR['PM_2.5'].fillna(column_mean11)
data_NR['PM_10'].fillna(column_mean12)

data_NR.head()
```

[5]:

Date	NO_2	SO_2	O_3	CO	PM_2.5	PM_10
2017-01-01	22.525000	2.425000	35.771667	553.710815	16.3	26.7
2017-01-02	31.235834	3.890417	43.341667	629.263916	16.3	20.9
2017-01-03	22.701668	3.487917	56.488750	304.118683	17.9	32.1
2017-01-04	17.751667	2.367500	78.111252	299.715424	26.2	48.6
2017-01-05	11.007916	2.674167	101.402084	150.582916	24.6	44.5

[6]:
```
data_NR.describe()
```

[6]:

	NO_2	SO_2	O_3	CO	PM_2.5	PM_10
count	1091.000000	1093.000000	1094.000000	1090.000000	1022.000000	1032.000000
mean	21.873568	1.978501	68.560783	308.482452	14.992857	35.353973
std	11.902061	1.115598	23.170956	160.722107	6.087932	28.894236
min	4.245833	0.000000	15.056521	17.467501	1.700000	9.600000
25%	12.418188	1.220833	49.006149	198.868439	11.100000	24.800000
50%	18.887501	1.803333	69.943756	263.924561	14.200000	29.900000
75%	29.322292	2.504167	87.273094	377.548126	17.775000	36.900000
max	65.085716	7.175000	129.175003	1169.722534	78.100000	624.200000

1.2 EMEP-Ayia Marina -Background

[7]:
```
date_Z=[date[i] for i in range(len(p_id)) if p_id[i]==6 and s_code[i]==9]
NO_2=[p_value[i] for i in range(len(p_id)) if p_id[i]==2 and  s_code[i]==8]
SO_2=[p_value[i] for i in range(len(p_id)) if(p_id[i]==4 and s_code[i]==9)]
O_3=[p_value[i] for i in range(len(p_id)) if(p_id[i]==5 and s_code[i]==9)]
CO=[p_value[i] for i in range(len(p_id)) if(p_id[i]==6 and s_code[i]==9)]
for i in range(len(CO)):
    if (CO[i]=='BDL'):
        CO[i]='nan'
    elif NO_2[i]=='BDL':
        NO_2[i]='nan'
    elif SO_2[i]=='BDL':
        SO_2[i]='nan'
    elif O_3[i]=="BDL":
        O_3[i]='nan'
    else:
        pass

for i in range(len(CO)):
    if (CO[i]=='nan'):
        CO[i]='0'
    elif SO_2[i]=='nan':
        SO_2[i]='0'
    elif NO_2[i]=='nan':
        NO_2[i]='0'
    elif O_3[i]=='nan':
        O_3[i]='0'
    else:
        pass

for i in range(len(SO_2)):
    if SO_2[i]=='BDL':
        SO_2[i]='nan'
for i in range(len(O_3)):
    if O_3[i]=='BDL':
        O_3[i]='nan'
for i in range(len(SO_2)):
    if (SO_2[i]=='nan'):
        SO_2[i]='0'
for i in range(len(O_3)):
    if (O_3[i]=='nan'):
        O_3[i]='0'


data_Z=[[date_Z[i],NO_2[i],SO_2[i],O_3[i], CO[i]] for i in range(len(date_Z))]
data_Z= pd.DataFrame(data_Z, columns = ['Date','NO_2','SO_2','O_3', 'CO'])
```

```
data_Z["CO"] = pd.to_numeric(data_Z["CO"], downcast="float")
data_Z["NO_2"] = pd.to_numeric(data_Z["NO_2"], downcast="float")
data_Z["SO_2"] = pd.to_numeric(data_Z["SO_2"], downcast="float")
data_Z["O_3"] = pd.to_numeric(data_Z["O_3"], downcast="float")

column_mean13 = data_Z['CO'].mean()
column_mean14 = data_Z['NO_2'].mean()
column_mean15 = data_Z['SO_2'].mean()
column_mean16 = data_Z['O_3'].mean()

data_Z['CO'].fillna(column_mean13)
data_Z['NO_2'].fillna(column_mean14)
data_Z['SO_2'].fillna(column_mean15)
data_Z['O_3'].fillna(column_mean16)
data_Z['Date'] = pd.to_datetime(data_Z['Date'])
data_Z=data_Z.set_index('Date')
data_Z_O=data_Z
data_Z=data_Z.resample('D').mean()
PM_2_5_Z=[P_value[i] for i in range(len(P_id)) if(P_id[i]==26 and S_code[i]==9)]
PM_10_Z=[P_value[i] for i in range(len(P_id)) if(P_id[i]==25 and S_code[i]==9)]
PM_10_Z[0]=6.9
data_Z['PM_2.5']=PM_2_5_Z
data_Z['PM_10']=PM_10_Z
column_mean17 = data_Z['PM_2.5'].mean()
column_mean18 = data_Z['PM_10'].mean()
PM_10_Z[0]=column_mean18
#data_Z['PM_2.5'].fillna(column_mean17)
#data_Z['PM_10'].fillna(column_mean18)

data_Z=data_Z.interpolate()
data_AM=data_Z
data_AM.head()
```

[7]:

Date	NO_2	SO_2	O_3	CO	PM_2.5	PM_10
2017-01-01	3.523750	0.750870	72.954094	177.931366	2.5	6.9
2017-01-02	7.679584	1.183333	84.351250	200.942917	2.7	5.8
2017-01-03	6.755455	2.286250	93.035004	118.492271	4.0	6.3
2017-01-04	6.279167	0.467083	92.485832	162.221664	3.3	5.1
2017-01-05	10.433333	1.863750	117.720001	157.161255	2.8	6.7

[8]:
```
data_AM.describe()
```

[8]:

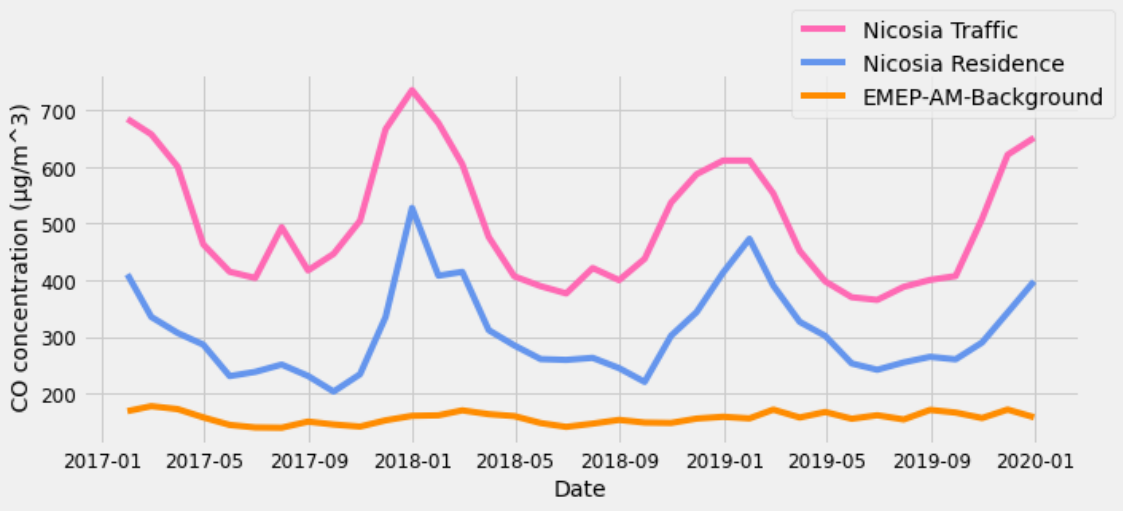
	NO_2	SO_2	O_3	CO	PM_2.5	\
count	1095.000000	1095.000000	1095.000000	1095.000000	1095.000000	
mean	8.420279	1.170504	96.236687	157.305588	10.705205	
std	2.952318	0.962954	14.476766	28.529747	5.949673	

```
min       2.715000     0.000000    60.569565    82.362503     0.500000
25%       6.381042     0.563333    85.797916   138.268333     6.875000
50%       7.892917     0.966667    95.721115   153.547058     9.900000
75%       9.839693     1.435000   106.540264   170.518753    13.400000
max      26.083334     9.247827   144.033340   346.175018    71.000000

              PM_10
count  1095.000000
mean     21.843744
std      19.841889
min       3.000000
25%      12.700000
50%      17.800000
75%      24.090000
max     268.900000
```

1.3 Data visualization and Analytics

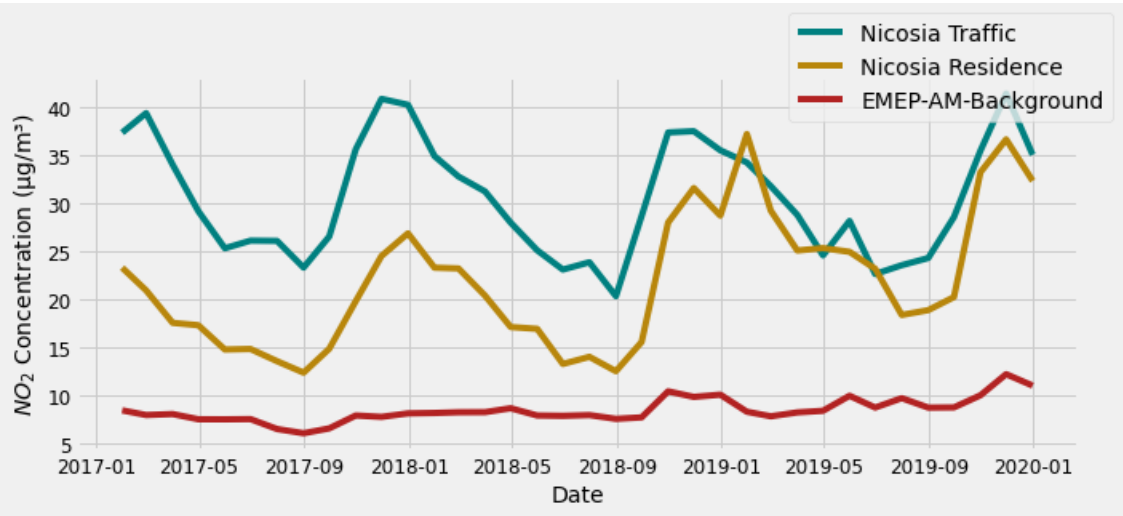
[9]:
```
import matplotlib.pyplot as plt
import seaborn as sns
data_NT_by_month=data_NT.resample('M').mean()
data_NR_by_month=data_NR.resample('M').mean()
data_AM_by_month=data_AM.resample('M').mean()
fig= plt.figure(figsize=(10,4))
#colors =␣
↪['lightpink','pink','fuchsia','mistyrose','hotpink','deeppink','magenta']
sns.lineplot(x=data_NT_by_month.
↪index,y='CO',data=data_NT_by_month,color='hotpink')
sns.lineplot(x=data_NR_by_month.
↪index,y='CO',data=data_NR_by_month,color='cornflowerblue')
sns.lineplot(x=data_AM_by_month.
↪index,y='CO',data=data_AM_by_month,color='darkorange')
plt.legend(['Nicosia Traffic', 'Nicosia Residence', 'EMEP-AM-Background'], loc␣
↪= 2, bbox_to_anchor = (0.7,1.2))
plt.ylabel(' CO concentration ( g/m^3)')
```

[9]:
```
Text(0, 0.5, ' CO concentration ( g/m^3)')
```

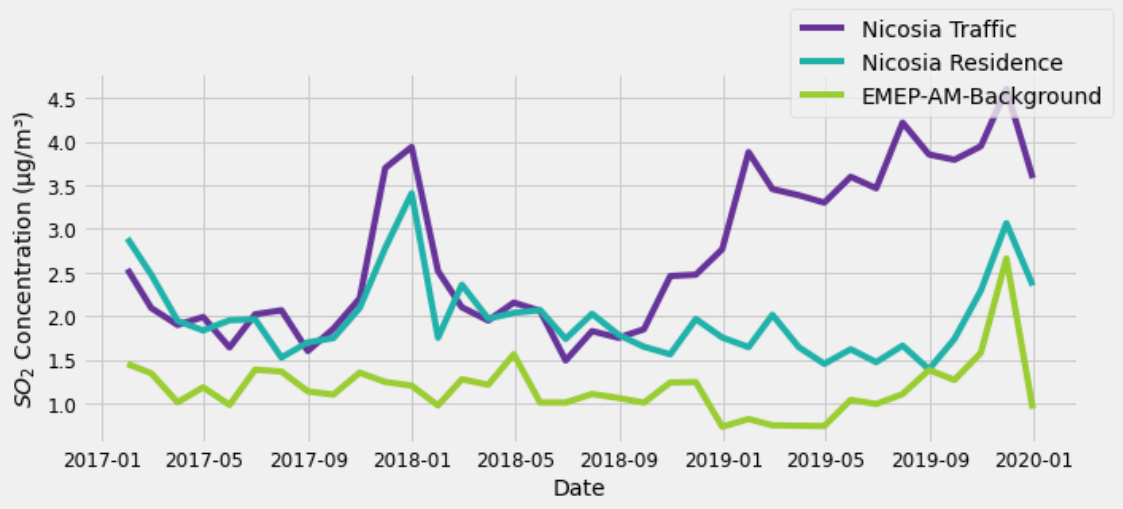


[10]:
```
fig= plt.figure(figsize=(10,4))
#colors =␣
 ↪['lightpink','pink','fuchsia','mistyrose','hotpink','deeppink','magenta']
sns.lineplot(x=data_NT_by_month.
 ↪index,y='NO_2',data=data_NT_by_month,color='teal')
sns.lineplot(x=data_NR_by_month.
 ↪index,y='NO_2',data=data_NR_by_month,color='darkgoldenrod')
sns.lineplot(x=data_AM_by_month.
 ↪index,y='NO_2',data=data_AM_by_month,color='firebrick')
plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc =␣
 ↪2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( r'$NO_{2}$'  + ' Concentration (µg/m³)')
```

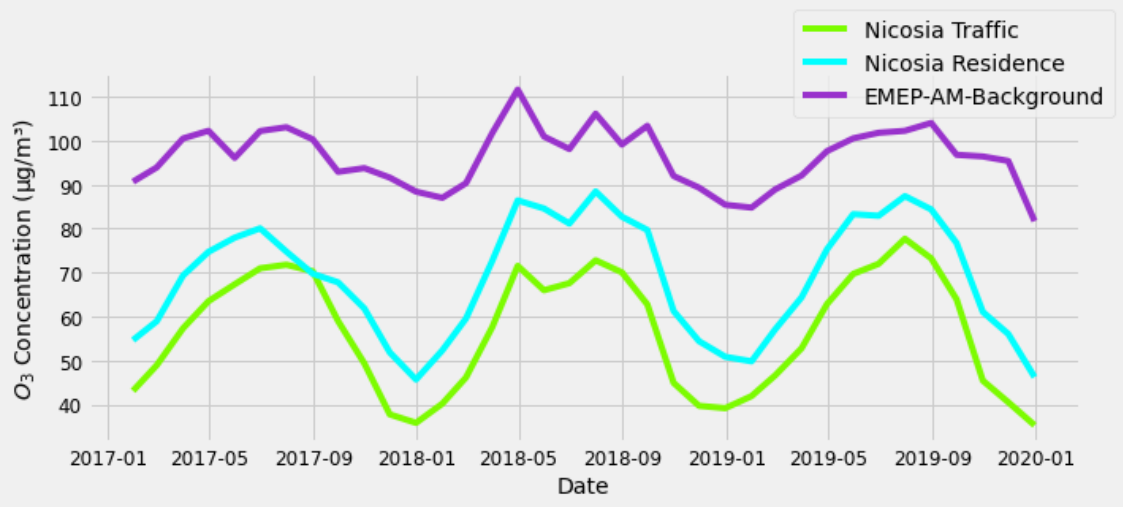
[10]: Text(0, 0.5, 'NO_{2} Concentration (µg/m³)')



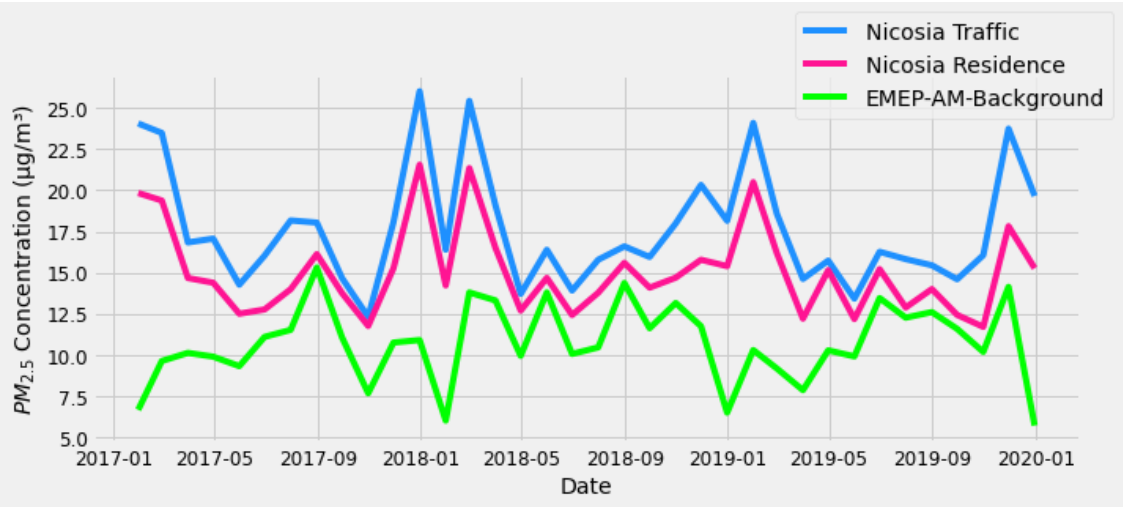
[11]:
```
fig= plt.figure(figsize=(10,4))
#colors =␣
 ↪['lightpink','pink','fuchsia','mistyrose','hotpink','deeppink','magenta']
sns.lineplot(x=data_NT_by_month.
 ↪index,y='SO_2',data=data_NT_by_month,color='rebeccapurple')
sns.lineplot(x=data_NR_by_month.
 ↪index,y='SO_2',data=data_NR_by_month,color='lightseagreen')
sns.lineplot(x=data_AM_by_month.
 ↪index,y='SO_2',data=data_AM_by_month,color='yellowgreen')
plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'],loc =␣
 ↪2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( r'$SO_{2}$'  + ' Concentration (µg/m³)')
plt.show()
```



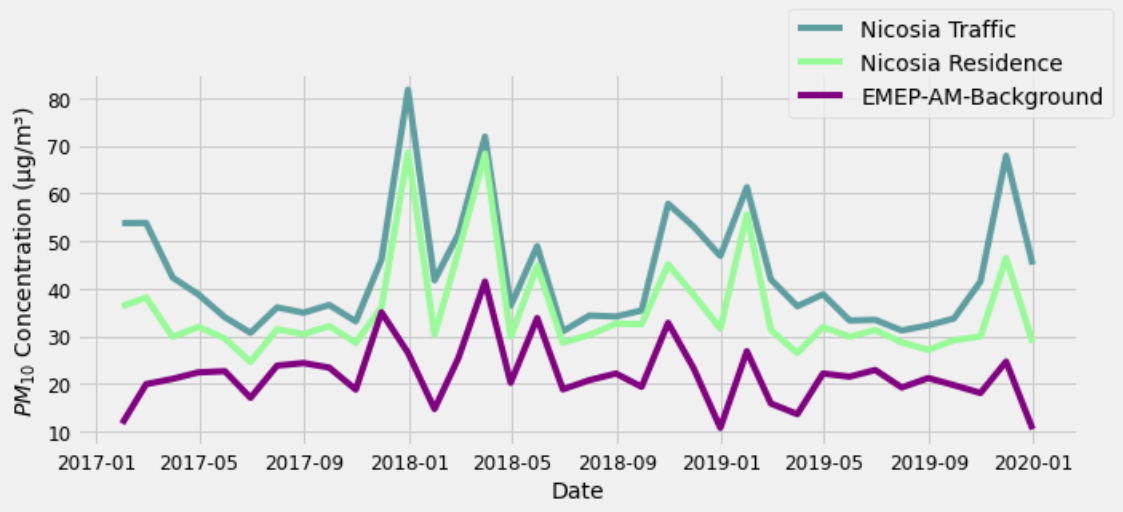
[12]:
```
fig= plt.figure(figsize=(10,4))
#colors =␣
 ↪['lightpink','pink','fuchsia','mistyrose','hotpink','deeppink','magenta']
sns.lineplot(x=data_NT_by_month.
 ↪index,y='O_3',data=data_NT_by_month,color='lawngreen')
sns.lineplot(x=data_NR_by_month.
 ↪index,y='O_3',data=data_NR_by_month,color='cyan')
sns.lineplot(x=data_AM_by_month.
 ↪index,y='O_3',data=data_AM_by_month,color='darkorchid')
plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc =␣
 ↪2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( r'$O_{3}$'  + ' Concentration (µg/m³)')
plt.show()
```



[13]:
```
fig= plt.figure(figsize=(10,4))
#colors =␣
 ↪['lightpink','pink','fuchsia','mistyrose','hotpink','deeppink','magenta']
sns.lineplot(x=data_NT_by_month.index,y='PM_2.
 ↪5',data=data_NT_by_month,color='dodgerblue')
sns.lineplot(x=data_NR_by_month.index,y='PM_2.
 ↪5',data=data_NR_by_month,color='deeppink')
sns.lineplot(x=data_AM_by_month.index,y='PM_2.
 ↪5',data=data_AM_by_month,color='lime')
plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc =␣
 ↪2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( r'$PM_{2.5}$'  + ' Concentration (µg/m³)')
plt.show()
```



[14]:
```
fig= plt.figure(figsize=(10,4))
#colors =␣
 ↪['lightpink','pink','fuchsia','mistyrose','hotpink','deeppink','magenta']
sns.lineplot(x=data_NT_by_month.
 ↪index,y='PM_10',data=data_NT_by_month,color='cadetblue')
sns.lineplot(x=data_NR_by_month.
 ↪index,y='PM_10',data=data_NR_by_month,color='palegreen')
sns.lineplot(x=data_AM_by_month.
 ↪index,y='PM_10',data=data_AM_by_month,color='purple')
plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc =␣
 ↪2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( r'$PM_{10}$'  + ' Concentration (µg/m³)')
plt.show()
```



[15]:
```
data_NT['Year']=data_NT.index.year
data_NT['Month']=data_NT.index.month
data_NT['Day_of_week']=data_NT.index.dayofweek
data_NT['Day']=data_NT.index.day
data_NT['Hour']=data_NT.index.hour
data_NT['Year']=data_NT.index.year
data_NT_O['Hour']=data_NT_O.index.hour
```

[16]:
```
data_NR['Year']=data_NT.index.year
data_NR['Month']=data_NR.index.month
data_NR['Day_of_week']=data_NR.index.dayofweek
data_NR['Day']=data_NR.index.day
data_NR['Hour']=data_NR.index.hour
data_NR['Year']=data_NR.index.year
```

```
data_NR_O['Hour']=data_NR_O.index.hour
```

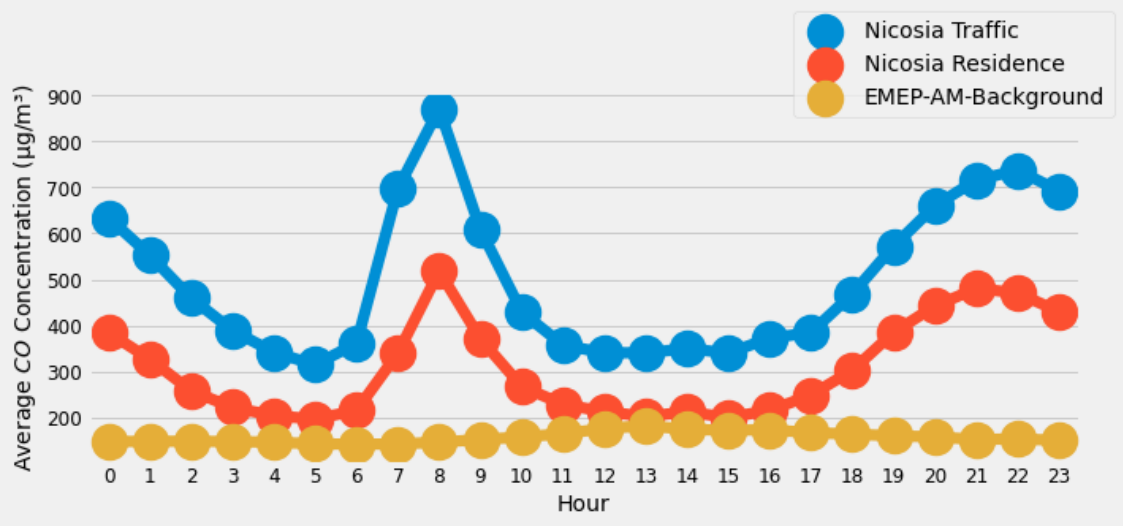
[17]:
```
data_AM['Year']=data_NT.index.year
data_AM['Month']=data_AM.index.month
data_AM['Day_of_week']=data_AM.index.dayofweek
data_AM['Day']=data_AM.index.day
data_AM['Hour']=data_AM.index.hour
data_AM['Year']=data_AM.index.year
data_Z_O['Hour']=data_Z_O.index.hour
```

[18]:
```
frames = [data_NT, data_NR, data_AM]
result = pd.concat(frames)
A=['Nicosia Traffic' for i in range(1095)]
B=['Nicosia Residence' for i in range(1095)]
C=['EMEP-AM-Background' for i in range(1095)]
D=A+B+C
result['Station']=D
#result.head()
```

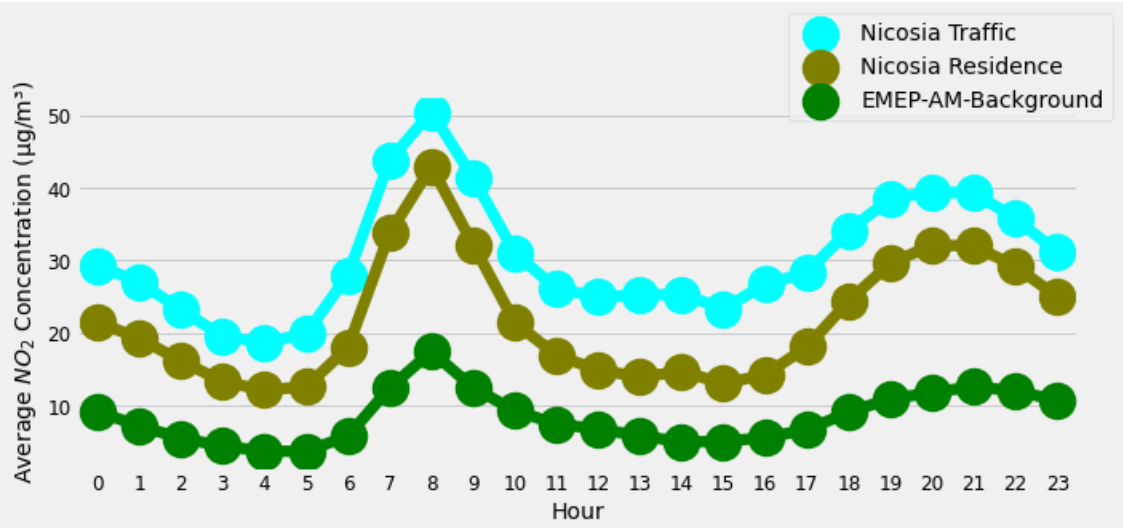
[19]:
```
frame = [data_NT_O, data_NR_O, data_Z_O]
original = pd.concat(frame)
E=['Nicosia Traffic' for i in range(26280)]
F=['Nicosia Residence' for i in range(26280)]
G=['EMEP-AM-Background' for i in range(26280)]
H=E+F+G
original['Station']=H
#original.head()
```

1.4 Hourly Visualization

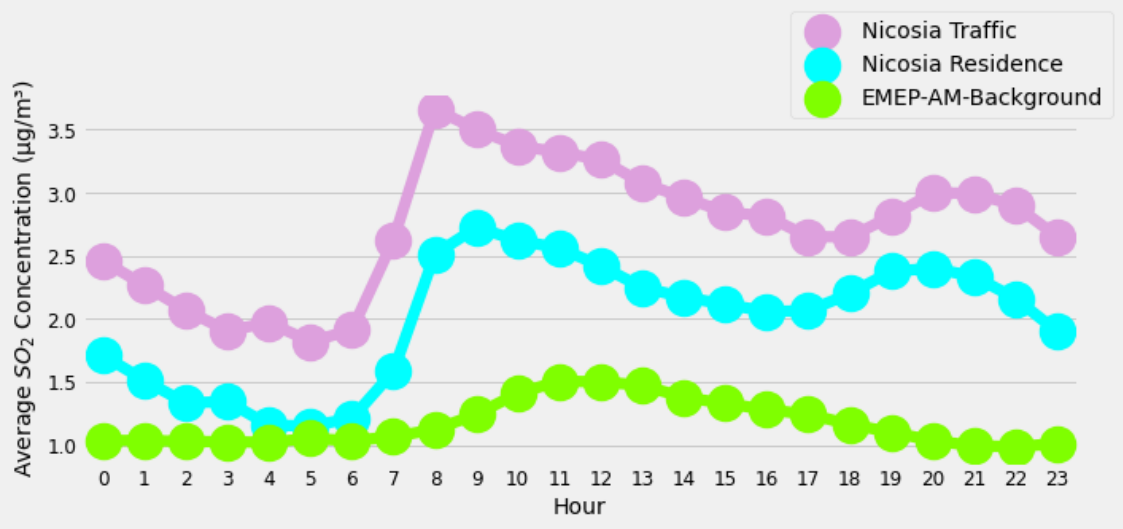
[20]:
```
fig= plt.figure(figsize=(10,4))
sns.pointplot(x="Hour", y="CO", hue="Station", data=original,
              ci=None)
#plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc␣
↪= 2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$CO_{}$'  + ' Concentration (µg/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()
```



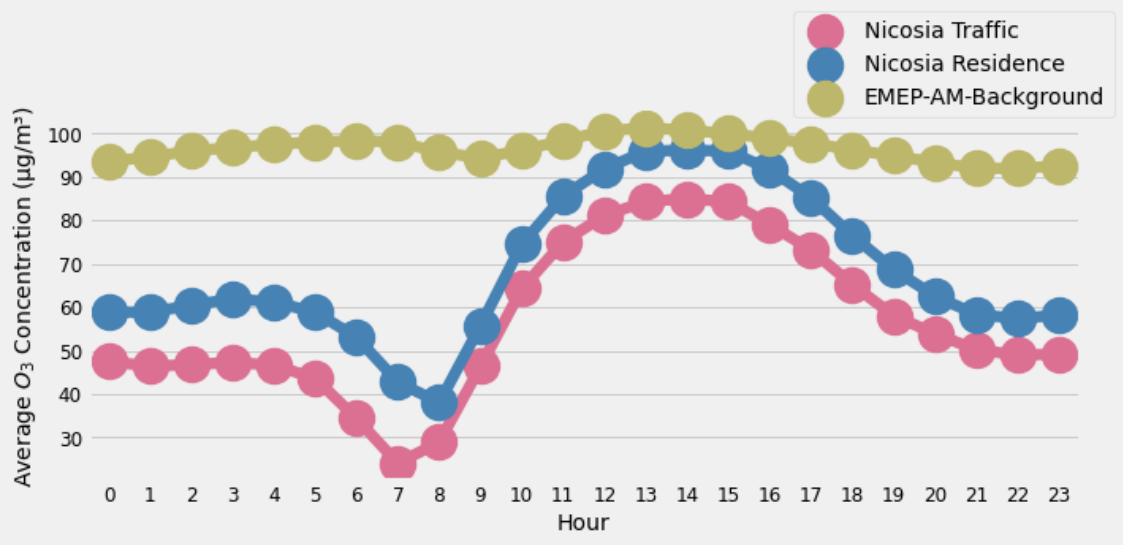
[21]:
```
fig= plt.figure(figsize=(10,4))
sns.pointplot(x="Hour", y="NO_2", hue="Station",␣
 ↪data=original,palette=['cyan','olive','green'],
              ci=None)
#plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc␣
 ↪= 2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$NO_{2}$'  + ' Concentration (µg/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()
```



[22]:
```
fig= plt.figure(figsize=(10,4))
sns.pointplot(x="Hour", y="SO_2", hue="Station",␣
 ↪data=original,palette=['plum','aqua','chartreuse'],
              ci=None)
#plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc␣
 ↪= 2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$SO_{2}$'  + ' Concentration (µg/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()
```

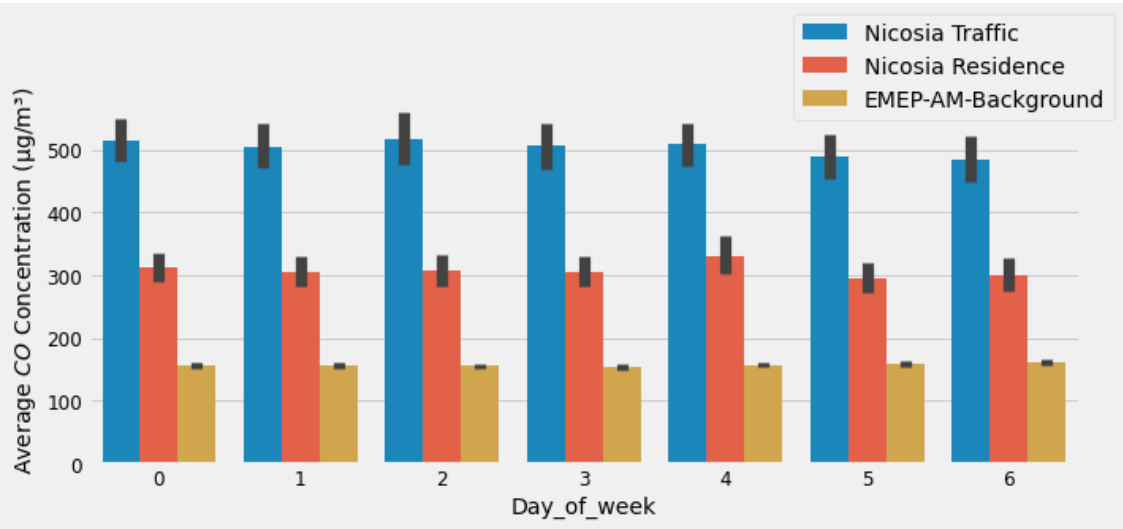


[23]:
```
fig= plt.figure(figsize=(10,4))
sns.pointplot(x="Hour", y="O_3", hue="Station",␣
 ↪data=original,palette=['palevioletred','steelblue','darkkhaki'],
              ci=None)
#plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc␣
 ↪= 2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$O_{3}$'  + ' Concentration (µg/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.3))
plt.show()
```

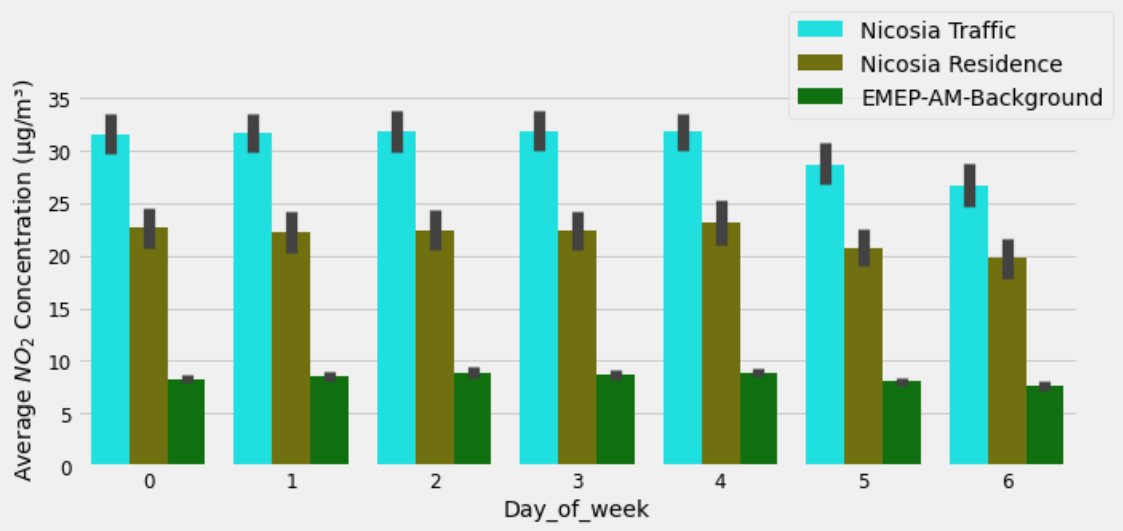


1.5 Day of Week visualization

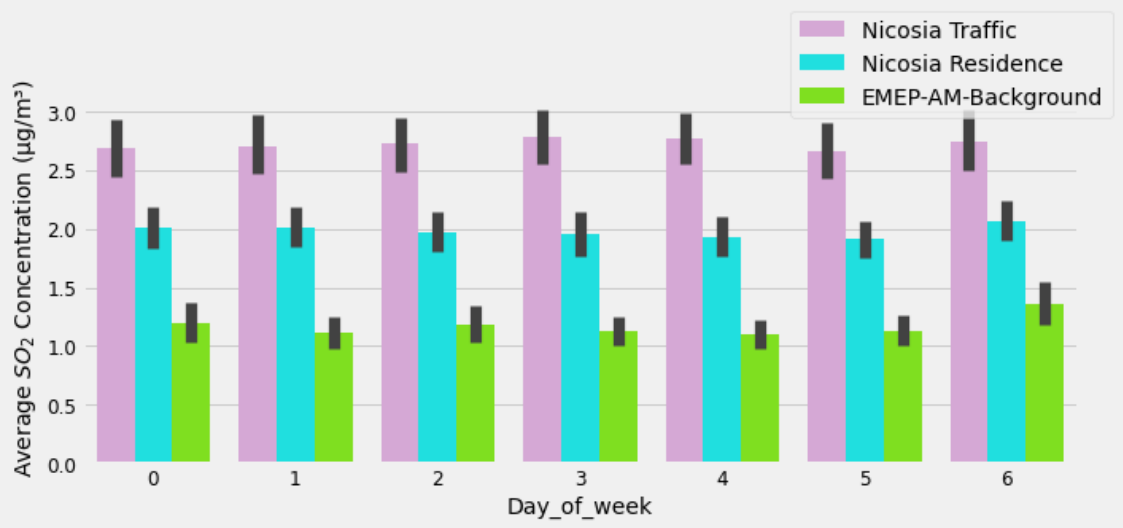
[24]:
```
fig= plt.figure(figsize=(10,4))
sns.barplot(x="Day_of_week", y="CO", hue="Station", data=result)
#plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc␣
↪= 2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$CO_{}$'  + ' Concentration (µg/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()
```



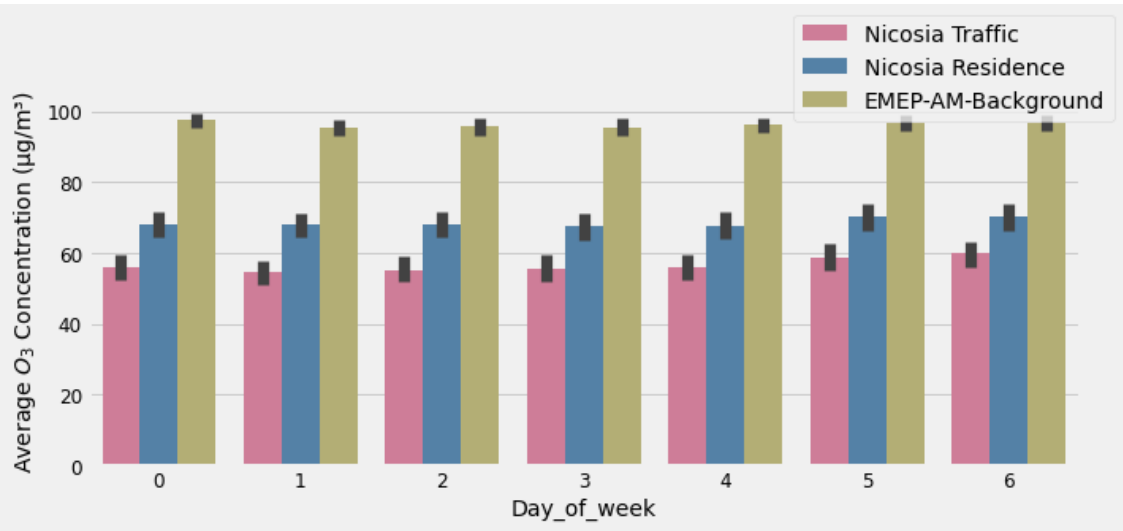
[25]:
```
fig= plt.figure(figsize=(10,4))
sns.barplot(x="Day_of_week", y="NO_2", hue="Station", data=result,␣
 ↪palette=['cyan','olive','green'])
#plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc␣
 ↪= 2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$NO_{2}$'  + ' Concentration (µg/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()
```



[26]:
```
fig= plt.figure(figsize=(10,4))
sns.barplot(x="Day_of_week", y="SO_2", hue="Station", data=result,␣
 ↪palette=['plum','aqua','chartreuse'])
#plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc␣
 ↪= 2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$SO_{2}$'  + ' Concentration (µg/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()
```

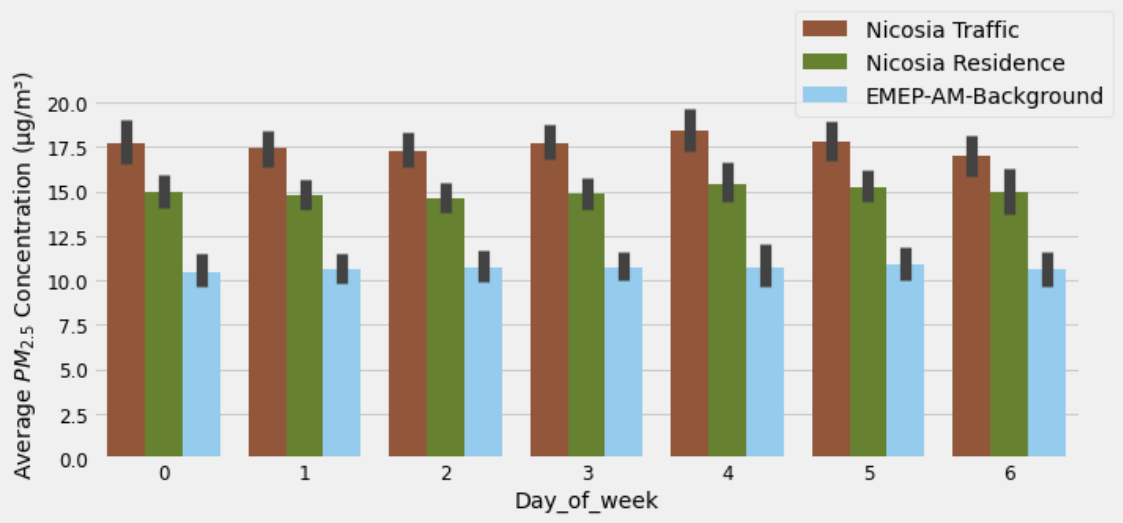


[27]:
```
fig= plt.figure(figsize=(10,4))
sns.barplot(x="Day_of_week", y="O_3", hue="Station", data=result,␣
 ↪palette=['palevioletred','steelblue','darkkhaki'])
#plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc␣
 ↪= 2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$O_{3}$'  + ' Concentration (µg/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()
```

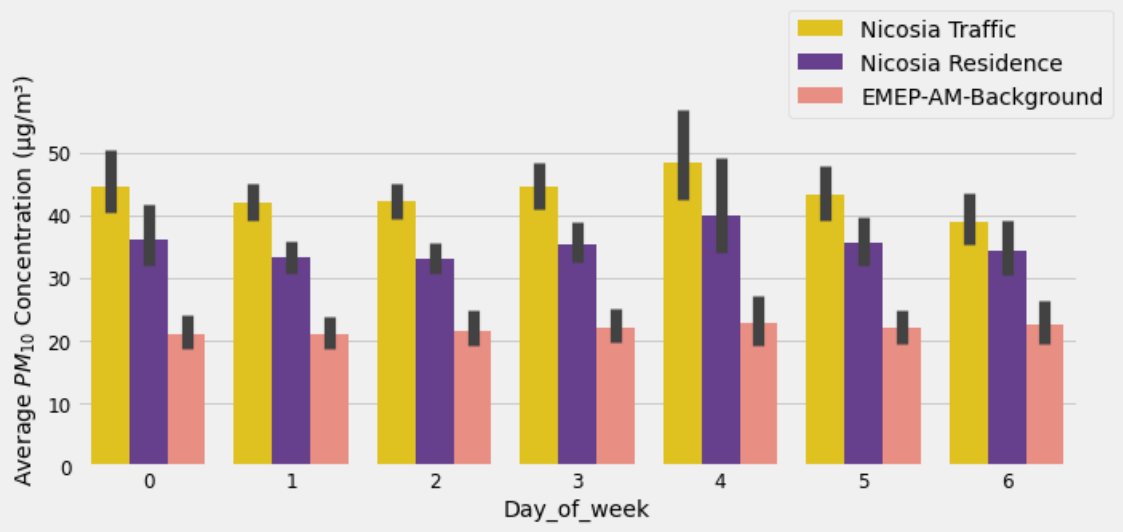


[28]:
```
fig= plt.figure(figsize=(10,4))
```

```
sns.barplot(x="Day_of_week", y="PM_2.5", hue="Station", data=result,␣
 ↪palette=['sienna','olivedrab','lightskyblue'])
#plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc␣
 ↪= 2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$PM_{2.5}$'  + ' Concentration (µg/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()
```

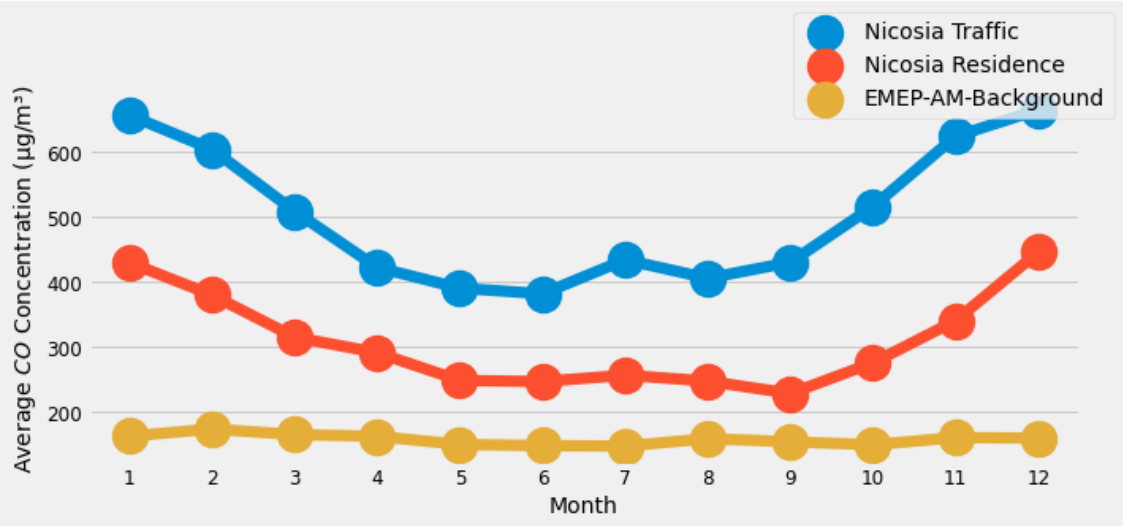


[29]:
```
fig= plt.figure(figsize=(10,4))
sns.barplot(x="Day_of_week", y="PM_10", hue="Station", data=result,␣
 ↪palette=['gold','rebeccapurple','salmon'])
#plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc␣
 ↪= 2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$PM_{10}$'  + ' Concentration (µg/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()
```

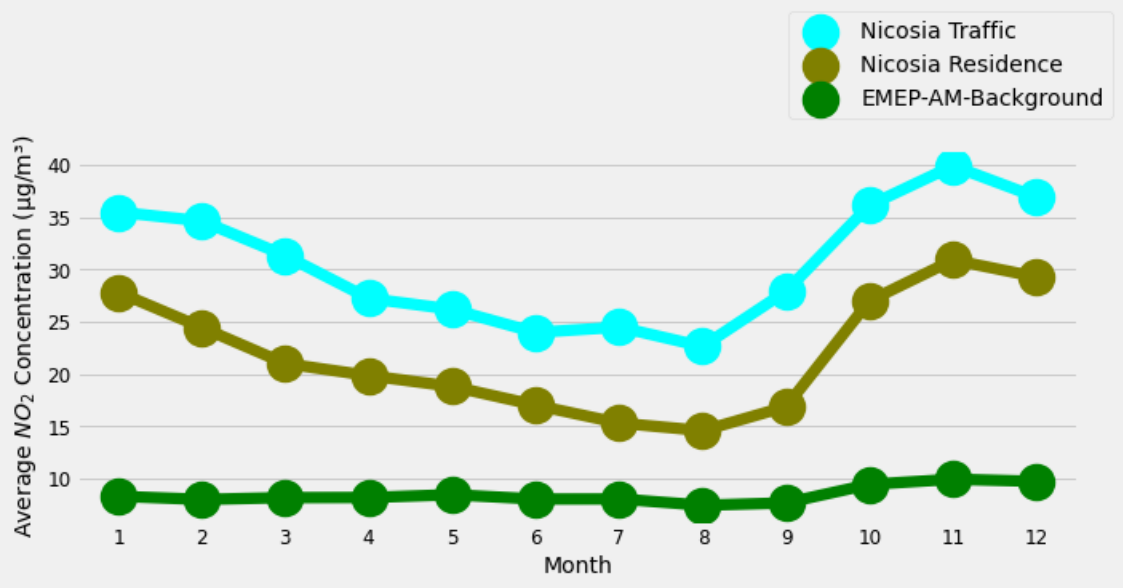


1.6 Monthly Visualization

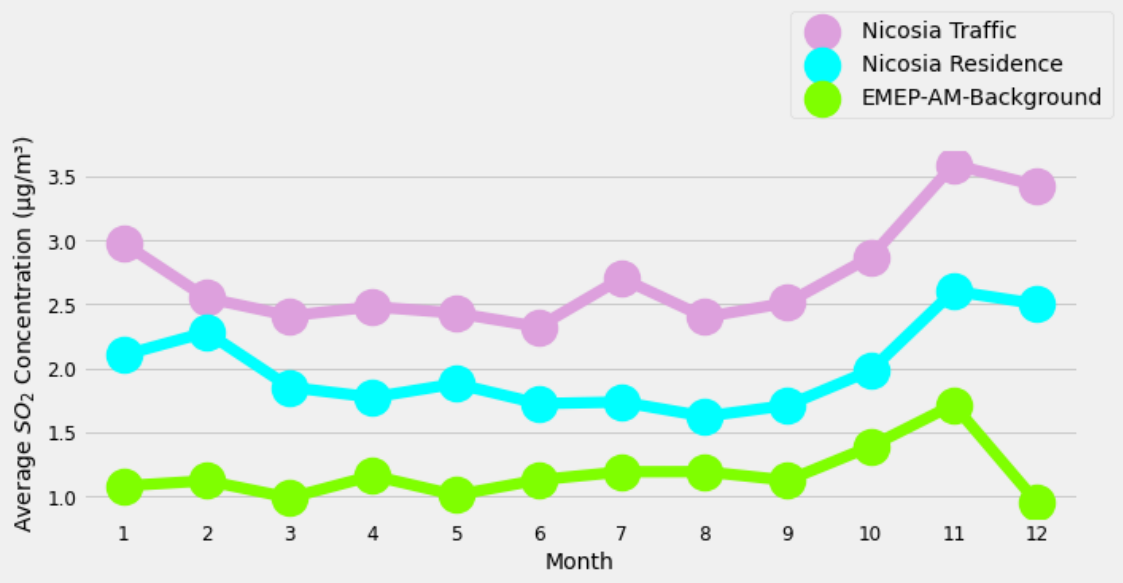
[30]:
```
fig= plt.figure(figsize=(10,4))
sns.pointplot(x="Month", y="CO", hue="Station", data=result,
              ci=None)
#plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc
 ↪= 2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$CO_{}$'  + ' Concentration (µg/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()
```



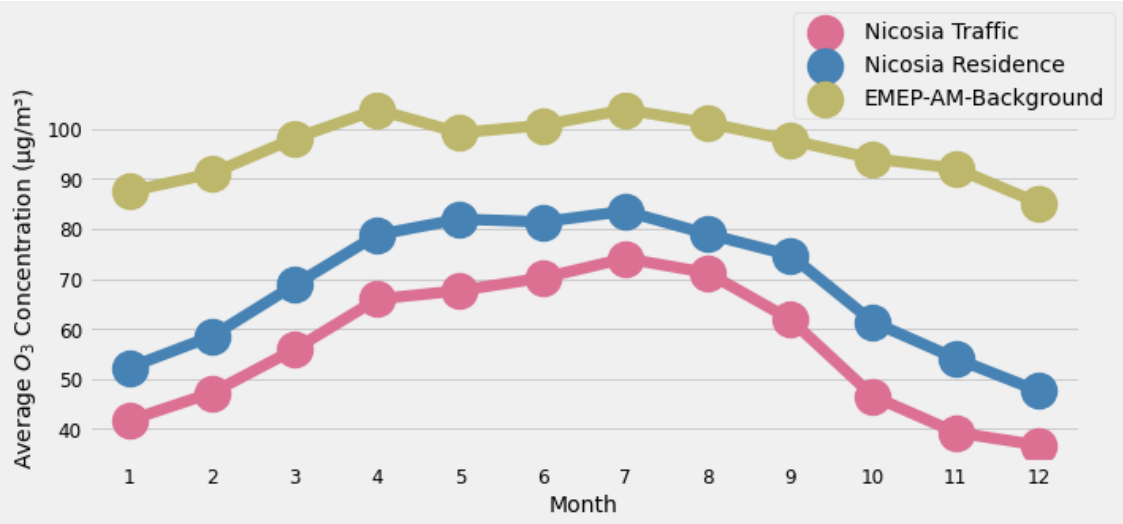
[31]:
```
fig= plt.figure(figsize=(10,4))
sns.pointplot(x="Month", y="NO_2", hue="Station",␣
 ↪data=result,palette=['cyan','olive','green'],
              ci=None)
#plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc␣
 ↪= 2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$NO_{2}$'  + ' Concentration (µg/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.4))
plt.show()
```



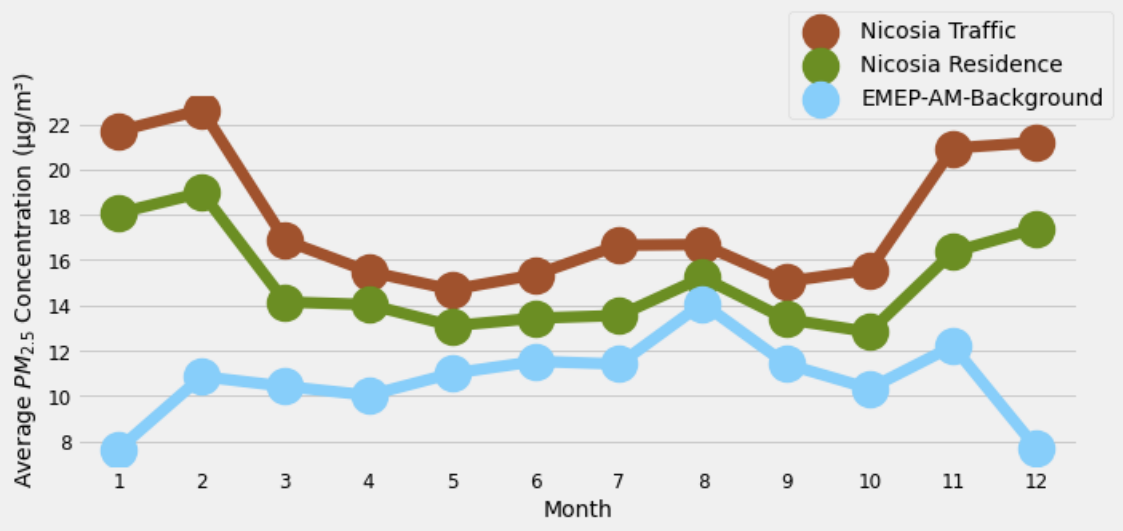
[32]:
```
fig= plt.figure(figsize=(10,4))
sns.pointplot(x="Month", y="SO_2", hue="Station",␣
 ↪data=result,palette=['plum','aqua','chartreuse'],
              ci=None)
#plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc␣
 ↪= 2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$SO_{2}$'  + ' Concentration (µg/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.4))
plt.show()
```



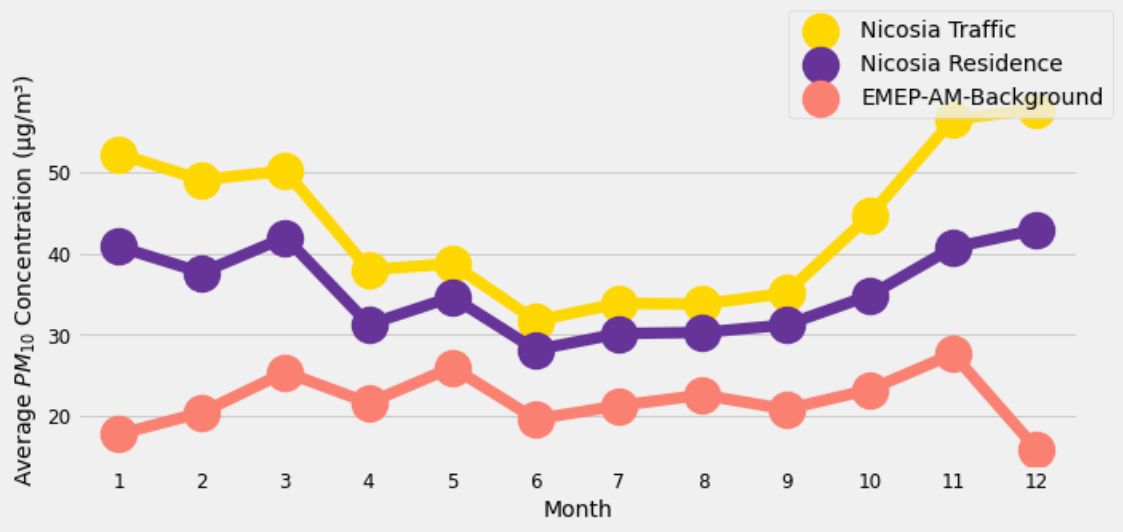
[33]:
```
fig= plt.figure(figsize=(10,4))
sns.pointplot(x="Month", y="O_3", hue="Station",␣
 ↪data=result,palette=['palevioletred','steelblue','darkkhaki'],
              ci=None)
#plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc␣
 ↪= 2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$O_{3}$'  + ' Concentration (µg/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()
```



[34]:
```
fig= plt.figure(figsize=(10,4))
sns.pointplot(x="Month", y="PM_2.5", hue="Station",␣
 ↪data=result,palette=['sienna','olivedrab','lightskyblue'],
              ci=None)
#plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc␣
 ↪= 2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$PM_{2.5}$'  + ' Concentration (µg/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()
```

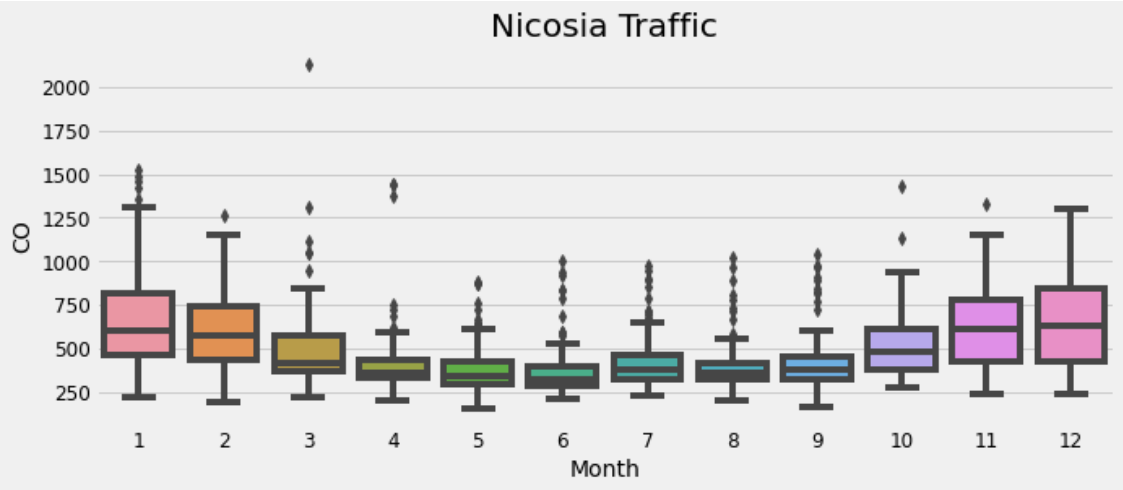


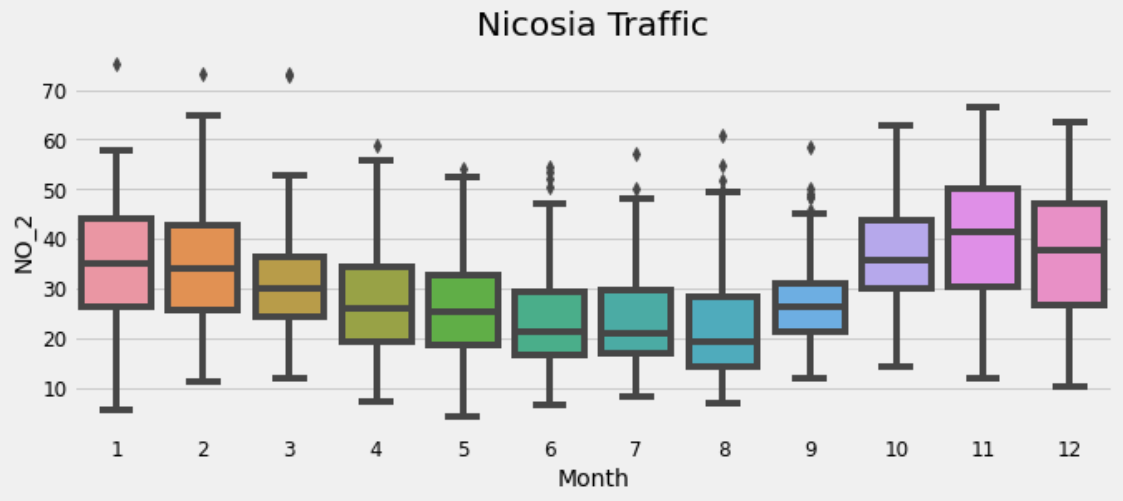
[35]:
```
fig= plt.figure(figsize=(10,4))
sns.pointplot(x="Month", y="PM_10", hue="Station",␣
 ↪data=result,palette=['gold','rebeccapurple','salmon'],
              ci=None)
#plt.legend(['Nicosia Traffic', 'Nicosia Residence','EMEP-AM-Background'], loc␣
 ↪= 2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$PM_{10}$'  + ' Concentration (µg/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()
```



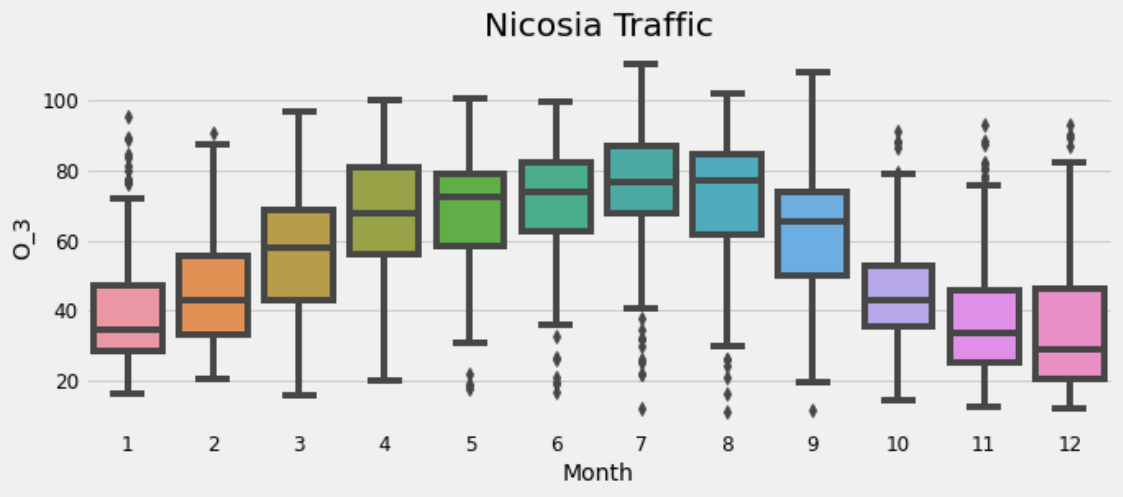
[36]:
```
palette=[['cyan','olive','green'],['plum','aqua','chartreuse'],['palevioletred','steelblue','d
                                                                                              ␣
 ↪['sienna','olivedrab','lightskyblue'],['gold','rebeccapurple','salmon'],['plum','olivedrab'
```

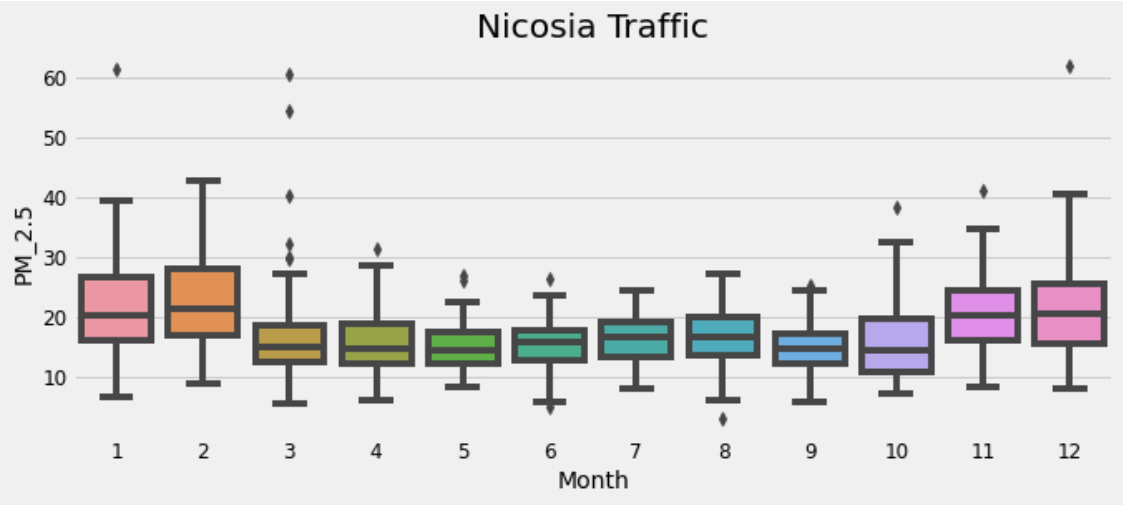
[37]:
```
column=['CO','NO_2','SO_2','O_3','PM_2.5','PM_10']
for i in range(len(column)):
    fig= plt.figure(figsize=(10,4))
    sns.boxplot(data=data_NT, x='Month', y=column[i])
    plt.title('Nicosia Traffic')
    plt.show()
```

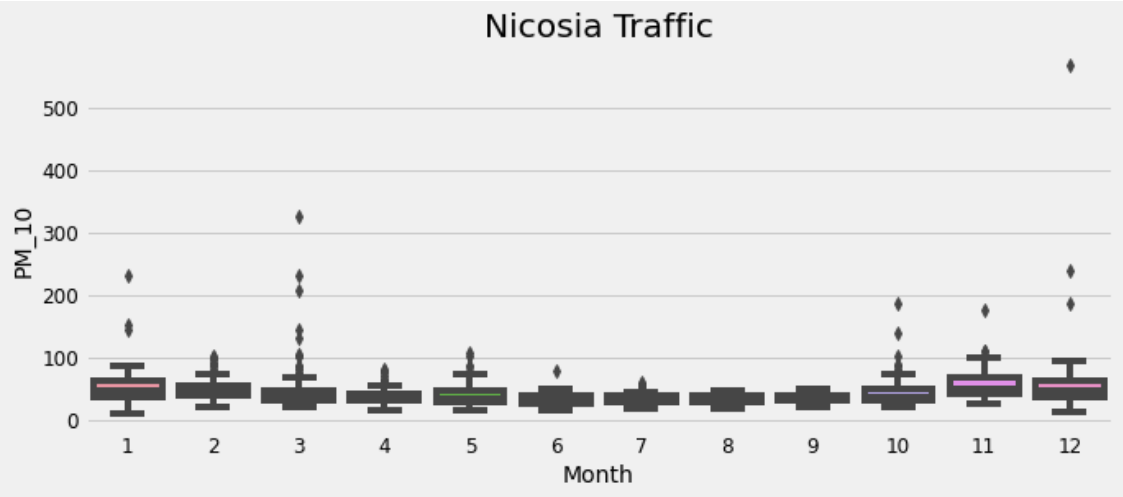



Nicosia Traffic
NO_2
70
60
50
40
30
20
10
1
2
3
4
5
6
7
8
9
10
11
12
Month

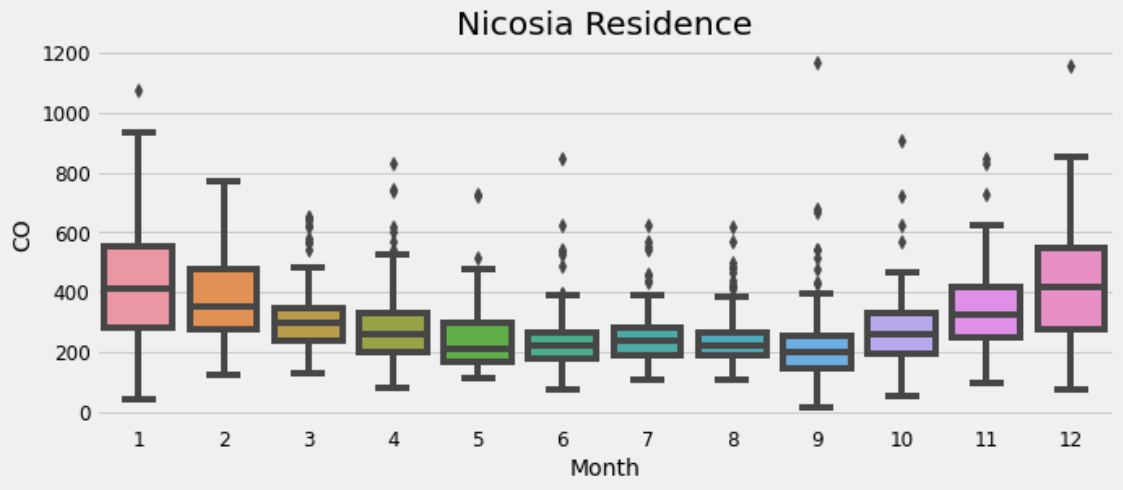

Nicosia Traffic
SO_2
8
6
4
2
0
1
2
3
4
5
6
7
8
9
10
11
12
Month

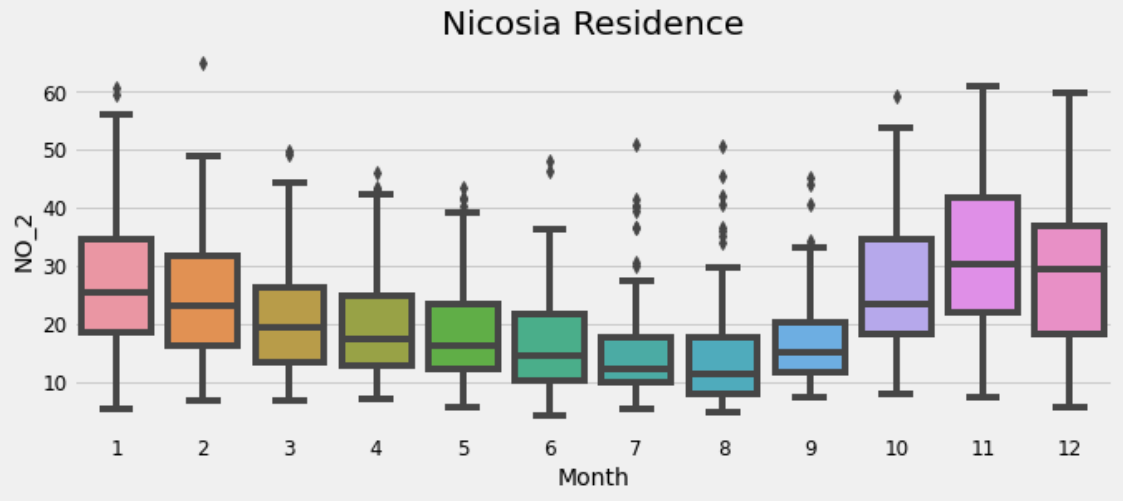

Nicosia Traffic
O_3
100
80
60
40
20
1
2
3
4
5
6
7
8
9
10
11
12
Month

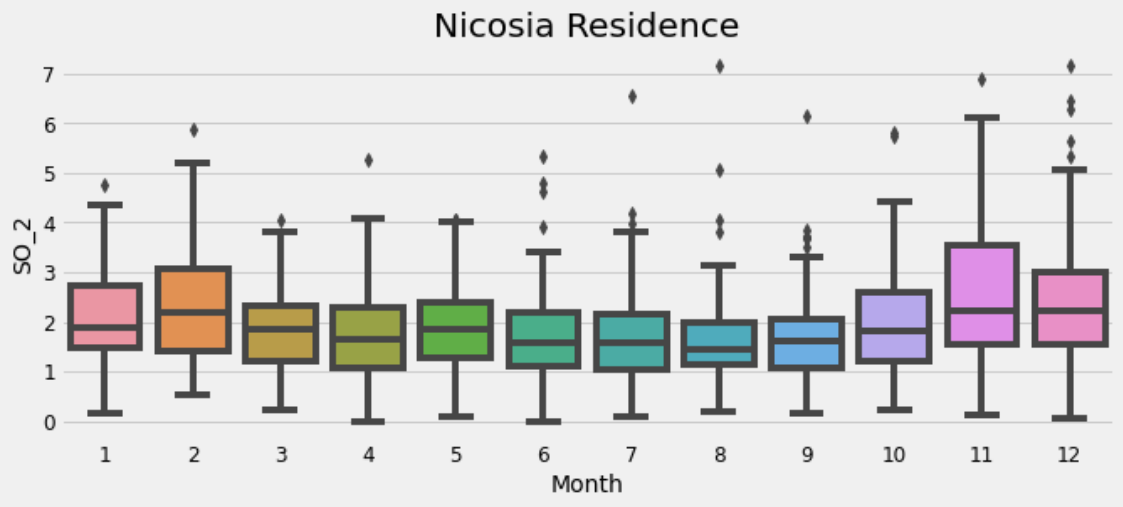

Nicosia Traffic
PM_2.5
60
50
40
30
20
10
1
2
3
4
5
6
7
8
9
10
11
12
Month

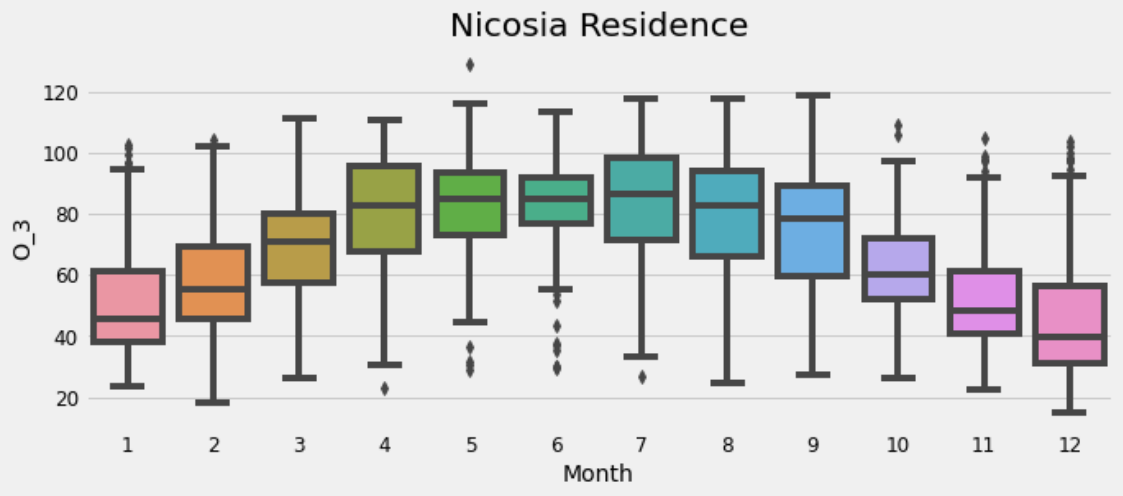

Nicosia Traffic
PM_10
500
400
300
200
100
0
1
2
3
4
5
6
7
8
9
10
11
12
Month

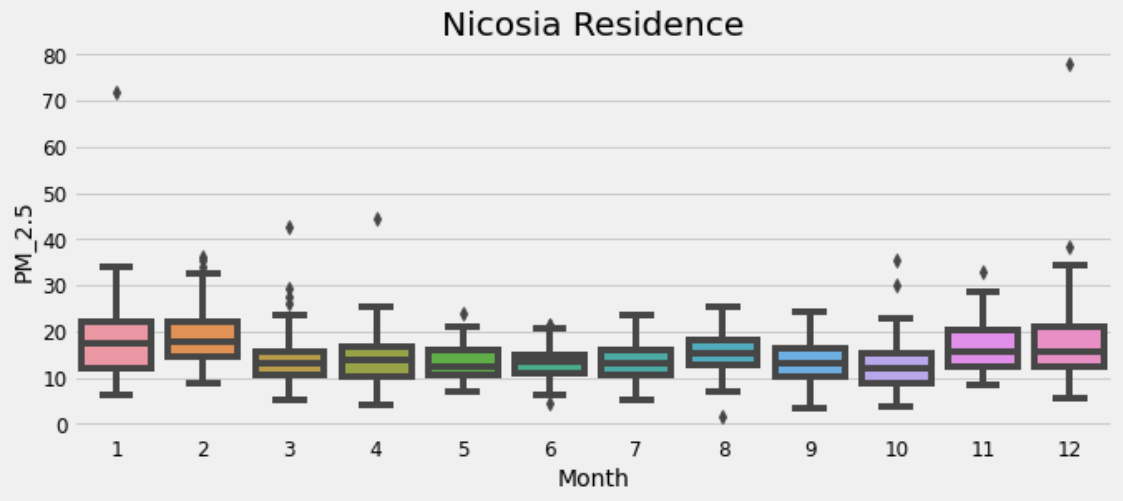
[38]:
```
column=['CO','NO_2','SO_2','O_3','PM_2.5','PM_10']
for i in range(len(column)):
    fig= plt.figure(figsize=(10,4))
    sns.boxplot(data=data_NR, x='Month', y=column[i])
    plt.title('Nicosia Residence')
    plt.show()
```

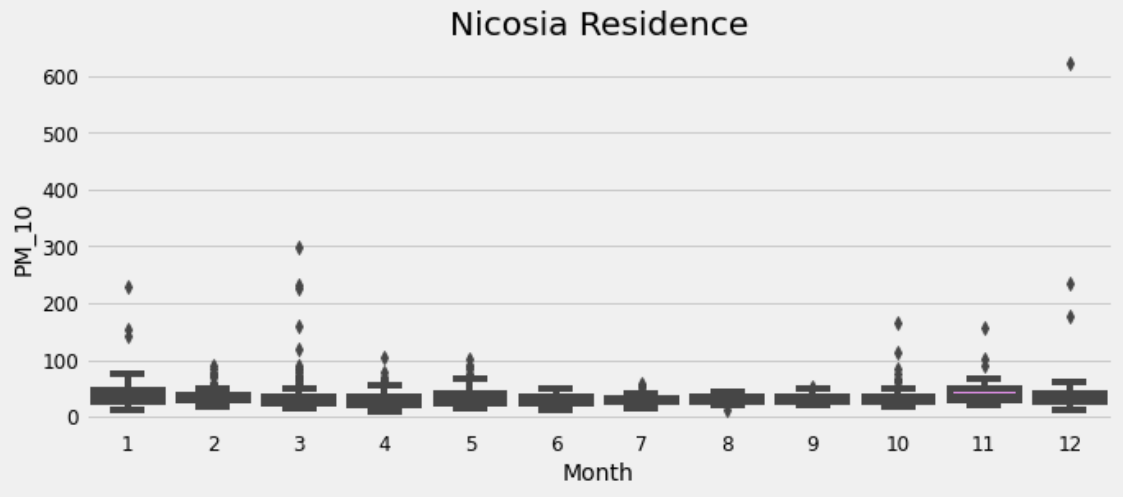




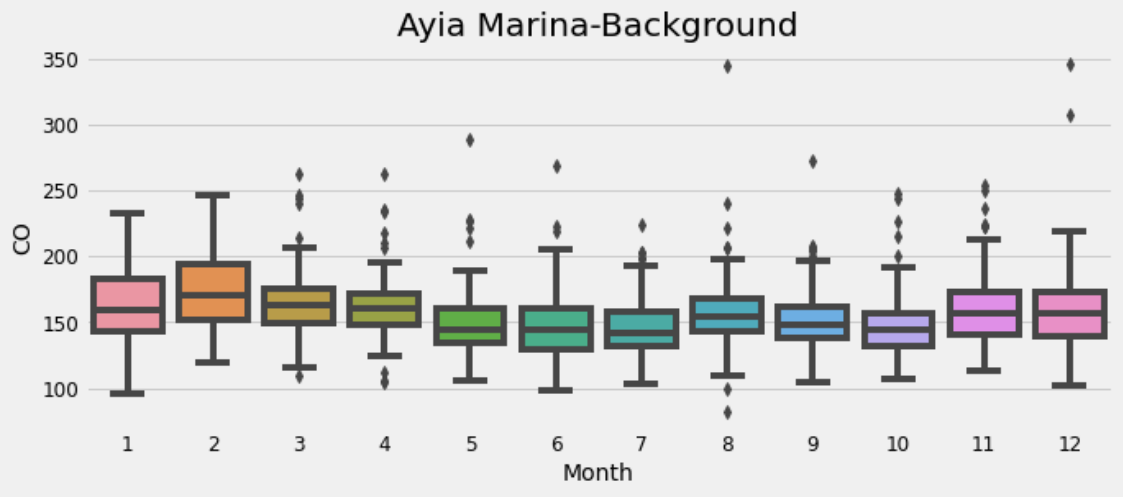

Nicosia Residence
SO_2
Month
0
1
2
3
4
5
6
7
1
2
3
4
5
6
7
8
9
10
11
12

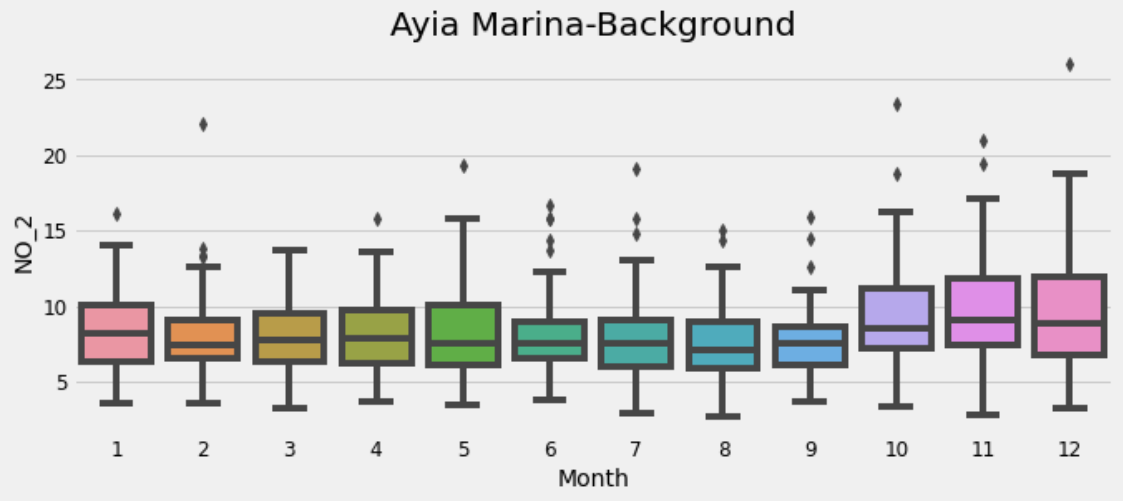

Nicosia Residence
O_3
Month
20
40
60
80
100
120
1
2
3
4
5
6
7
8
9
10
11
12

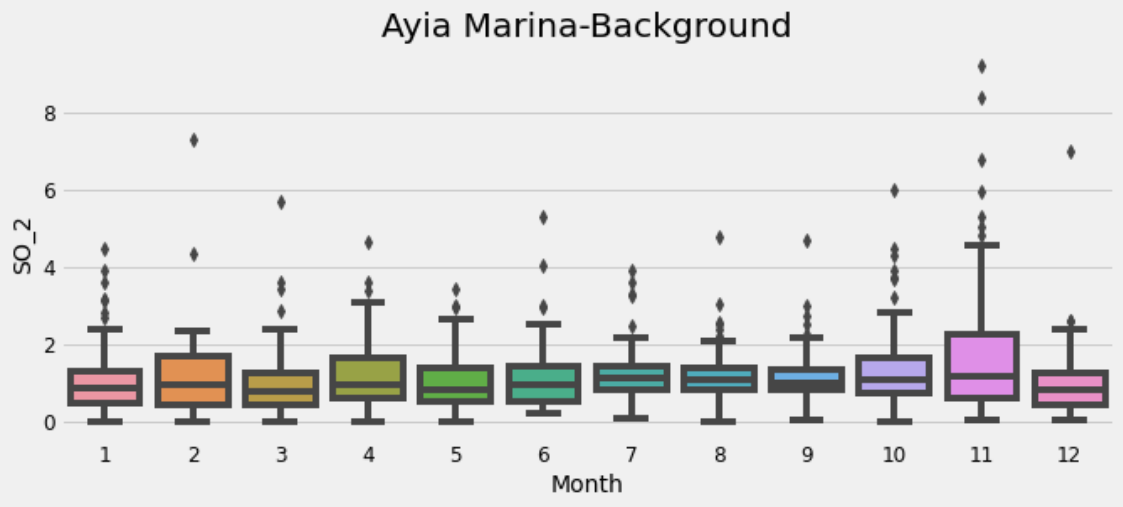


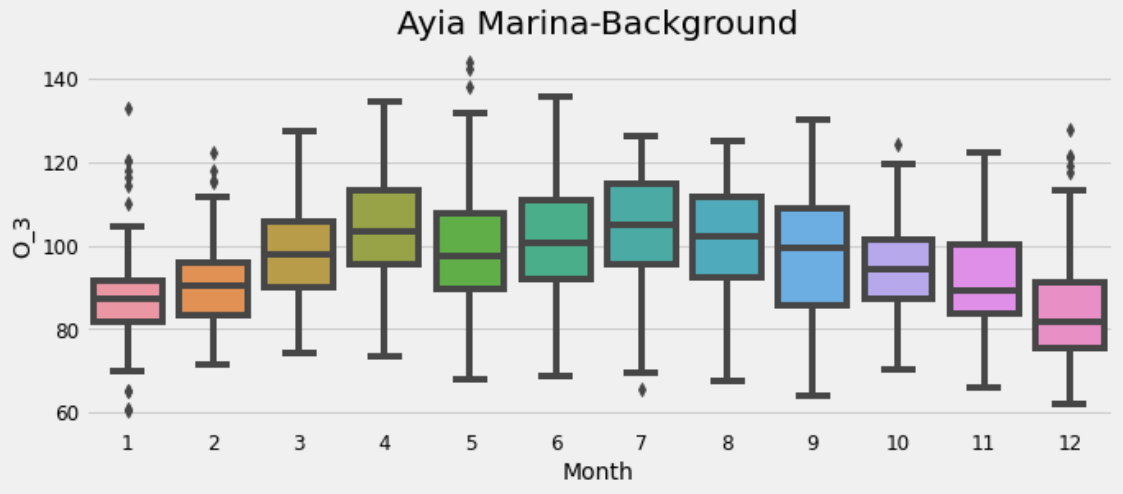


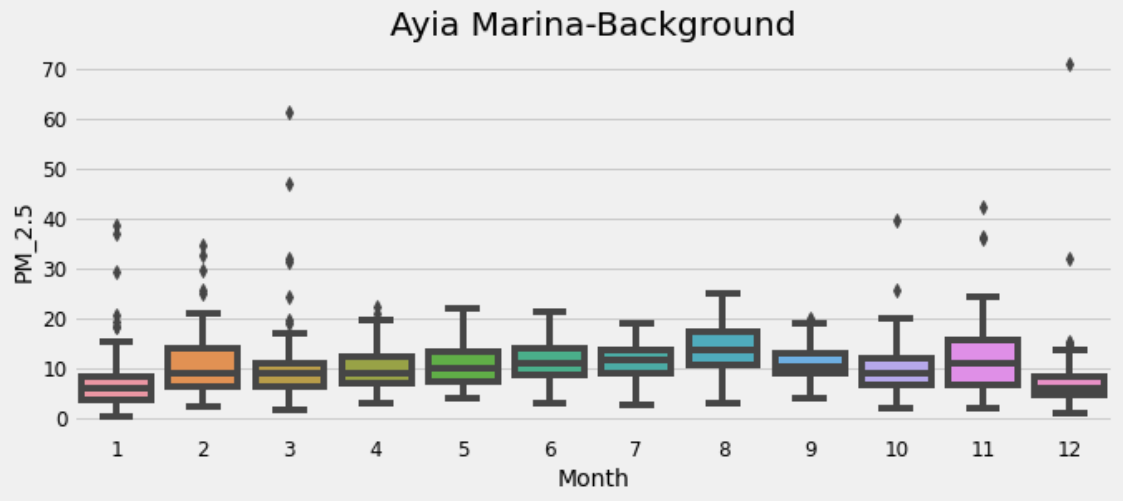
[39]:
```
column=['CO','NO_2','SO_2','O_3','PM_2.5','PM_10']
for i in range(len(column)):
    fig= plt.figure(figsize=(10,4))
    sns.boxplot(data=data_AM, x='Month', y=column[i])
    plt.title('Ayia Marina-Background')
    plt.show()
```

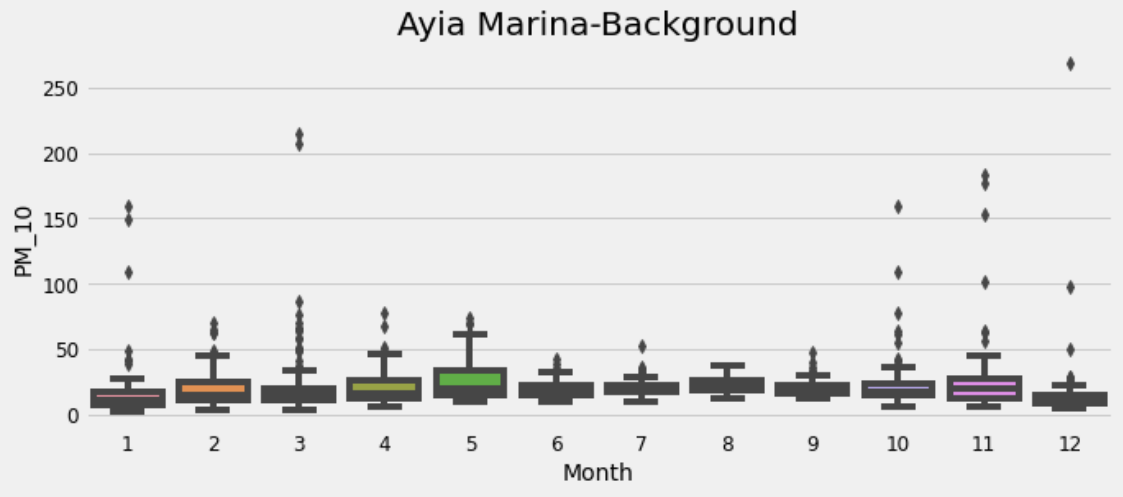

Ayia Marina-Background
CO
350
300
250
200
150
100
1
2
3
4
5
6
7
8
9
10
11
12
Month


Ayia Marina-Background
NO_2
25
20
15
10
5
1
2
3
4
5
6
7
8
9
10
11
12
Month


Ayia Marina-Background
SO_2
0
2
4
6
8
1
2
3
4
5
6
7
8
9
10
11
12
Month

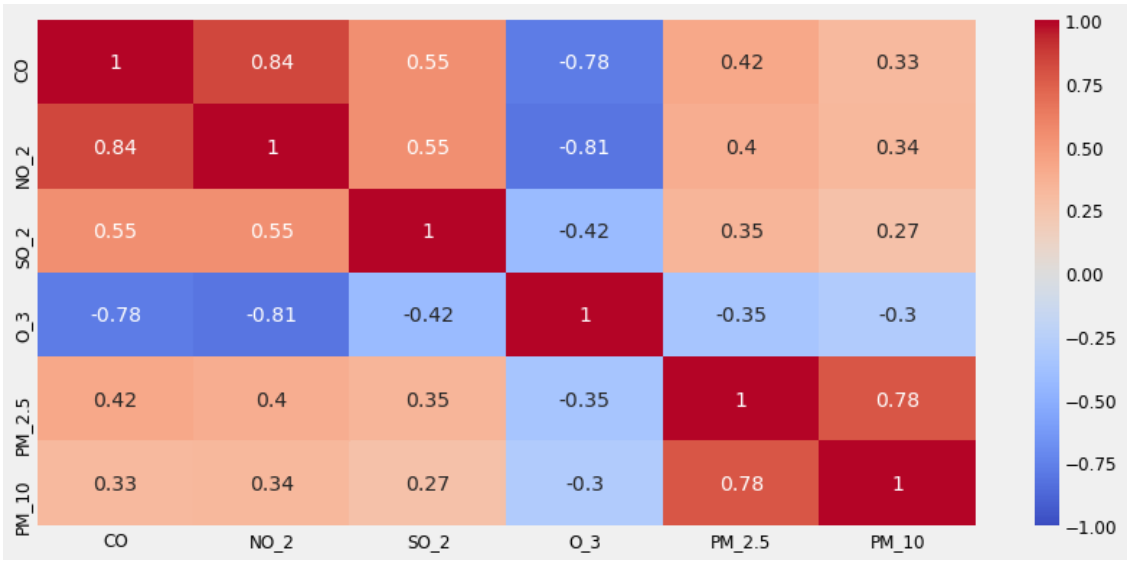

Ayia Marina-Background
O_3
60
80
100
120
140
1
2
3
4
5
6
7
8
9
10
11
12
Month





[40]:
```
fig= plt.figure(figsize=(13,6))
sns.heatmap(result[['CO','NO_2','SO_2','O_3','PM_2.5','PM_10']].corr(), annot =␣
↪True, vmin=-1, vmax=1,
            center= 0, cmap= 'coolwarm')
```

[40]: <AxesSubplot:>



1.7 AQI Nicosia Traffic

[41]:
```
NO_2=data_NT['NO_2'].to_list()
I_NO_2=[]
for i in range(len(NO_2)):
    if NO_2[i]>=0 and NO_2[i]<=50:
        I=((25-0)/(50-0))*(NO_2[i]-0)+0
    elif NO_2[i]>=51 and NO_2[i]<=100:
        I=((50-25)/(100-50))*(NO_2[i]-50)+25
    elif NO_2[i]>=101 and NO_2[i]<=200:
        I=((75-50)/(200-100))*(NO_2[i]-100)+50
    elif NO_2[i]>=201 and NO_2[i]<=400:
        I=((100-75)/(400-200))*(NO_2[i]-200)+75
    else:
        I=101
    I_NO_2.append(I)
```

[42]:
```
O_3=data_NT['O_3'].to_list()
I_O_3=[]
for i in range(len(O_3)):
    if O_3[i]>=0 and O_3[i]<=60:
        I=((25-0)/(60-0))*(O_3[i]-0)+0
    elif O_3[i]>=61 and O_3[i]<=120:
        I=((50-25)/(120-60))*(O_3[i]-60)+25
    elif O_3[i]>=121 and O_3[i]<=180:
        I=((75-50)/(180-120))*(O_3[i]-120)+50
    elif O_3[i]>=181 and O_3[i]<=240:
```

```
        I=((100-75)/(240-180))*(O_3[i]-180)+75
    else:
        I=101
    I_O_3.append(I)
len(I_O_3)
```

[42]: 1095

[43]:
```
PM_2_5=data_NT['PM_2.5'].to_list()
I_PM_2_5=[]
for i in range(len(PM_2_5)):
    if PM_2_5[i]>=0 and PM_2_5[i]<=15:
        I=((25-0)/(15-0))*(PM_2_5[i]-0)+0
    elif PM_2_5[i]>=16 and PM_2_5[i]<=30:
        I=((50-25)/(30-15))*(PM_2_5[i]-15)+25
    elif PM_2_5[i]>=31 and PM_2_5[i]<=55:
        I=((75-50)/(55-30))*(PM_2_5[i]-30)+50
    elif PM_2_5[i]>=56 and PM_2_5[i]<=110:
        I=((100-75)/(110-55))*(PM_2_5[i]-55)+75
    else:
        I=101
    I_PM_2_5.append(I)

len(I_PM_2_5)
```

[43]: 1095

[44]:
```
PM_10=data_NT['PM_10'].to_list()
I_PM_10=[]
for i in range(len(PM_10)):
    if PM_10[i]>=0 and PM_10[i]<=25:
        I=((25-0)/(25-0))*(PM_10[i]-0)+0
    elif PM_10[i]>=26 and PM_10[i]<=50:
        I=((50-25)/(50-25))*(PM_10[i]-25)+25
    elif PM_10[i]>=51 and PM_10[i]<=90:
        I=((75-50)/(90-50))*(PM_10[i]-50)+50
    elif PM_10[i]>=91 and PM_10[i]<=180:
        I=((100-75)/(180-90))*(PM_10[i]-90)+75
    else:
        I=101
    I_PM_10.append(I)
```

[45]:
```
l=[[I_NO_2[i],I_O_3[i],I_PM_2_5[i],I_PM_10[i]] for i in range(1095)]
AQI=[np.round(max(l[i]),2) for i in range(1095)]
l_pollution=[]
for i in range(len(AQI)):
    if AQI[i]>=0 and AQI[i]<=50:
```

```
        l_pollution.append('Good')
    elif AQI[i]>=51 and AQI[i]<=75:
        l_pollution.append('Moderate')
    else:
        l_pollution.append('Unhealthy')

data_NT['AQI']=AQI
data_NT['Pollution Level']=l_pollution
data_NT.head()
```

[45]:

Date	NO_2	SO_2	O_3	CO	PM_2.5	PM_10	Year	\
2017-01-01	22.656250	1.409583	31.888332	632.104187	17.4	30.5	2017	
2017-01-02	53.323479	1.470435	34.365417	888.864807	16.1	21.1	2017	
2017-01-03	40.632275	2.226364	46.317081	535.313354	30.4	80.7	2017	
2017-01-04	20.976250	0.873750	77.764168	315.857483	35.1	86.3	2017	
2017-01-05	18.100000	3.159167	89.449165	270.007507	28.7	70.2	2017	

Date	Month	Day_of_week	Day	Hour	AQI	Pollution Level
2017-01-01	1	6	1	0	30.50	Good
2017-01-02	1	0	2	0	26.83	Good
2017-01-03	1	1	3	0	101.00	Unhealthy
2017-01-04	1	2	4	0	72.69	Moderate
2017-01-05	1	3	5	0	62.62	Moderate

1.8 AQI Nicosia Residence

[46]:
```
NO_2=data_NR['NO_2'].to_list()
I_NO_2=[]
for i in range(len(NO_2)):
    if NO_2[i]>=0 and NO_2[i]<=50:
        I=((25-0)/(50-0))*(NO_2[i]-0)+0
    elif NO_2[i]>=51 and NO_2[i]<=100:
        I=((50-25)/(100-50))*(NO_2[i]-50)+25
    elif NO_2[i]>=101 and NO_2[i]<=200:
        I=((75-50)/(200-100))*(NO_2[i]-100)+50
    elif NO_2[i]>=201 and NO_2[i]<=400:
        I=((100-75)/(400-200))*(NO_2[i]-200)+75
    else:
        I=101
    I_NO_2.append(I)
```

[47]:
```
O_3=data_NR['O_3'].to_list()
I_O_3=[]
for i in range(len(O_3)):
```

```
    if O_3[i]>=0 and O_3[i]<=60:
        I=((25-0)/(60-0))*(O_3[i]-0)+0
    elif O_3[i]>=61 and O_3[i]<=120:
        I=((50-25)/(120-60))*(O_3[i]-60)+25
    elif O_3[i]>=121 and O_3[i]<=180:
        I=((75-50)/(180-120))*(O_3[i]-120)+50
    elif O_3[i]>=181 and O_3[i]<=240:
        I=((100-75)/(240-180))*(O_3[i]-180)+75
    else:
        I=101
    I_O_3.append(I)
```

[48]:
```
PM_2_5=data_NR['PM_2.5'].to_list()
I_PM_2_5=[]
for i in range(len(PM_2_5)):
    if PM_2_5[i]>=0 and PM_2_5[i]<=15:
        I=((25-0)/(15-0))*(PM_2_5[i]-0)+0
    elif PM_2_5[i]>=16 and PM_2_5[i]<=30:
        I=((50-25)/(30-15))*(PM_2_5[i]-15)+25
    elif PM_2_5[i]>=31 and PM_2_5[i]<=55:
        I=((75-50)/(55-30))*(PM_2_5[i]-30)+50
    elif PM_2_5[i]>=56 and PM_2_5[i]<=110:
        I=((100-75)/(110-55))*(PM_2_5[i]-55)+75
    else:
        I=101
    I_PM_2_5.append(I)
```

[49]:
```
PM_10=data_NR['PM_10'].to_list()
I_PM_10=[]
for i in range(len(PM_10)):
    if PM_10[i]>=0 and PM_10[i]<=25:
        I=((25-0)/(25-0))*(PM_10[i]-0)+0
    elif PM_10[i]>=26 and PM_10[i]<=50:
        I=((50-25)/(50-25))*(PM_10[i]-25)+25
    elif PM_10[i]>=51 and PM_10[i]<=90:
        I=((75-50)/(90-50))*(PM_10[i]-50)+50
    elif PM_10[i]>=91 and PM_10[i]<=180:
        I=((100-75)/(180-90))*(PM_10[i]-90)+75
    else:
        I=101
    I_PM_10.append(I)
```

[50]:
```
l=[[I_NO_2[i],I_O_3[i],I_PM_2_5[i],I_PM_10[i]] for i in range(1095)]
AQI=[np.round(max(l[i]),2) for i in range(1095)]
l_pollution=[]
for i in range(len(AQI)):
    if AQI[i]>=0 and AQI[i]<=50:
```

```
        l_pollution.append('Good')
    elif AQI[i]>=51 and AQI[i]<=75:
        l_pollution.append('Moderate')
    else:
        l_pollution.append('Unhealthy')

data_NR['AQI']=AQI
data_NR['Pollution Level']=l_pollution
data_NR.head()
```

[50]:

Date	NO_2	SO_2	O_3	CO	PM_2.5	PM_10	Year	\
2017-01-01	22.525000	2.425000	35.771667	553.710815	16.3	26.7	2017	
2017-01-02	31.235834	3.890417	43.341667	629.263916	16.3	20.9	2017	
2017-01-03	22.701668	3.487917	56.488750	304.118683	17.9	32.1	2017	
2017-01-04	17.751667	2.367500	78.111252	299.715424	26.2	48.6	2017	
2017-01-05	11.007916	2.674167	101.402084	150.582916	24.6	44.5	2017	

Date	Month	Day_of_week	Day	Hour	AQI	Pollution Level
2017-01-01	1	6	1	0	27.17	Good
2017-01-02	1	0	2	0	27.17	Good
2017-01-03	1	1	3	0	32.10	Good
2017-01-04	1	2	4	0	48.60	Good
2017-01-05	1	3	5	0	44.50	Good

1.9 EMEP-AM-Background

[51]:
```
NO_2=data_AM['NO_2'].to_list()
I_NO_2=[]
for i in range(len(NO_2)):
    if NO_2[i]>=0 and NO_2[i]<=50:
        I=((25-0)/(50-0))*(NO_2[i]-0)+0
    elif NO_2[i]>=51 and NO_2[i]<=100:
        I=((50-25)/(100-50))*(NO_2[i]-50)+25
    elif NO_2[i]>=101 and NO_2[i]<=200:
        I=((75-50)/(200-100))*(NO_2[i]-100)+50
    elif NO_2[i]>=201 and NO_2[i]<=400:
        I=((100-75)/(400-200))*(NO_2[i]-200)+75
    else:
        I=101
    I_NO_2.append(I)
```

[52]:
```
O_3=data_AM['O_3'].to_list()
I_O_3=[]
for i in range(len(O_3)):
```

```
    if O_3[i]>=0 and O_3[i]<=60:
        I=((25-0)/(60-0))*(O_3[i]-0)+0
    elif O_3[i]>=61 and O_3[i]<=120:
        I=((50-25)/(120-60))*(O_3[i]-60)+25
    elif O_3[i]>=121 and O_3[i]<=180:
        I=((75-50)/(180-120))*(O_3[i]-120)+50
    elif O_3[i]>=181 and O_3[i]<=240:
        I=((100-75)/(240-180))*(O_3[i]-180)+75
    else:
        I=101
    I_O_3.append(I)
```

[53]:
```
PM_2_5=data_AM['PM_2.5'].to_list()
I_PM_2_5=[]
for i in range(len(PM_2_5)):
    if PM_2_5[i]>=0 and PM_2_5[i]<=15:
        I=((25-0)/(15-0))*(PM_2_5[i]-0)+0
    elif PM_2_5[i]>=16 and PM_2_5[i]<=30:
        I=((50-25)/(30-15))*(PM_2_5[i]-15)+25
    elif PM_2_5[i]>=31 and PM_2_5[i]<=55:
        I=((75-50)/(55-30))*(PM_2_5[i]-30)+50
    elif PM_2_5[i]>=56 and PM_2_5[i]<=110:
        I=((100-75)/(110-55))*(PM_2_5[i]-55)+75
    else:
        I=101
    I_PM_2_5.append(I)
```

[54]:
```
PM_10=data_AM['PM_10'].to_list()
I_PM_10=[]
for i in range(len(PM_10)):
    if PM_10[i]>=0 and PM_10[i]<=25:
        I=((25-0)/(25-0))*(PM_10[i]-0)+0
    elif PM_10[i]>=26 and PM_10[i]<=50:
        I=((50-25)/(50-25))*(PM_10[i]-25)+25
    elif PM_10[i]>=51 and PM_10[i]<=90:
        I=((75-50)/(90-50))*(PM_10[i]-50)+50
    elif PM_10[i]>=91 and PM_10[i]<=180:
        I=((100-75)/(180-90))*(PM_10[i]-90)+75
    else:
        I=101
    I_PM_10.append(I)
```

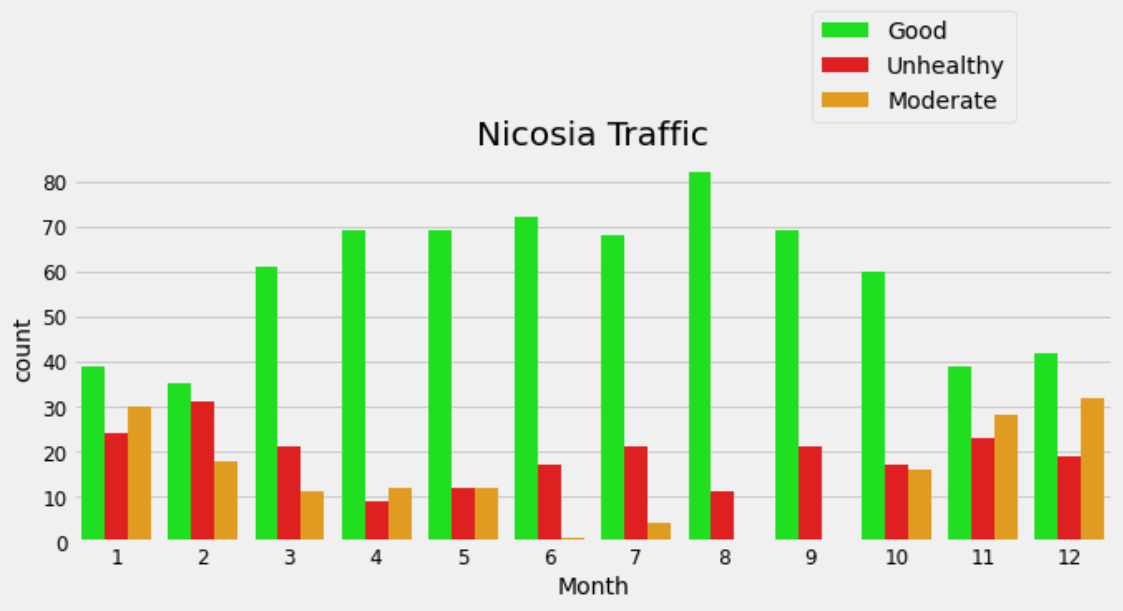
[55]:
```
l=[[I_NO_2[i],I_O_3[i],I_PM_2_5[i],I_PM_10[i]] for i in range(1095)]
AQI=[np.round(max(l[i]),2) for i in range(1095)]
l_pollution=[]
for i in range(len(AQI)):
    if AQI[i]>=0 and AQI[i]<=50:
```

```
        l_pollution.append('Good')
    elif AQI[i]>=51 and AQI[i]<=75:
        l_pollution.append('Moderate')
    else:
        l_pollution.append('Unhealthy')
data_AM['AQI']=AQI
data_AM['Pollution Level']=l_pollution
data_AM.head()
```

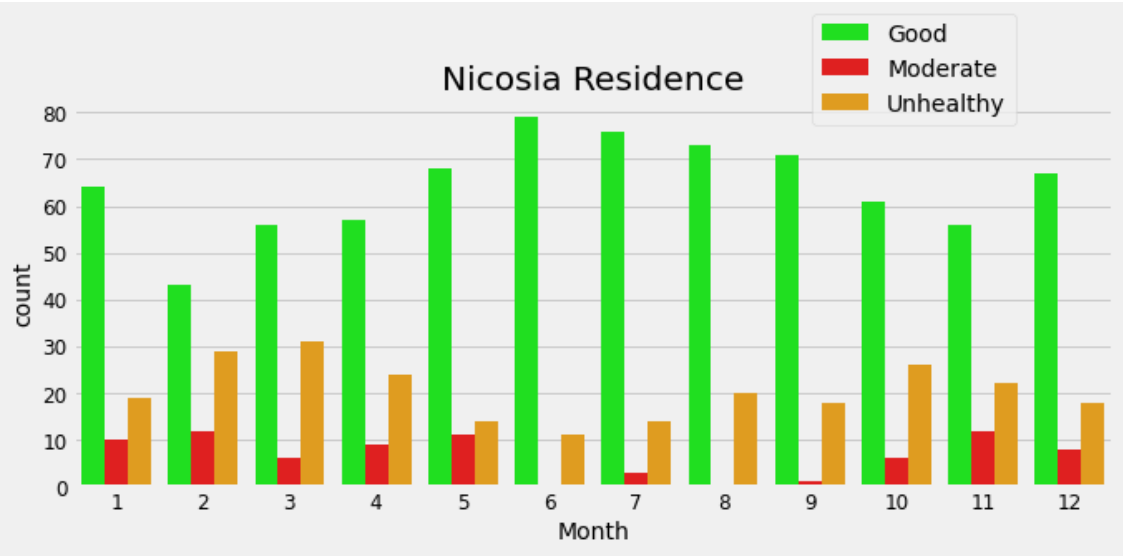
[55]:

Date	NO_2	SO_2	O_3	CO	PM_2.5	PM_10	Year	Month	Day_of_week	Day	Hour	AQI	Pollution Level
2017-01-01	3.523750	0.750870	72.954094	177.931366	2.5	6.9	2017	1	6	1	0	30.40	Good
2017-01-02	7.679584	1.183333	84.351250	200.942917	2.7	5.8	2017	1	0	2	0	35.15	Good
2017-01-03	6.755455	2.286250	93.035004	118.492271	4.0	6.3	2017	1	1	3	0	38.76	Good
2017-01-04	6.279167	0.467083	92.485832	162.221664	3.3	5.1	2017	1	2	4	0	38.54	Good
2017-01-05	10.433333	1.863750	117.720001	157.161255	2.8	6.7	2017	1	3	5	0	49.05	Good

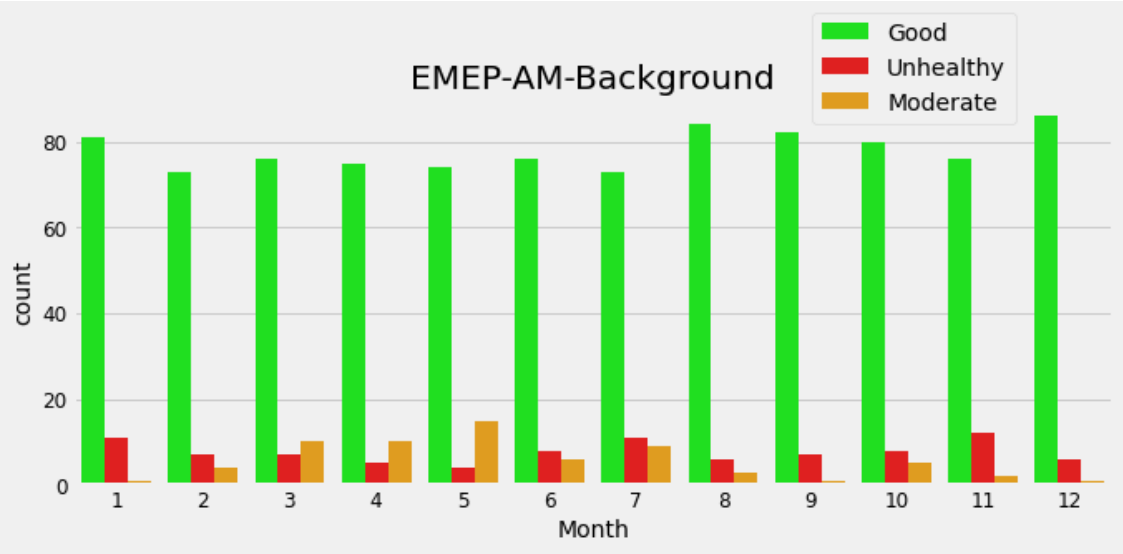
[56]:
```
fig= plt.figure(figsize=(10,4))
sns.countplot(x = 'Month', data = data_NT,hue='Pollution␣
 ↪Level',palette=['lime','red','orange'])
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.4))
plt.title('Nicosia Traffic')
plt.show()
```



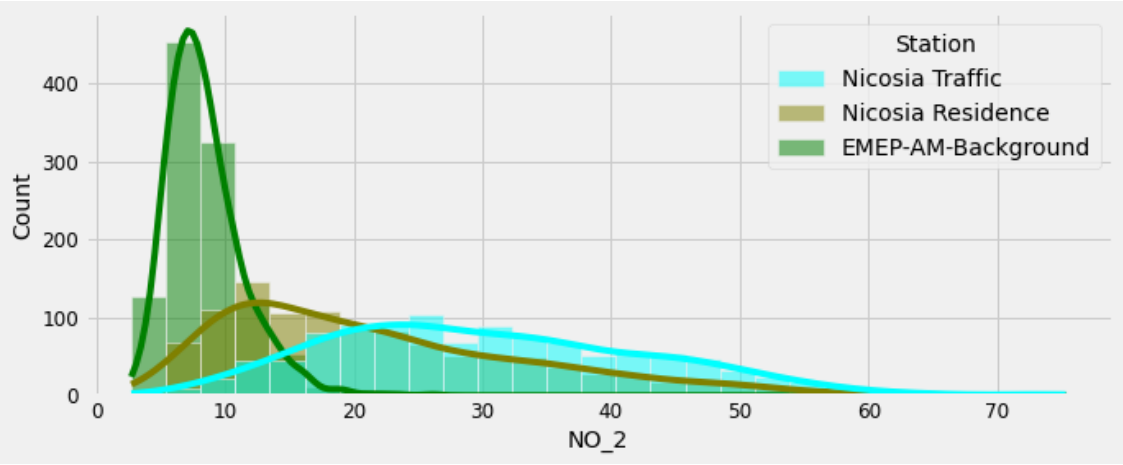
[57]:
```
fig= plt.figure(figsize=(10,4))
sns.countplot(x = 'Month', data = data_NR,hue='Pollution␣
 ↪Level',palette=['lime','red','orange'])
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.title('Nicosia Residence')
plt.show()
```



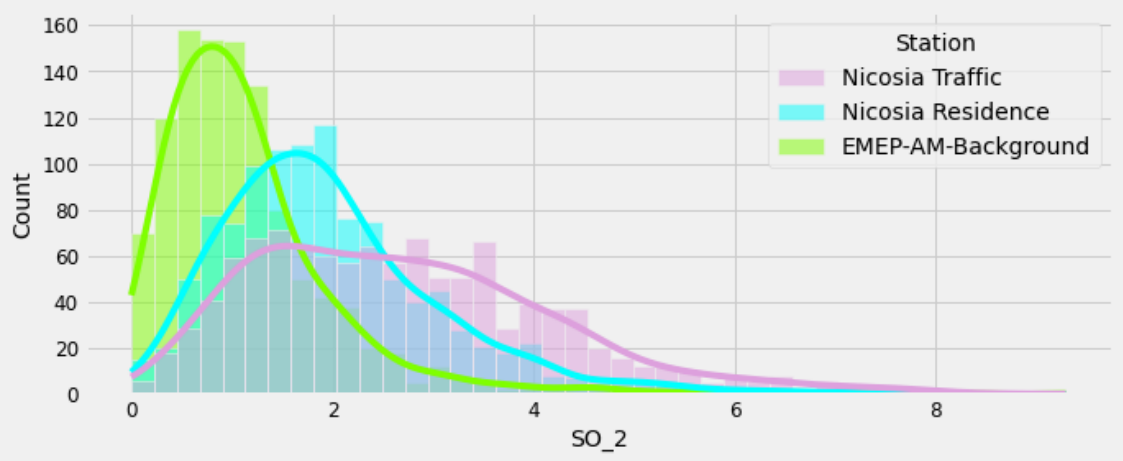
[58]:
```
fig= plt.figure(figsize=(10,4))
sns.countplot(x = 'Month', data = data_AM,hue='Pollution␣
 ↪Level',palette=['lime','red','orange'])
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.title('EMEP-AM-Background')
plt.show()
```



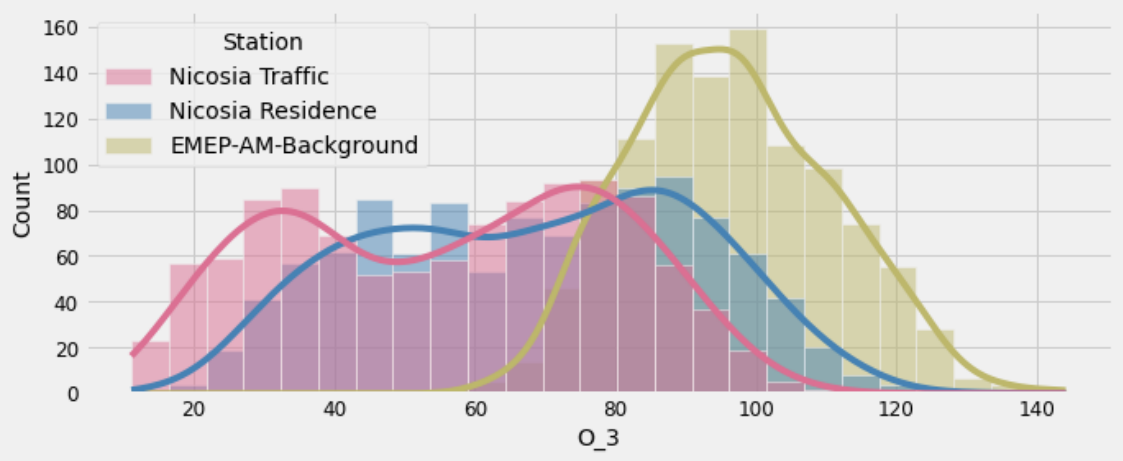
[59]:
```
# importing packages
fig= plt.figure(figsize=(10,4))
sns.histplot(x='NO_2', data=result, kde=True, hue='Station',palette=palette[0])

plt.show()
```



[60]:
```
# importing packages
fig= plt.figure(figsize=(10,4))
sns.histplot(x='SO_2', data=result, kde=True, hue='Station',palette=palette[1])

plt.show()
```

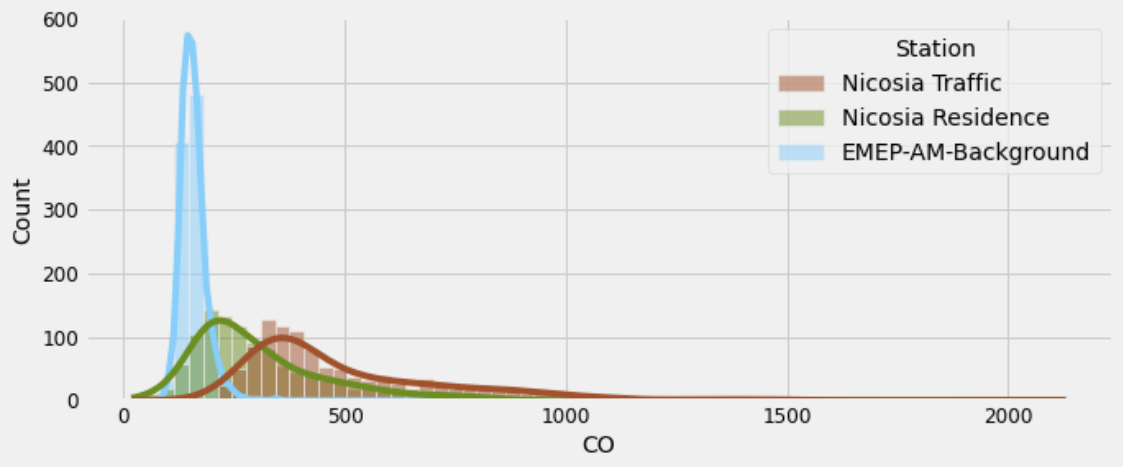


[61]:
```
# importing packages
fig= plt.figure(figsize=(10,4))
sns.histplot(x='O_3', data=result, kde=True, hue='Station',palette=palette[2])

plt.show()
```

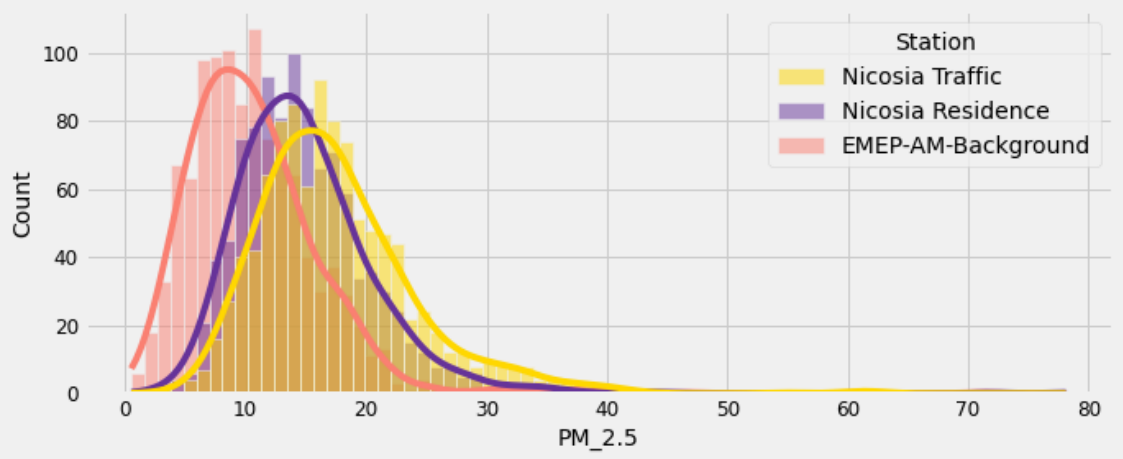


[62]:
```
# importing packages
fig= plt.figure(figsize=(10,4))
sns.histplot(x='CO', data=result, kde=True, hue='Station',palette=palette[3])
```

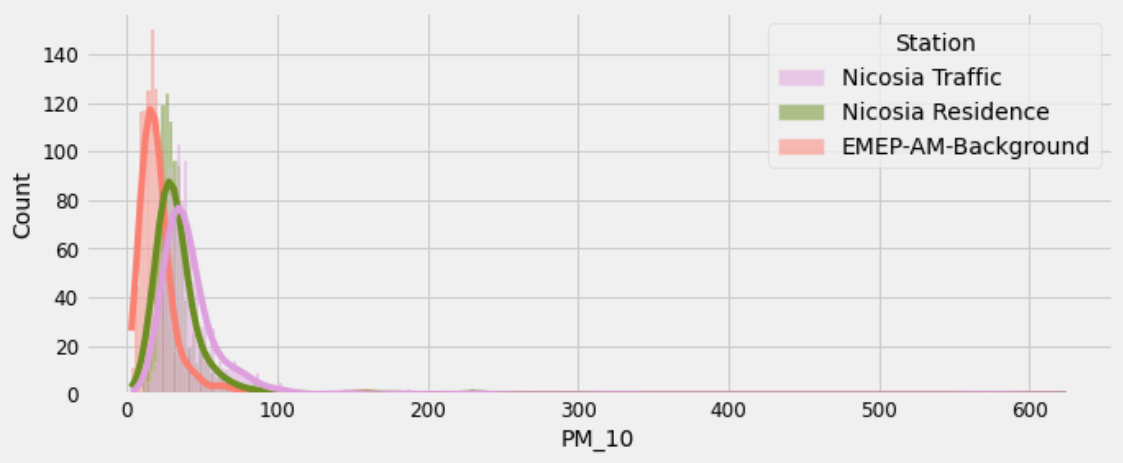
```
plt.show()
```



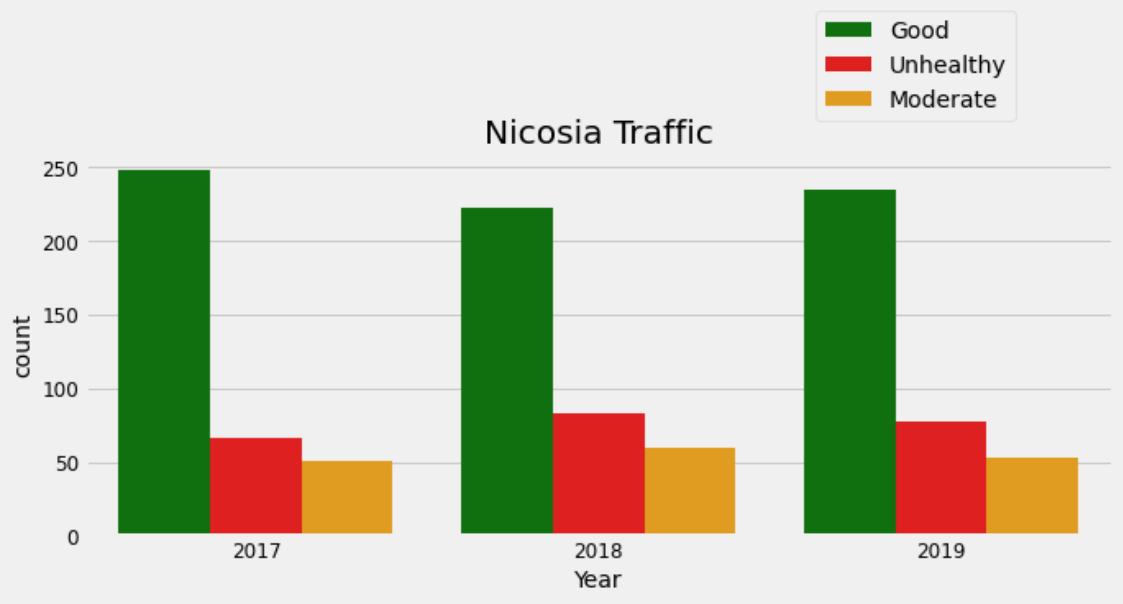
[63]:
```
# importing packages
fig= plt.figure(figsize=(10,4))
sns.histplot(x='PM_2.5', data=result, kde=True,␣
 ↪hue='Station',palette=palette[4])

plt.show()
```



[64]:
```
# importing packages
fig= plt.figure(figsize=(10,4))
sns.histplot(x='PM_10', data=result, kde=True, hue='Station',palette=palette[5])

plt.show()
```

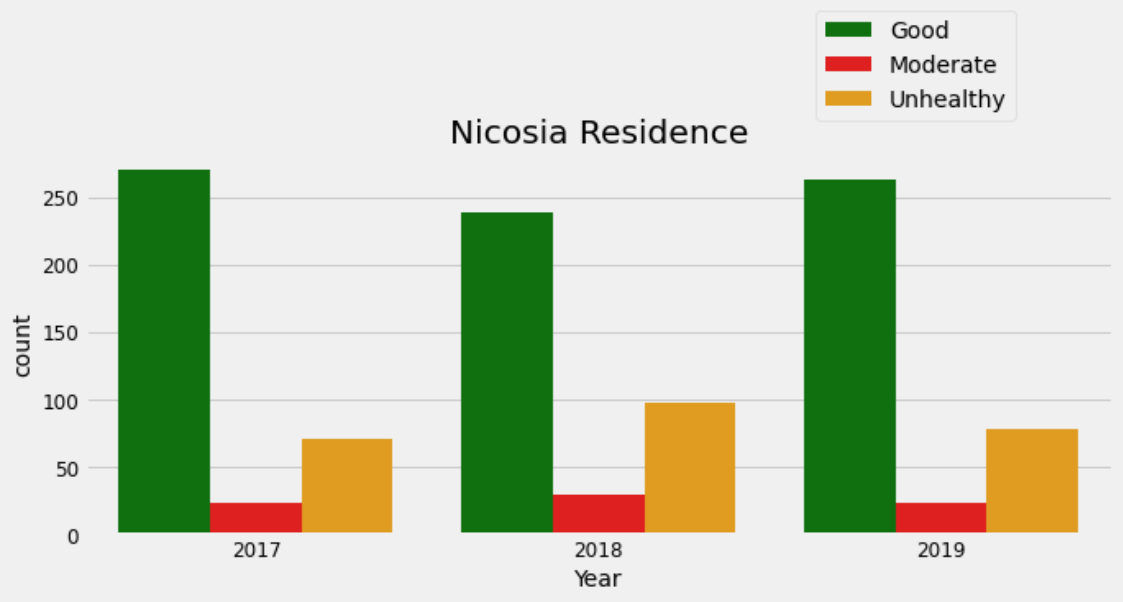


[65]:
```
fig= plt.figure(figsize=(10,4))
sns.countplot(x = 'Year', data = data_NT,hue='Pollution␣
 ↪Level',palette=['green','red','orange'])
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.4))
plt.title('Nicosia Traffic')
plt.show()
```

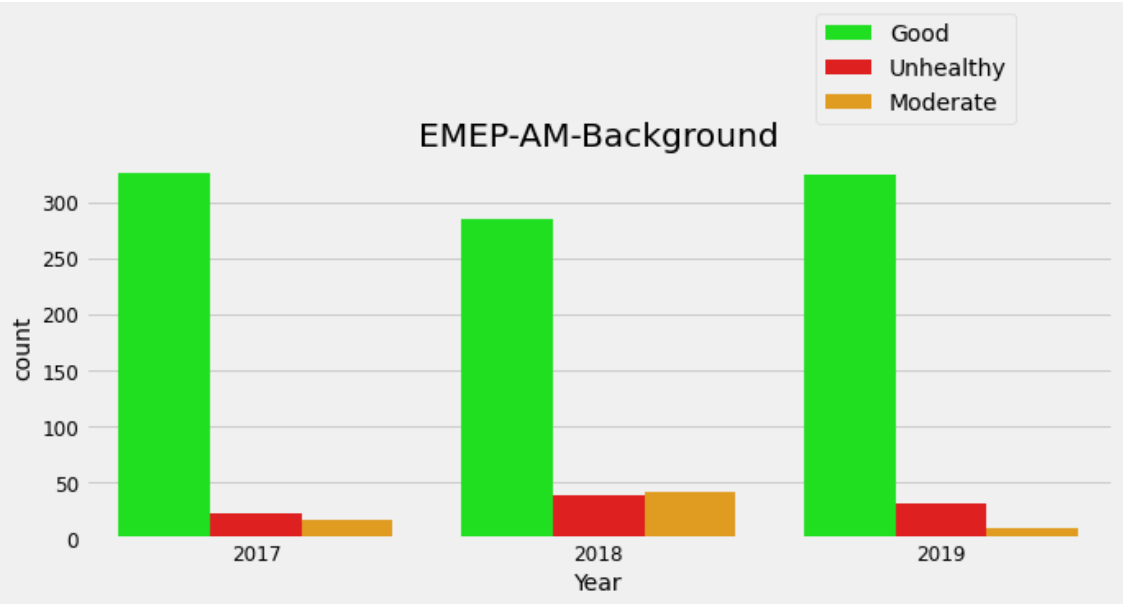


[66]:
```
fig= plt.figure(figsize=(10,4))
sns.countplot(x = 'Year', data = data_NR,hue='Pollution␣
 ↪Level',palette=['green','red','orange'])
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.4))
```

```
plt.title('Nicosia Residence')
plt.show()
```



[67]:
```
fig= plt.figure(figsize=(10,4))
sns.countplot(x = 'Year', data = data_AM,hue='Pollution␣
 ↪Level',palette=['lime','red','orange'])
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.4))
plt.title('EMEP-AM-Background')
plt.show()
```



[68]:
```
import plotly.graph_objects as go
import numpy as np
def scaler(x):
    A=np.array(x)
    A=(A-min(A))/(max(A)-min(A))
    A=list(A)
    return A


NO_2_NT=scaler(data_NT['NO_2'].to_list())
SO_2_NT=scaler(data_NT['SO_2'].to_list())
CO_NT=scaler(data_NT['CO'].to_list())
O_3_NT=scaler(data_NT['O_3'].to_list())
PM_2_5_NT=scaler(data_NT['PM_2.5'].to_list())
PM_10_NT=scaler(data_NT['PM_10'].to_list())


NO_2_NR=scaler(data_NR['NO_2'].to_list())
SO_2_NR=scaler(data_NR['SO_2'].to_list())
CO_NR=scaler(data_NR['CO'].to_list())
O_3_NR=scaler(data_NR['O_3'].to_list())
PM_2_5_NR=scaler(data_NR['PM_2.5'].to_list())
PM_10_NR=scaler(data_NR['PM_10'].to_list())

NO_2_AM=scaler(data_AM['NO_2'].to_list())
SO_2_AM=scaler(data_AM['SO_2'].to_list())
CO_AM=scaler(data_AM['CO'].to_list())
O_3_AM=scaler(data_AM['O_3'].to_list())
PM_2_5_AM=scaler(data_AM['PM_2.5'].to_list())
PM_10_AM=scaler(data_AM['PM_10'].to_list())

L_NT=NO_2_NT+SO_2_NT+CO_NT+O_3_NT+PM_2_5_NT+PM_10_NT
L_NR=NO_2_NR+SO_2_NR+CO_NR+O_3_NR+PM_2_5_NR+PM_10_NR
L_AM=NO_2_AM+SO_2_AM+CO_AM+O_3_AM+PM_2_5_AM+PM_10_AM


x1=['NO_2' for i in range(1095)]
x2=['SO_2' for i in range(1095)]
x3=['CO' for i in range(1095)]
x4=['O_3' for i in range(1095)]
x5=['PM_2.5' for i in range(1095)]
x6=['PM_10' for i in range(1095)]
x=x1+x2+x3+x4+x5+x6

fig = go.Figure()
```

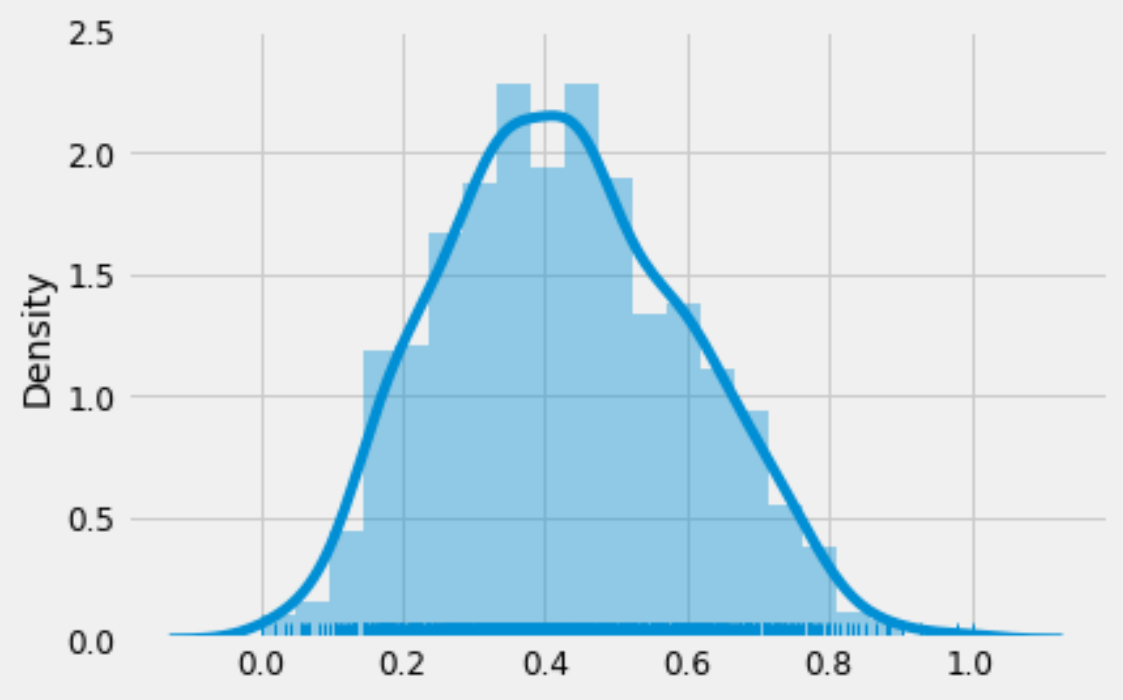
```
# Defining x axis
x = x

fig.add_trace(go.Box(

    # defining y axis in corresponding
    # to x-axis
    y=L_NT,
    x=x,
    name='Nicosia Traffic',
    marker_color='blue'
))

fig.add_trace(go.Box(
    y=L_NR,
    x=x,
    name='Nicosia Residence',
    marker_color='lime'
))

fig.add_trace(go.Box(
    y=L_AM,
    x=x,
    name='EMEP-AM Background',
    marker_color='sienna'

))

fig.update_layout(

    # group together boxes of the different
    # traces for each value of x
    boxmode='group'
)

fig.update_xaxes(title_text="Pollutants")
fig.update_yaxes(title_text="Normalized Concentrations")
fig.show()
```

1.10 Checking distribution of data

[69]:
```
import numpy as np  # numpy used to create data from plotting
import seaborn as sns  # common form of importing seaborn

# Generate normally distributed data
#data = np.random.randn(1000)
```

```
data=O_3_AM

# Plot a histogram with both a rugplot and kde graph superimposed
sns.distplot(data, kde=True, rug=True)
```

[69]: <AxesSubplot:ylabel='Density'>



[70]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import cufflinks as cf
import warnings
import itertools
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
plt.style.use('fivethirtyeight')
import statsmodels.api as sm
import matplotlib
matplotlib.rcParams['axes.labelsize'] = 14
matplotlib.rcParams['xtick.labelsize'] = 12
matplotlib.rcParams['ytick.labelsize'] = 12
matplotlib.rcParams['text.color'] = 'k'
```

```
data1=pd.read_csv("Data_2017.csv")
data2=pd.read_csv("Data_2018.csv")
data3=pd.read_csv("Data_2019.csv")
```

[71]:
```
p_id1=data1['pollutant_id'].to_list()
date1=data1['date_time'].to_list()
p_value1=data1['pollutant_value'].to_list()

p_id2=data2['pollutant_id'].to_list()
date2=data2['date_time'].to_list()
p_value2=data2['pollutant_value'].to_list()

p_id3=data3['pollutant_id'].to_list()
date3=data3['date_time'].to_list()
p_value3=data3['pollutant_value'].to_list()

p_id=p_id1+p_id2+p_id3
date=date1+date2+date3
p_value=p_value1+p_value2+p_value3

date=[date[i] for i in range(len(p_id)) if(p_id[i]==6)]
CO_2017=[p_value[i] for i in range(len(p_id)) if(p_id[i]==6)]

for i in range(len(CO_2017)):
    if (CO_2017[i]=='BDL'):
        CO_2017[i]='nan'
    else:
        pass

for i in range(len(CO_2017)):
    if (CO_2017[i]=='nan'):
        CO_2017[i]='0'
    else:
        pass


#CO_2017=[int(CO_2017[i]) for i in range(len(CO_2017))]
data=[[date[i], CO_2017[i]] for i in range(len(date))]
data= pd.DataFrame(data, columns = ['Date', 'CO'])
data["CO"] = pd.to_numeric(data["CO"], downcast="float")
column_means = data['CO'].mean()
data['CO'].fillna(column_means)

data.shape
```

[71]:
```
(210240, 2)
```

[72]:
```
data1=data
data1['Date'] = pd.to_datetime(data1['Date'])
data1=data1.set_index('Date')
```

[73]:
```
data1['Month']=data1.index.month
data1['Day_of_week']=data1.index.dayofweek
data1['Day']=data1.index.day
data1['Hour']=data1.index.hour
data1['Year']=data1.index.year
data1.head()
data5=data1
data5.tail()
```

[73]:

Date	CO	Month	Day_of_week	Day	Hour	Year
2019-12-31 19:00:00	1665.099976	12	1	31	19	2019
2019-12-31 20:00:00	2269.500000	12	1	31	20	2019
2019-12-31 21:00:00	2996.699951	12	1	31	21	2019
2019-12-31 22:00:00	2517.300049	12	1	31	22	2019
2019-12-31 23:00:00	1723.000000	12	1	31	23	2019

1.11 Determining Skewness and Curtosis of Data

[74]:
```
from scipy.stats import skew,kurtosis
skew(data1['CO'].dropna())
```

[74]:
```
3.7431626319885254
```

[75]:
```
import numpy as np
kurtosis(data1['CO'].dropna())
```

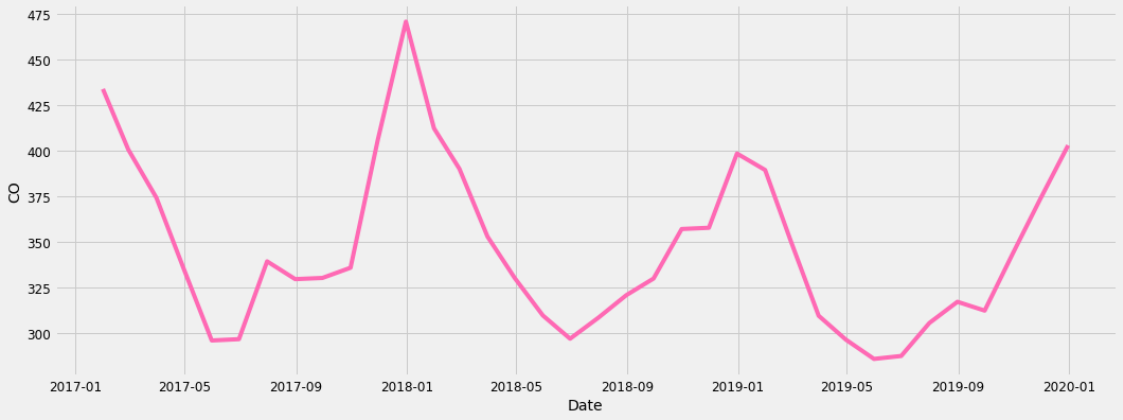
[75]:
```
25.936581574330027
```

[76]:
```
L=np.random.normal(0,1,1000)
```

1.12 Data Visualization

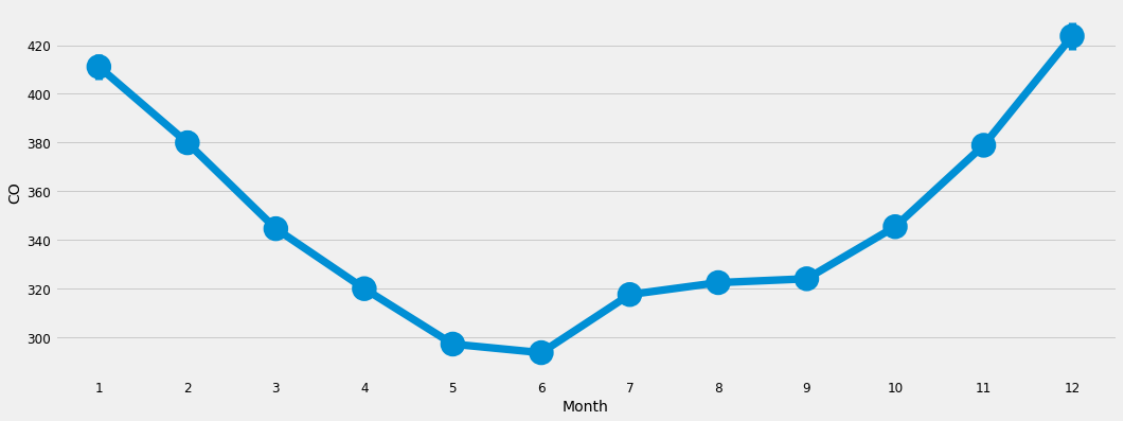
[77]:
```
import matplotlib.pyplot as plt
import seaborn as sns
data_by_month=data1.resample('M').mean()
fig= plt.figure(figsize=(16,6))
#colors =␣
 ↪['lightpink','pink','fuchsia','mistyrose','hotpink','deeppink','magenta']
sns.lineplot(x=data_by_month.index,y='CO',data=data_by_month,color='hotpink')
```

[77]: <AxesSubplot:xlabel='Date', ylabel='CO'>



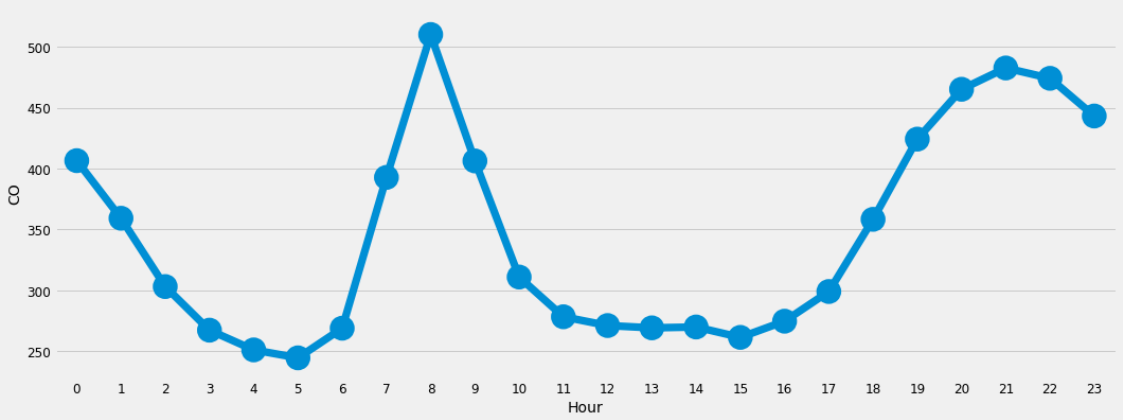
[78]:
```
fig= plt.figure(figsize=(16,6))
sns.pointplot(data=data1,x='Month',y='CO',colr='magenda')
```

[78]: <AxesSubplot:xlabel='Month', ylabel='CO'>



[79]:
```
fig= plt.figure(figsize=(16,6))
sns.pointplot(data=data1,x='Hour',y='CO')
```

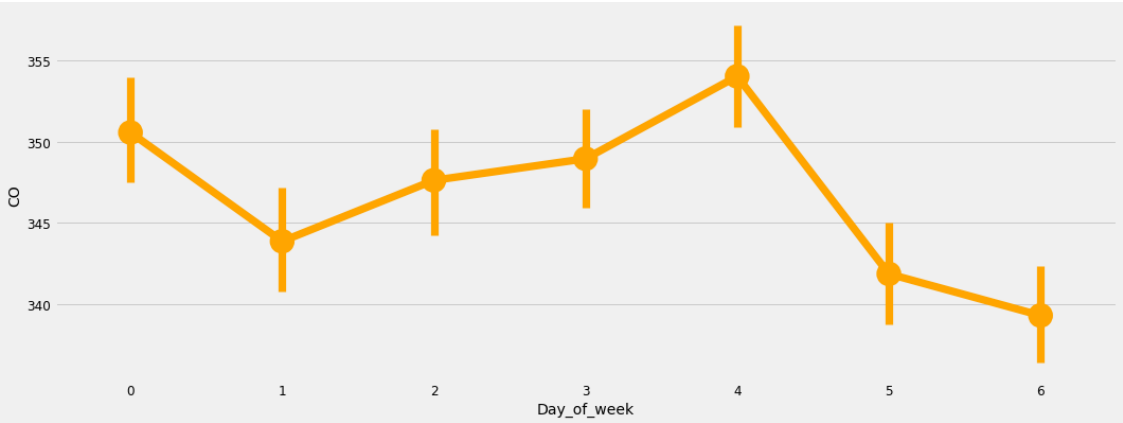
[79]: <AxesSubplot:xlabel='Hour', ylabel='CO'>



As expected, the CO concentration is higher during morning hours and evening hours, attributed to high traffic during those hours.

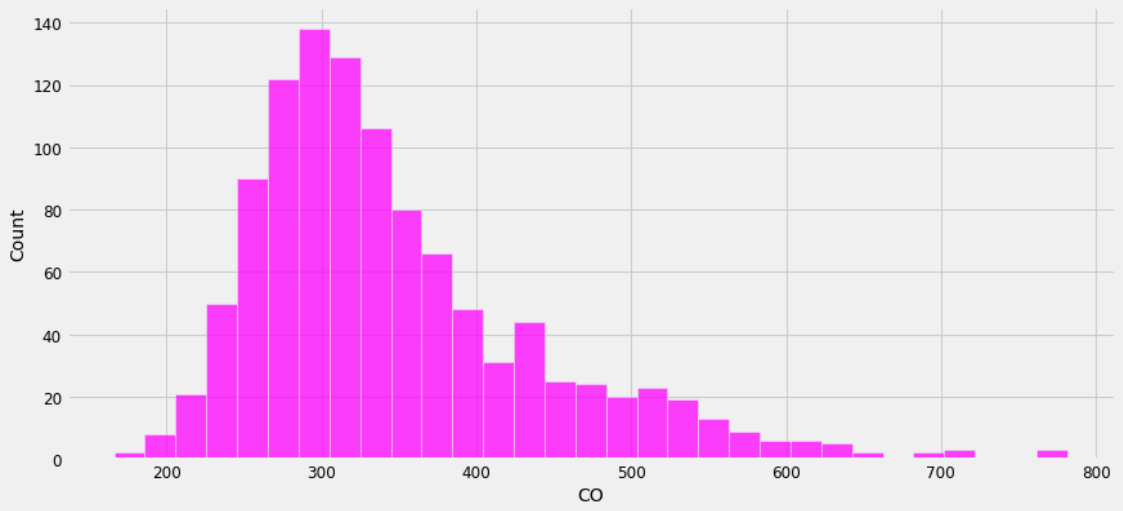
[80]:
```
fig= plt.figure(figsize=(16,6))
sns.pointplot(data=data1,x='Day_of_week',y='CO',color='orange')
```

[80]: <AxesSubplot:xlabel='Day_of_week', ylabel='CO'>



[81]:
```
fig= plt.figure(figsize=(13,6))
data2=data1.resample('D').mean()
sns.histplot(data2, x="CO",color='fuchsia')
```

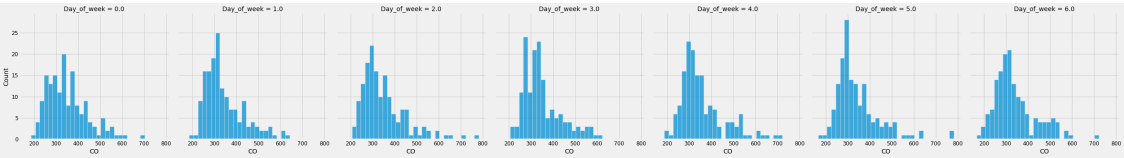
[81]: <AxesSubplot:xlabel='CO', ylabel='Count'>



[82]:
```
fig= plt.figure(figsize=(13,6))
sns.displot(data2, x="CO", col="Day_of_week",multiple="dodge")
```

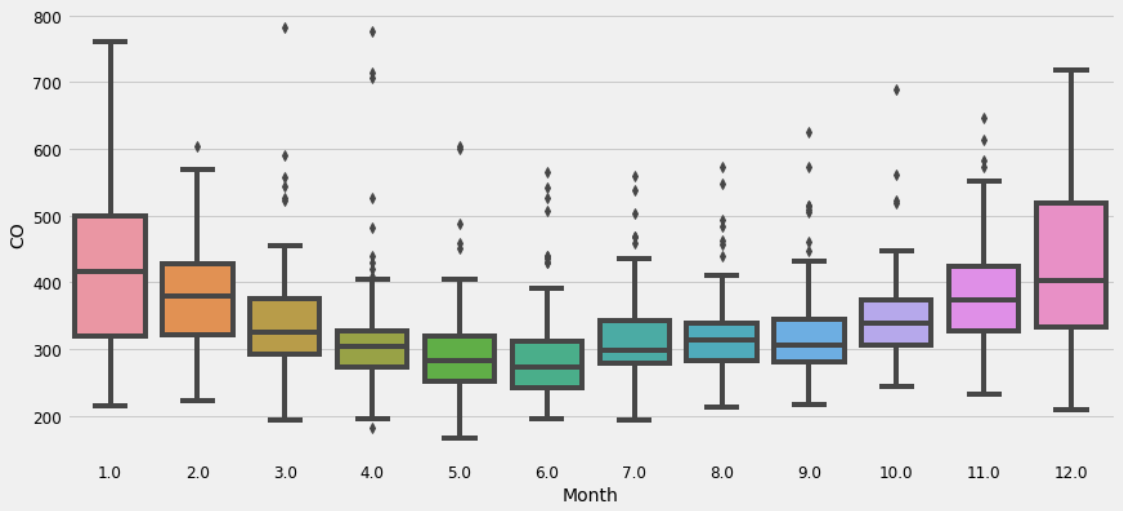
[82]: <seaborn.axisgrid.FacetGrid at 0x1563413a0>

<Figure size 936x432 with 0 Axes>



[83]:
```
fig= plt.figure(figsize=(13,6))
sns.boxplot(data=data2, x='Month', y='CO')
```

[83]: <AxesSubplot:xlabel='Month', ylabel='CO'>

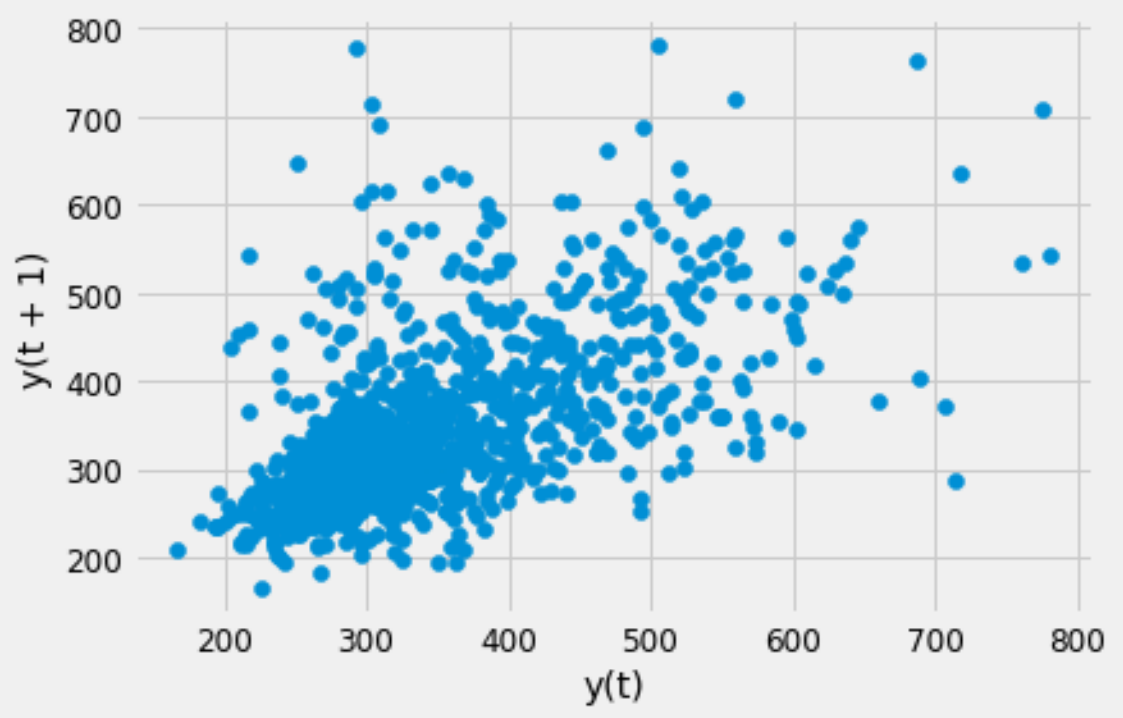


1.13 Time Series Lag Plot

A lag plot helps to check if a time series data set is random or not. A random data will be evenly spread whereas a shape or trend indicates the data is not random.

[84]:
```
pd.plotting.lag_plot(data2['CO'])
```

[84]:
```
<AxesSubplot:xlabel='y(t)', ylabel='y(t + 1)'>
```



[85]:
```
#Printing the date upto period 'Day'
data['Date'] = pd.to_datetime(data['Date']).dt.to_period('H')
#Drops all rows with missing values
#data=data.dropna()
#Setting the date column as an index column to allow resampling
data=data.set_index('Date')
#Resampling data to give only the mean daily CO concentrations
data=data.resample('D').mean()

data['date'] = data.index
#data['date'] = pd.to_datetime(data['date']).dt.to_period('D')
#mean_value=data['CO'].mean()
data=data.fillna(column_means)
data.head()
```

[85]:

Date	CO	date
2017-01-01	470.245972	2017-01-01
2017-01-02	512.894531	2017-01-02
2017-01-03	296.407288	2017-01-03
2017-01-04	302.471497	2017-01-04
2017-01-05	220.845215	2017-01-05

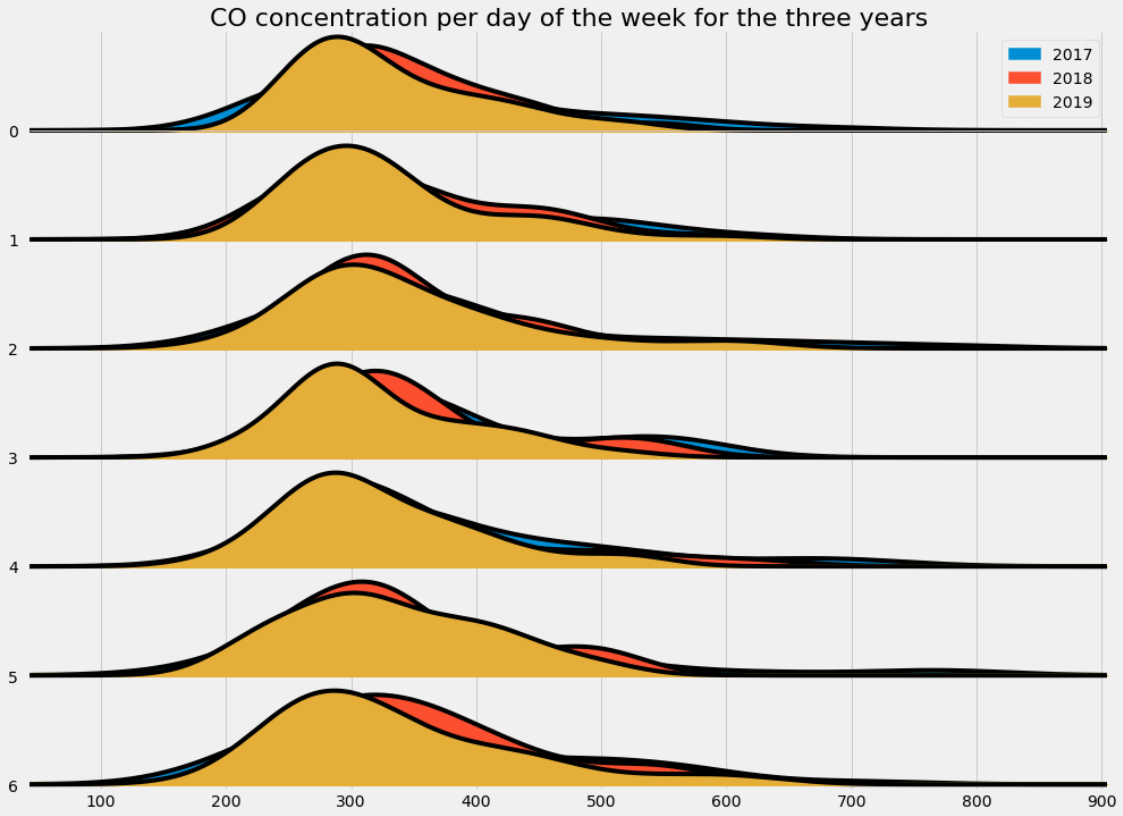
[86]:
```
CO  =data['CO'].to_list()
date=data['date'].to_list()
CO=pd.Series(CO,date)
CO2017=list(CO['2017'])
CO2018=list(CO['2018'])
CO2019=list(CO['2019'])
a_dict = {'Date':CO.index[:365],
    "2017":CO2017,
                  "2018":CO2018,"2019":CO2019}
df = pd.DataFrame(a_dict)
df=df.set_index('Date')
df['Day_of_week']=df.index.dayofweek
df.head()
```

[86]:

Date	2017	2018	2019	Day_of_week
2017-01-01	470.245972	375.883850	415.435516	6
2017-01-02	512.894531	550.797729	300.007111	0
2017-01-03	296.407288	358.899994	319.485382	1
2017-01-04	302.471497	318.951843	316.105865	2
2017-01-05	220.845215	333.370667	265.206879	3

[87]:
```
import joypy
# Draw Plot
plt.figure(figsize=(14,6), dpi= 80)
fig, axes = joypy.joyplot(df, column=['2017', '2018', '2019'],␣
 ↪by="Day_of_week", ylim='own',overlap=0, figsize=(14,10),grid=True,␣
 ↪legend=True)

# Decoration
plt.title('CO concentration per day of the week for the three years',␣
 ↪fontsize=22)
plt.legend(['2017', '2018', '2019'])
plt.show()
```

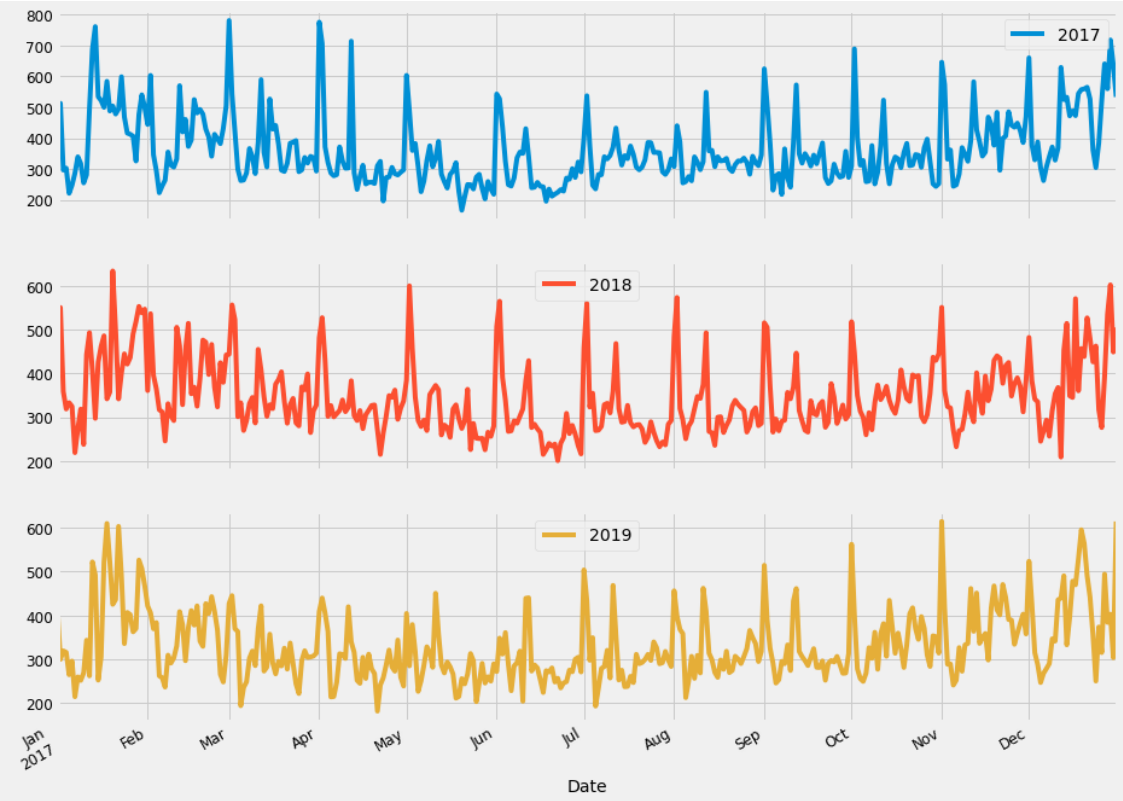
```
<Figure size 1120x480 with 0 Axes>
```



[88]:
```
monthly_totals = df.groupby(df.index.month).mean()
monthly_totals.reset_index(inplace=True)
```

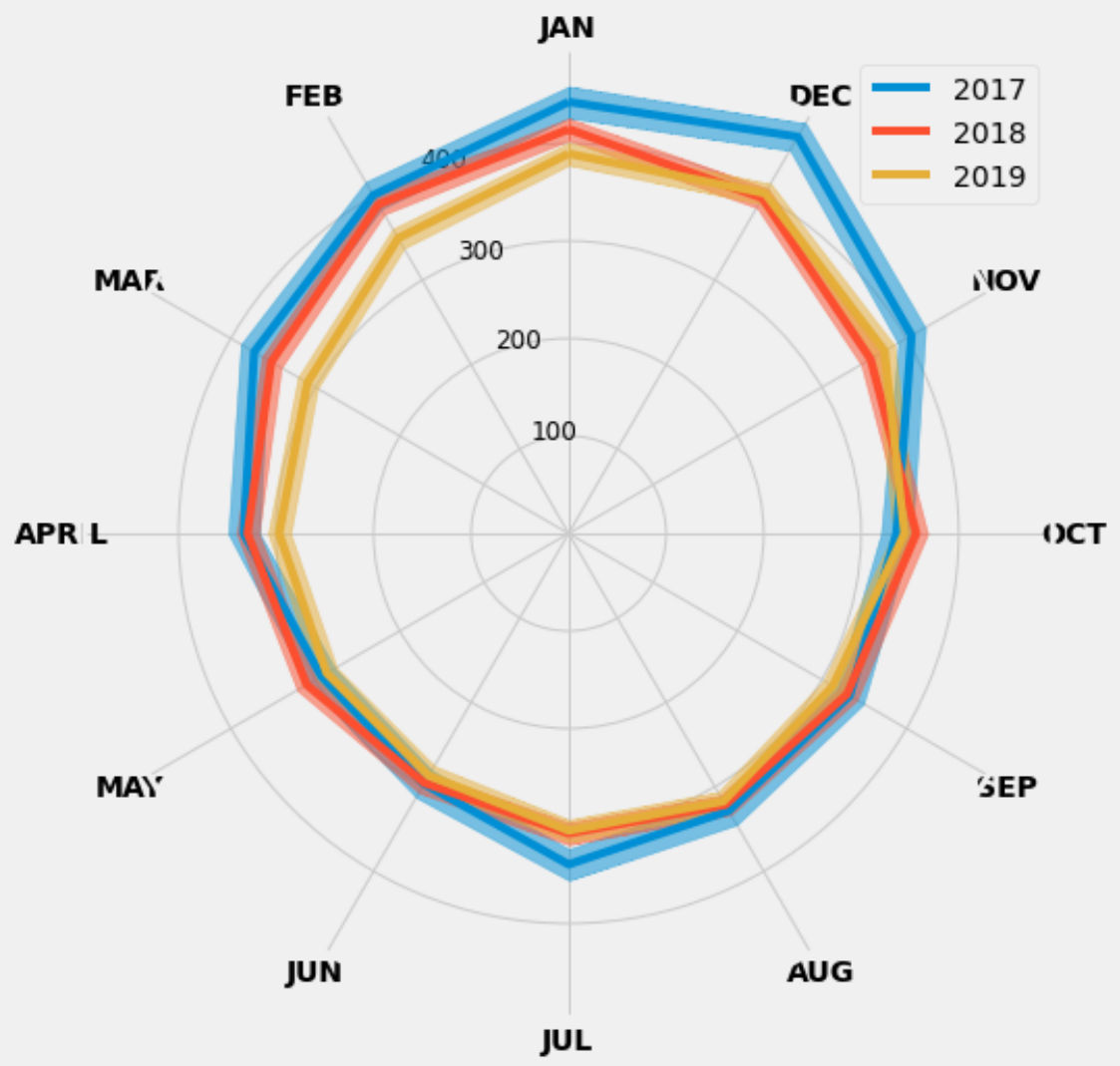
[89]:
```
df.drop(['Day_of_week'], axis=1).plot(subplots=True, figsize=(14,12))
```

[89]:
```
array([<AxesSubplot:xlabel='Date'>, <AxesSubplot:xlabel='Date'>,
       <AxesSubplot:xlabel='Date'>], dtype=object)
```

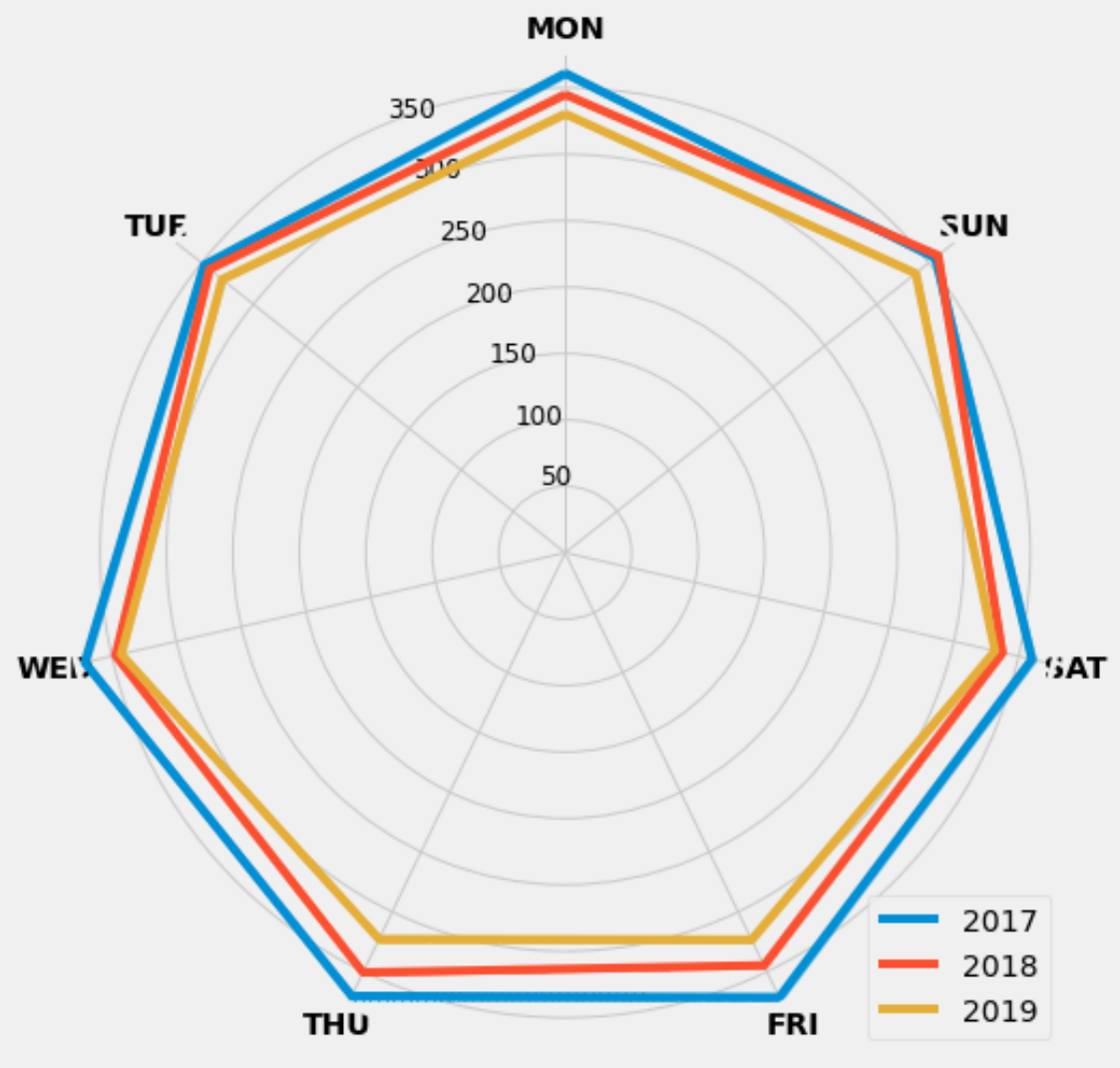


[90]:
```
def plot_polar(df, error_alpha=0.5):
    """Plot DataFrame of day-of-week data as a polar plot with shaded regions
    representing S.E.M.

    DataFrame should be indexed 0-6 with 0=Monday, 6=Sunday
    """
    # add last row to complete cycle (otherwise plot lines don't connect)
    df = df.append(df.iloc[0, :])
    #convert index to radians
    radians = np.linspace(0, 2 * np.pi, num=12, endpoint=False)
    df.index = [radians[day] for day in df.index]
    plt.figure(figsize=(8, 8))
    ax = plt.axes(polar=True)
    ax.set_theta_zero_location('N')
    # Set up labels
    ax.set_xticks(radians)
    ax.set_xticklabels(['JAN', 'FEB', 'MAR', 'APRIL', 'MAY', 'JUN',␣
↪'JUL','AUG','SEP','OCT',
                        'NOV','DEC'], size=14, weight='bold')
```

```
    df.plot(ax=ax, lw=4)
    # need to lookup line color for shaded error region
    line_colors = {line.get_label(): line.get_color() for line in ax.
↪get_lines()}
    def plot_errors(series):
        sem = series.sem()
        ax.fill_between(np.append(radians, 0),
                        series - sem, series + sem,
                        color=line_colors[series.name], alpha=error_alpha)
    df.apply(plot_errors)
    ax.legend( fontsize=14)


plot_polar(monthly_totals.drop(['Date','Day_of_week'], axis=1))
```



[91]:
```
weekly_totals = df.groupby(df.index.dayofweek).mean()
def plot_polar(df, error_alpha=0.5):
    # add last row to complete cycle (otherwise plot lines don't connect)
    df = df.append(df.iloc[0, :])
    #convert index to radians
    radians = np.linspace(0, 2 * np.pi, num=7, endpoint=False)
    df.index = [radians[day] for day in df.index]
    plt.figure(figsize=(8, 8))
    ax = plt.axes(polar=True)
    ax.set_theta_zero_location('N')
    # Set up labels
    ax.set_xticks(radians)
    ax.set_xticklabels(['MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT', 'SUN'],␣
↪size=14, weight='bold')

    df.plot(ax=ax, lw=4)
    # need to lookup line color for shaded error region
    line_colors = {line.get_label(): line.get_color() for line in ax.
↪get_lines()}
    def plot_errors(series):
        sem = series.sem()
        ax.fill_between(np.append(radians, 0),
                        series - sem, series + sem,
                        color=line_colors[series.name], alpha=error_alpha)
    df.apply(plot_errors)
    ax.legend(loc=4, fontsize=14)


plot_polar(weekly_totals.drop(['Day_of_week'], axis=1))
```



[92]:
```
cf.getThemes()
```

[92]:
```
['ggplot', 'pearl', 'solar', 'space', 'white', 'polar', 'henanigans']
```

[93]:
```
cf.go_offline()
weekly_means = df.groupby(df.index.dayofweek).mean()

weekly_means.iplot(kind = 'bar', x = 'Day_of_week', y ='2017')
```

[94]:
```
from plotly.offline import iplot
cf.go_offline()
theme=['polar','pearl','ggplot']
year=['2017','2018','2019']
for i in range(3):
```

```
    cf.set_config_file(theme=theme[i])
    figsize=(2, 1)
    weekly_means.iplot(kind = 'bar', x = 'Day_of_week', y =year[i],␣
↪title=year[i])
```

Checking stationarity of data

To check the stationarity of the TS, we perform two tests. 1. Rolling statistics (rolling mean and rolling std) 2. Dickey Fuller(ADCF) test.

Rolling statistics

[95]:
```
rolmean=data.rolling(window=7).mean()
rolstd=data.rolling(window=7).std()
data1
```

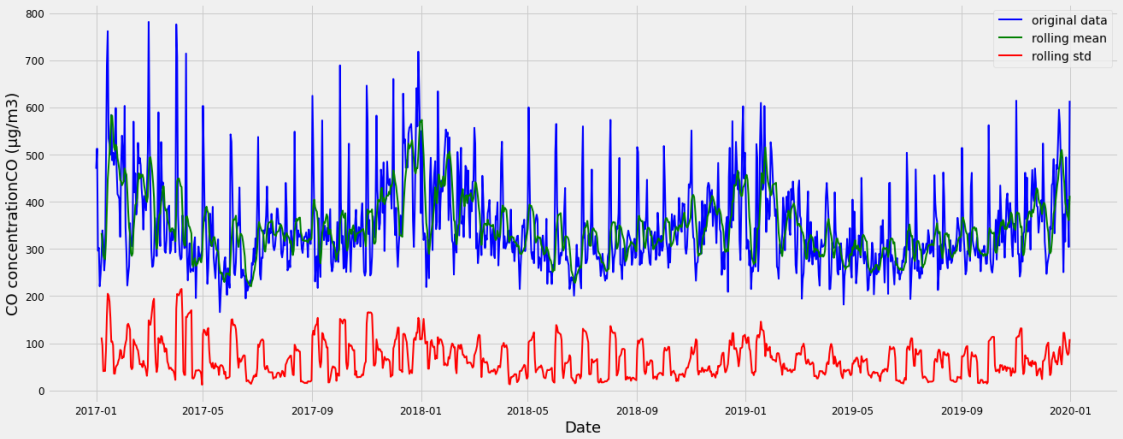
[95]:

Date	CO	Month	Day_of_week	Day	Hour	Year
2017-01-01 00:00:00	1554.890015	1	6	1	0	2017
2017-01-01 01:00:00	1126.829956	1	6	1	1	2017
2017-01-01 02:00:00	2066.790039	1	6	1	2	2017
2017-01-01 03:00:00	1713.380005	1	6	1	3	2017
2017-01-01 04:00:00	1685.310059	1	6	1	4	2017
…	…	…	…	…	…	…
2019-12-31 19:00:00	1665.099976	12	1	31	19	2019
2019-12-31 20:00:00	2269.500000	12	1	31	20	2019
2019-12-31 21:00:00	2996.699951	12	1	31	21	2019
2019-12-31 22:00:00	2517.300049	12	1	31	22	2019
2019-12-31 23:00:00	1723.000000	12	1	31	23	2019

[210240 rows x 6 columns]

[96]:
```
import matplotlib.pyplot as plt
fig= plt.figure(figsize=(20,8))
y=data.iloc[:,0]
y.index = y.index.to_timestamp()
plt.plot(y,color='blue',linewidth=2)
plt.plot(rolmean,color='green',linewidth=2)
plt.plot(rolstd,color='red',linewidth=2)
plt.legend(['original data', 'rolling mean','rolling std'], loc='best')
#plt.scatter(y.index,CO, color='blue')
plt.xlabel('Date', fontsize=18)
plt.ylabel( 'CO concentrationCO ( g/m3)', fontsize=18)
plt.grid(True)
#plt.plot(X3,y1,color='orange',linewidth=3)
#plt.title('Sigmoid and sine function')
#plt.grid(True)
plt.show
```

[96]: <function matplotlib.pyplot.show(close=None, block=None)>



From the graph, the rolling mean and the rolling standard deviation does not depict an upward trend hence stationary.

Dickey Fuller Test

[97]:
```
#Hypothesis
#H_0 : It is not stationary
#H_1 : It is stationary
from statsmodels.tsa.stattools import adfuller
print('Dickey Fuller Test Results:')
dftest=adfuller(data['CO'], autolag='AIC')
dfoutput=pd.Series(dftest[0:4], index=['Test statistic','p-value','#lags␣
↪Used','Number of Observations'])
for key,value in dftest[4].items():
    dfoutput['Critical Value(%s)'%key]=value
print(dfoutput)
```

```
Dickey Fuller Test Results:
Test statistic              -4.330625
p-value                      0.000393
#lags Used                  19.000000
Number of Observations    1075.000000
Critical Value(1%)          -3.436448
Critical Value(5%)          -2.864232
Critical Value(10%)         -2.568204
dtype: float64
```

From the results, the p-value<0.05, we reject the null hyphothesis and conclude that the data is stationary.

[98]:
```
data.head()
```

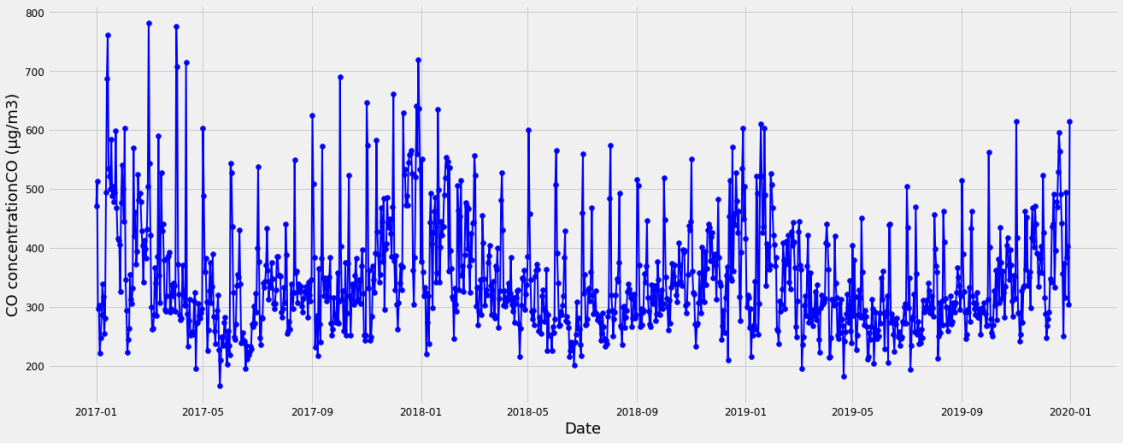
[98]:

Date	CO	date
2017-01-01	470.245972	2017-01-01
2017-01-02	512.894531	2017-01-02
2017-01-03	296.407288	2017-01-03
2017-01-04	302.471497	2017-01-04
2017-01-05	220.845215	2017-01-05

[99]:
```
import math
#Deriving the lists of Date and CO columns of the data
Date=data['date'].to_list()
CO  =data['CO'].to_list()
date=[i+1 for i in range(len(CO)) ]
#CO=[math.log(CO[i]) for i in range(len(CO))]
#Generating timeseries data based on the two lists
data1= pd.Series(CO,Date)
CO=pd.Series(CO)
```

[100]:
```
import matplotlib.pyplot as plt
fig= plt.figure(figsize=(20,8))
y=data1
y.index = y.index.to_timestamp()
plt.plot(y,color='blue',linewidth=2)
plt.scatter(y.index,CO, color='blue')
plt.xlabel('Date', fontsize=18)
plt.ylabel( 'CO concentrationCO ( g/m3)', fontsize=18)
plt.grid(True)
#plt.plot(X3,y1,color='orange',linewidth=3)
#plt.title('Sigmoid and sine function')
#plt.grid(True)
plt.show
```

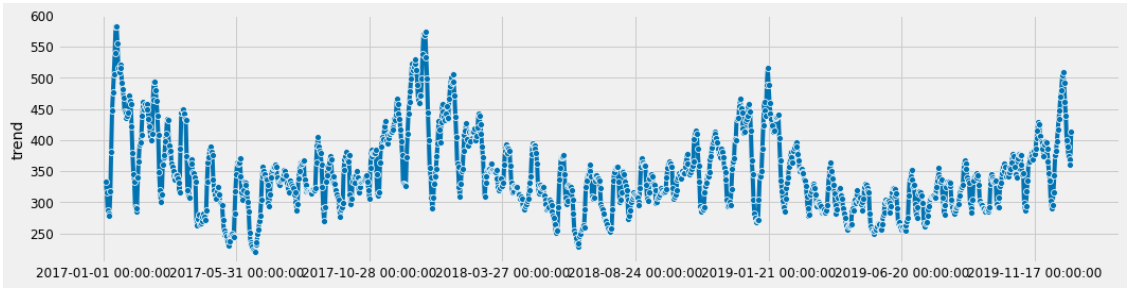
[100]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```

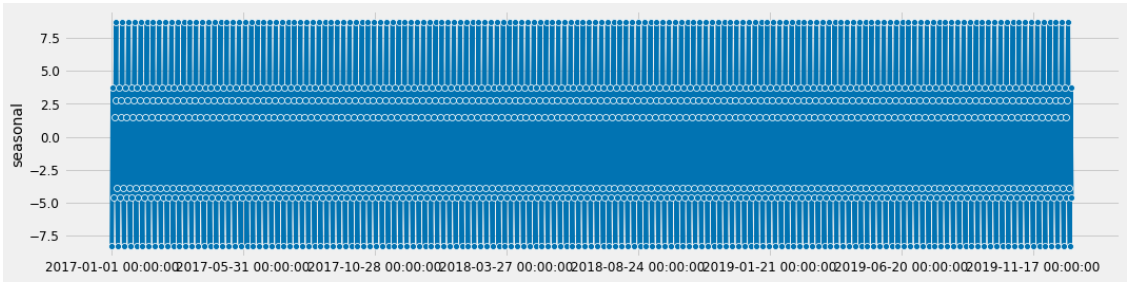


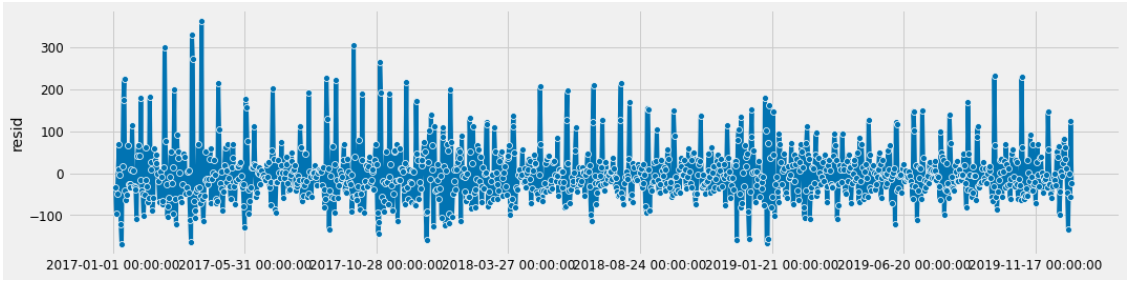
1.14 Decomposing my time series into components

[101]:
```
from sktime.utils.plotting import plot_series
from statsmodels.tsa.seasonal import seasonal_decompose

result = seasonal_decompose(data1, model='additive')
plot_series(result.trend)
plot_series(result.seasonal)
plot_series(result.resid)
```

[101]: (<Figure size 1152x288 with 1 Axes>, <AxesSubplot:ylabel='resid'>)

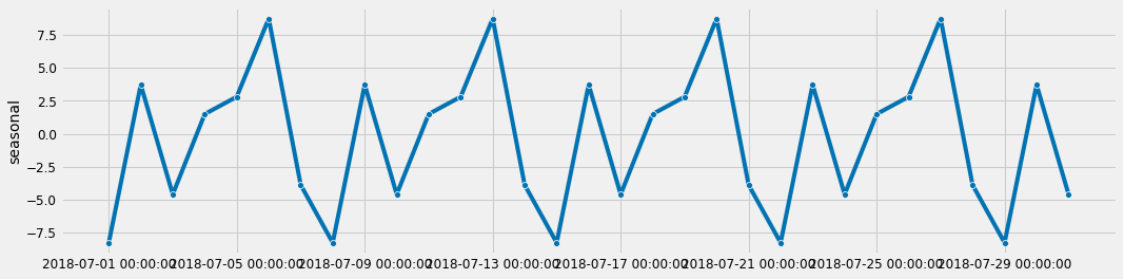


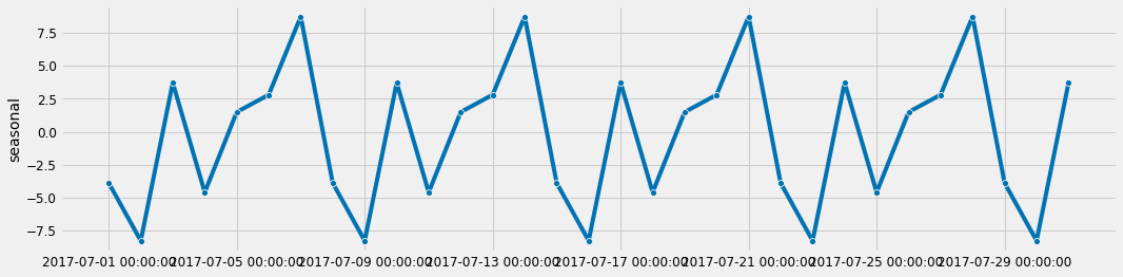




[102]:
```
#Seasonality for 2017 January, we might suspect a weekly seasonality.
plot_series(result.seasonal['2018-07'])
plot_series(result.seasonal['2017-07'])
```

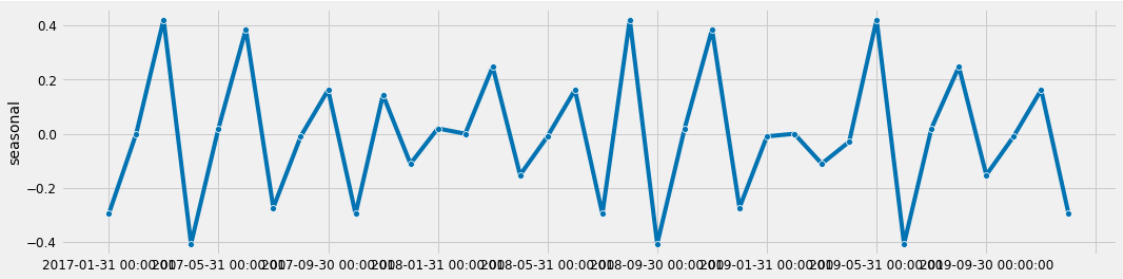
[102]: (<Figure size 1152x288 with 1 Axes>, <AxesSubplot:ylabel='seasonal'>)





[103]:
```
resample = result.seasonal.resample('M')
monthly_mean = resample.mean()
plot_series(monthly_mean)
```

[103]: (<Figure size 1152x288 with 1 Axes>, <AxesSubplot:ylabel='seasonal'>)

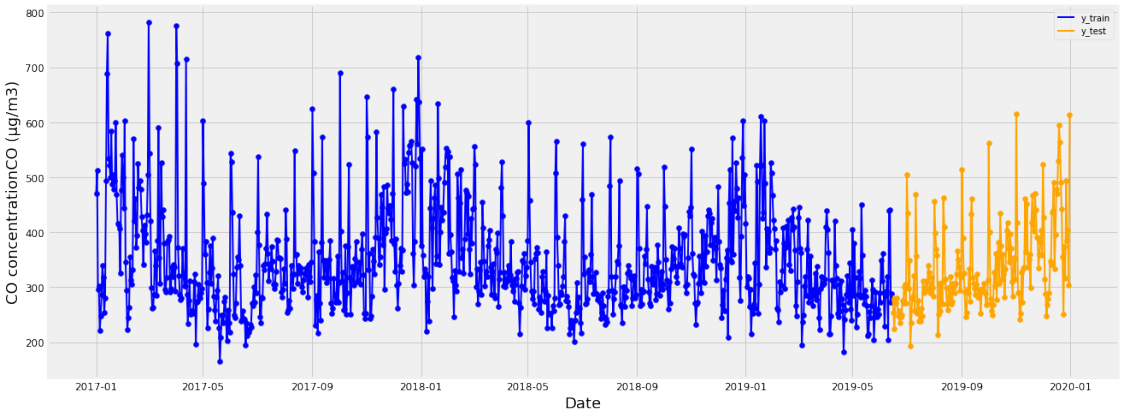


Data is split such that the first 895days is used for training the model and the last 200days for testing the model.

[104]:
```
from warnings import simplefilter

import numpy as np
import pandas as pd

from sktime.datasets import load_airline
from sktime.forecasting.arima import ARIMA, AutoARIMA
from sktime.forecasting.base import ForecastingHorizon
from sktime.forecasting.compose import (
    EnsembleForecaster,
    ReducedRegressionForecaster,
    TransformedTargetForecaster,
)
from sktime.forecasting.exp_smoothing import ExponentialSmoothing
from sktime.forecasting.model_selection import (
    ForecastingGridSearchCV,
    SlidingWindowSplitter,
    temporal_train_test_split,
)
from sktime.forecasting.naive import NaiveForecaster
from sktime.forecasting.theta import ThetaForecaster
from sktime.forecasting.trend import PolynomialTrendForecaster
from sktime.performance_metrics.forecasting import sMAPE, smape_loss
from sktime.transformations.series.detrend import Deseasonalizer, Detrender
from sktime.utils.plotting import plot_series

simplefilter("ignore", FutureWarning)
%matplotlib inline
import matplotlib.pyplot as plt
fig= plt.figure(figsize=(20,8))
ts=200
y_train, y_test = temporal_train_test_split(y, test_size=ts)
y_train_ind, y_test_ind = temporal_train_test_split(y.index, test_size=ts)
CO_train, CO_test = temporal_train_test_split(CO, test_size=ts)
#print(y_train.shape[0], y_test.shape[0])
#x_train=[i+1 for i in range(len(y_train))]
#x_test=[i+1 for i in range(len(y_train),len(y))]
plt.plot(y_train,color='blue',linewidth=2)
plt.scatter(y_train_ind,CO_train, color='blue')
plt.plot(y_test,color='orange',linewidth=2)
plt.scatter(y_test_ind,CO_test, color='orange')
plt.legend(['y_train', 'y_test'], loc='upper right')
plt.xlabel('Date', fontsize=18)
plt.ylabel( 'CO concentrationCO ( g/m3)', fontsize=18)
plt.grid(True)
#plt.plot(X3,y1,color='orange',linewidth=3)
#plt.title('Sigmoid and sine function')
```

```
#plt.grid(True)
plt.show
```

[104]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[105]:
```
##Relative forecasting horizon¶
fh = np.arange(len(y_test)) + 1
#fh
```

[106]:
```
fh = ForecastingHorizon(y_test.index, is_relative=False)
fh
```

[106]:
```
ForecastingHorizon(['2019-06-15', '2019-06-16', '2019-06-17', '2019-06-18',
                    '2019-06-19', '2019-06-20', '2019-06-21', '2019-06-22',
                    '2019-06-23', '2019-06-24',
                    …
                    '2019-12-22', '2019-12-23', '2019-12-24', '2019-12-25',
                    '2019-12-26', '2019-12-27', '2019-12-28', '2019-12-29',
                    '2019-12-30', '2019-12-31'],
                   dtype='datetime64[ns]', length=200, freq='D', is_relative=False)
```

2 Generating forecasts

Like in scikit-learn, in order to make forecasts, we need to first specify (or build) a model, then fit it to the training data, and finally call predict to generate forecasts for the given forecasting horizon.

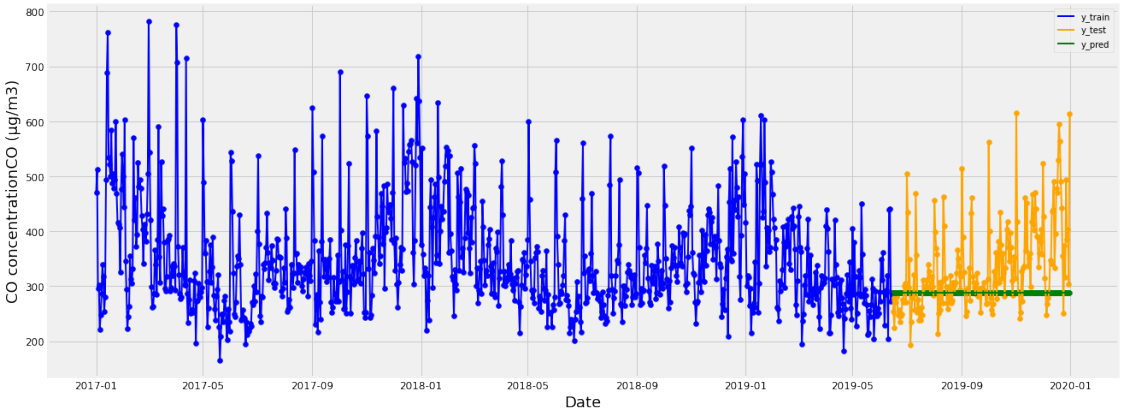
`sktime` comes with several forecasting algorithms (or forecasters) and tools for composite model building. All forecaster share a common interface. Forecasters are trained on a single series of data and make forecasts for the provided forecasting horizon.

2.0.1 Naïve baselines

Let's start with two naïve forecasting strategies which can serve as references for comparison of more sophisticated approaches.

(a) Predicting the last value We use the SMAPE (symmetric mean absolute percentage error) to quantify the accuracy of our forecasts. A lower sMAPE means higher accuracy.

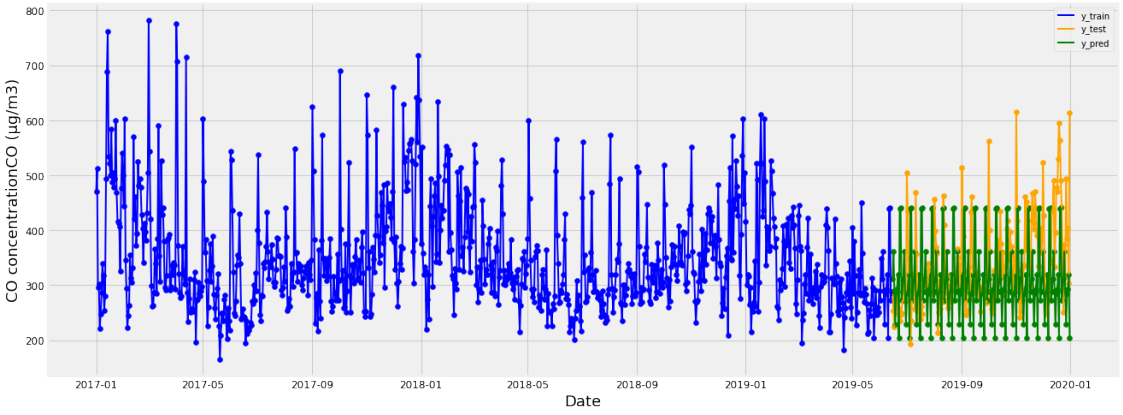
[107]:
```
# using sktime
forecaster = NaiveForecaster(strategy="last")
forecaster.fit(y_train)
y_pred = forecaster.predict(fh)
fig= plt.figure(figsize=(20,8))
y_train, y_test = temporal_train_test_split(y, test_size=ts)
y_train_ind, y_test_ind = temporal_train_test_split(y.index, test_size=ts)
CO_train, CO_test = temporal_train_test_split(CO, test_size=ts)
#print(y_train.shape[0], y_test.shape[0])
#x_train=[i+1 for i in range(len(y_train))]
#x_test=[i+1 for i in range(len(y_train),len(y))]
plt.plot(y_train,color='blue',linewidth=2)
plt.scatter(y_train_ind,CO_train, color='blue')
plt.plot(y_test,color='orange',linewidth=2)
plt.scatter(y_test_ind,CO_test, color='orange')
plt.plot(y_pred,color='green',linewidth=2)
plt.scatter(y_test_ind,y_pred, color='green')
plt.legend(['y_train', 'y_test','y_pred'], loc='upper right')
plt.xlabel('Date',fontsize=18)
plt.ylabel( 'CO concentrationCO ( g/m3)',fontsize=18)
plt.grid(True)
#plt.plot(X3,y1,color='orange',linewidth=3)
#plt.title('Sigmoid and sine function')
#plt.grid(True)
plt.show()
smape_loss(y_pred, y_test)
```



[107]: 0.1866793839428528

(b) Predicting the last value of the same season

[108]:
```
forecaster = NaiveForecaster(strategy="last", sp=12)
forecaster.fit(y_train)
y_pred = forecaster.predict(fh)
fig= plt.figure(figsize=(20,8))
y_train, y_test = temporal_train_test_split(y, test_size=ts)
y_train_ind, y_test_ind = temporal_train_test_split(y.index, test_size=ts)
CO_train, CO_test = temporal_train_test_split(CO, test_size=ts)
#print(y_train.shape[0], y_test.shape[0])
#x_train=[i+1 for i in range(len(y_train))]
#x_test=[i+1 for i in range(len(y_train),len(y))]
plt.plot(y_train,color='blue',linewidth=2)
plt.scatter(y_train_ind,CO_train, color='blue')
plt.plot(y_test,color='orange',linewidth=2)
plt.scatter(y_test_ind,CO_test, color='orange')
plt.plot(y_pred,color='green',linewidth=2)
plt.scatter(y_test_ind,y_pred, color='green')
plt.legend(['y_train', 'y_test','y_pred'], loc='upper right')
plt.xlabel('Date',fontsize=18)
plt.ylabel( 'CO concentrationCO ( g/m3)',fontsize=18)
plt.grid(True)
#plt.plot(X3,y1,color='orange',linewidth=3)
#plt.title('Sigmoid and sine function')
#plt.grid(True)
plt.show()
smape_loss(y_pred, y_test)
```



[108]: 0.24854332375265606

2.1 Forecasting with sktime

2.1.1 Reduction: from forecasting to regression

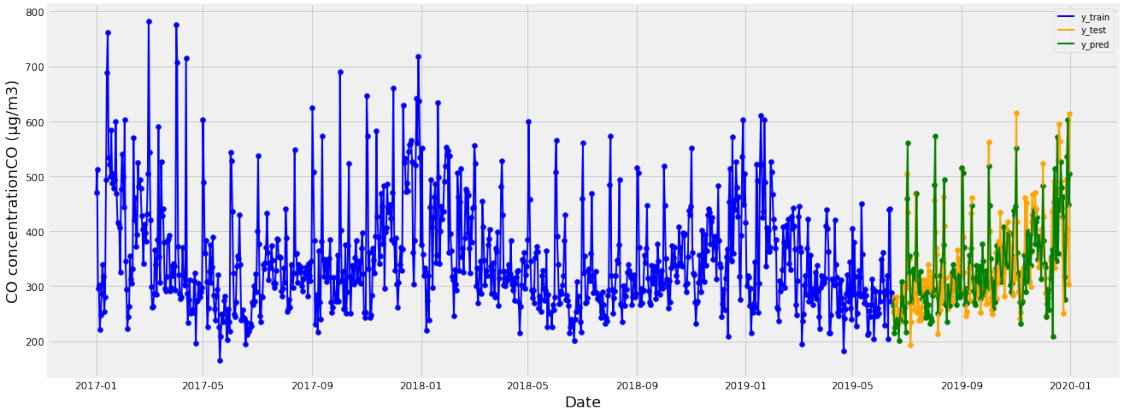
`sktime` provides a meta-estimator for this approach, which is:

- **modular** and **compatible with scikit-learn**, so that we can easily apply any scikit-learn regressor to solve our forecasting problem,
- **tuneable**, allowing us to tune hyper-parameters like the window length or strategy to generate forecasts
- **adaptive**, in the sense that it adapts the scikit-learn's estimator interface to that of a forecaster, making sure that we can tune and properly evaluate our model

2.2 K Neighbors Regressor

[109]:
```
from sklearn.neighbors import KNeighborsRegressor

regressor = KNeighborsRegressor(n_neighbors=1)
forecaster = ReducedRegressionForecaster(
    regressor=regressor, window_length=12, strategy="recursive"
)
forecaster.fit(y_train)
y_pred = forecaster.predict(fh)
fig= plt.figure(figsize=(20,8))
y_train, y_test = temporal_train_test_split(y, test_size=ts)
y_train_ind, y_test_ind = temporal_train_test_split(y.index, test_size=ts)
CO_train, CO_test = temporal_train_test_split(CO, test_size=ts)
#print(y_train.shape[0], y_test.shape[0])
#x_train=[i+1 for i in range(len(y_train))]
#x_test=[i+1 for i in range(len(y_train),len(y))]
plt.plot(y_train,color='blue',linewidth=2)
plt.scatter(y_train_ind,CO_train, color='blue')
plt.plot(y_test,color='orange',linewidth=2)
plt.scatter(y_test_ind,CO_test, color='orange')
plt.plot(y_pred,color='green',linewidth=2)
plt.scatter(y_test_ind,y_pred, color='green')
plt.legend(['y_train', 'y_test','y_pred'], loc='upper right')
plt.xlabel('Date',fontsize=18)
plt.ylabel( 'CO concentrationCO ( g/m3)',fontsize=18)
plt.grid(True)
#plt.plot(X3,y1,color='orange',linewidth=3)
#plt.title('Sigmoid and sine function')
#plt.grid(True)
plt.show()
```

```
smape_loss(y_pred, y_test)
```



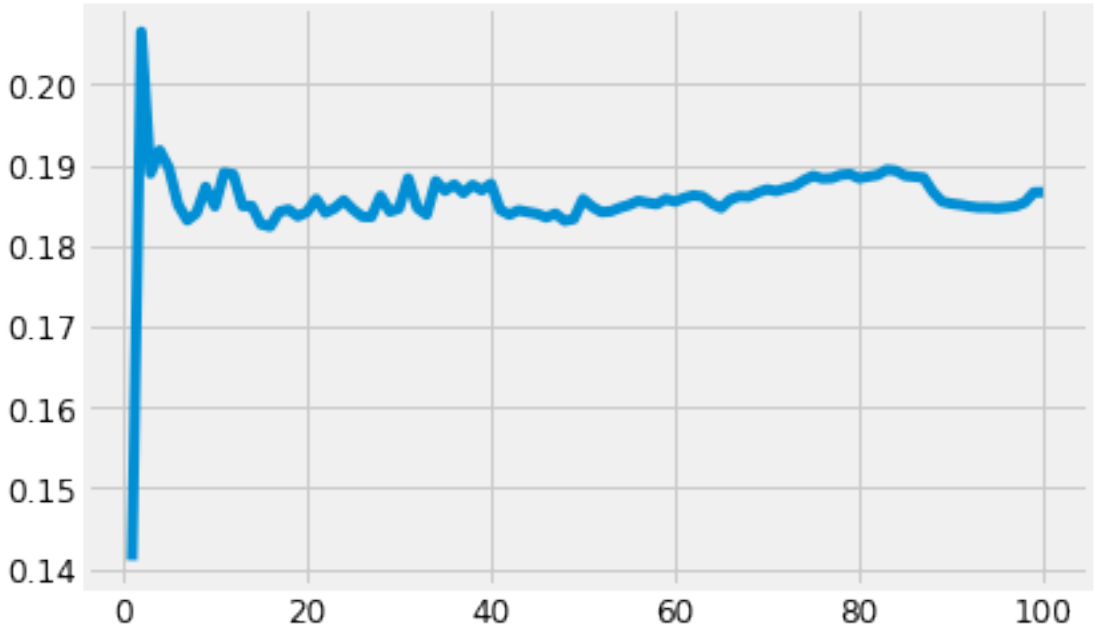
[109]:
```
0.14103414590189012
```

[110]:
```
from sklearn.neighbors import KNeighborsRegressor

regressor = [KNeighborsRegressor(n_neighbors=k) for k in range(1,101)]
forecaster = [ReducedRegressionForecaster(
    regressor=regressor[i], window_length=12, strategy="recursive"
) for i in range(len(regressor))]
model=[forecaster[i].fit(y_train) for i in range(len(regressor))]
y_pred =[ model[i].predict(fh) for i in range(len(regressor))]
loss=[smape_loss(y_pred[i], y_test) for i in range(len(regressor))]
```

3 Exploring different values of k

[111]:
```
index=[k for k in range(1,101)]

plt.plot(index,loss)
```

[111]:
```
[<matplotlib.lines.Line2D at 0x158c19580>]
```



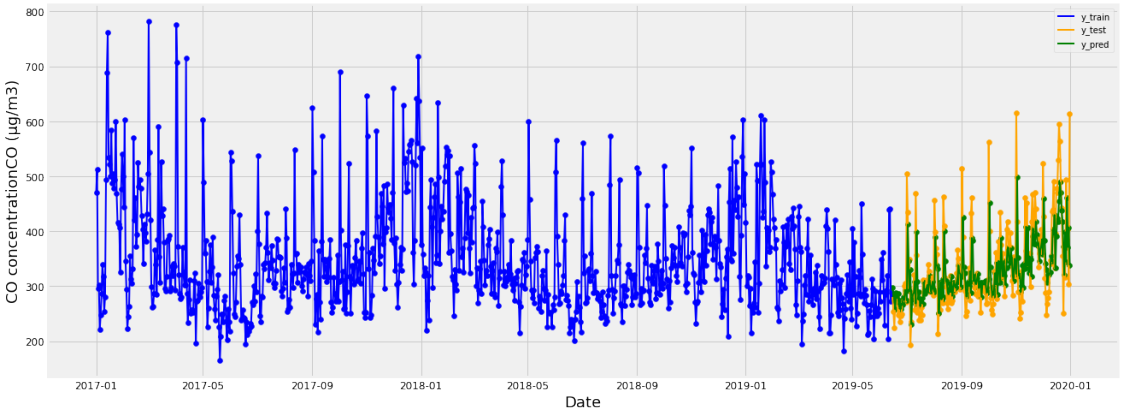
3.0.1 Statistical forecasters

`sktime` has a number of statistical forecasting algorithms, based on implementations in statsmodels. For example, to use exponential smoothing with an additive trend component and multiplicative seasonality, we can write the following.

Note that since this is monthly data, the seasonal periodicity (sp), or the number of periods per year, is 12.

[112]:
```
# ARIMA example
from statsmodels.tsa.arima.model import ARIMA
from random import random
model = ARIMA(y, order=(2, 1, 2))
model_fit = model.fit()
# make prediction
y_pred = model_fit.predict(len(y_train), len(y_train)+199, typ='levels')
fig= plt.figure(figsize=(20,8))

plt.plot(y_train,color='blue',linewidth=2)
plt.scatter(y_train_ind,CO_train, color='blue')
plt.plot(y_test,color='orange',linewidth=2)
plt.scatter(y_test_ind,CO_test, color='orange')
plt.plot(y_pred,color='green',linewidth=2)
plt.scatter(y_test_ind,y_pred, color='green')
plt.legend(['y_train', 'y_test','y_pred'], loc='upper right')
```

```
plt.xlabel('Date',fontsize=18)
plt.ylabel( 'CO concentrationCO ( g/m3)',fontsize=18)
plt.grid(True)
#plt.plot(X3,y1,color='orange',linewidth=3)
#plt.title('Sigmoid and sine function')
#plt.grid(True)
plt.show()
smape_loss(y_pred, y_test)
#model_fit.summary()
```



[112]: 0.1296346599392966

[113]:
```
model_fit.summary(1)
```

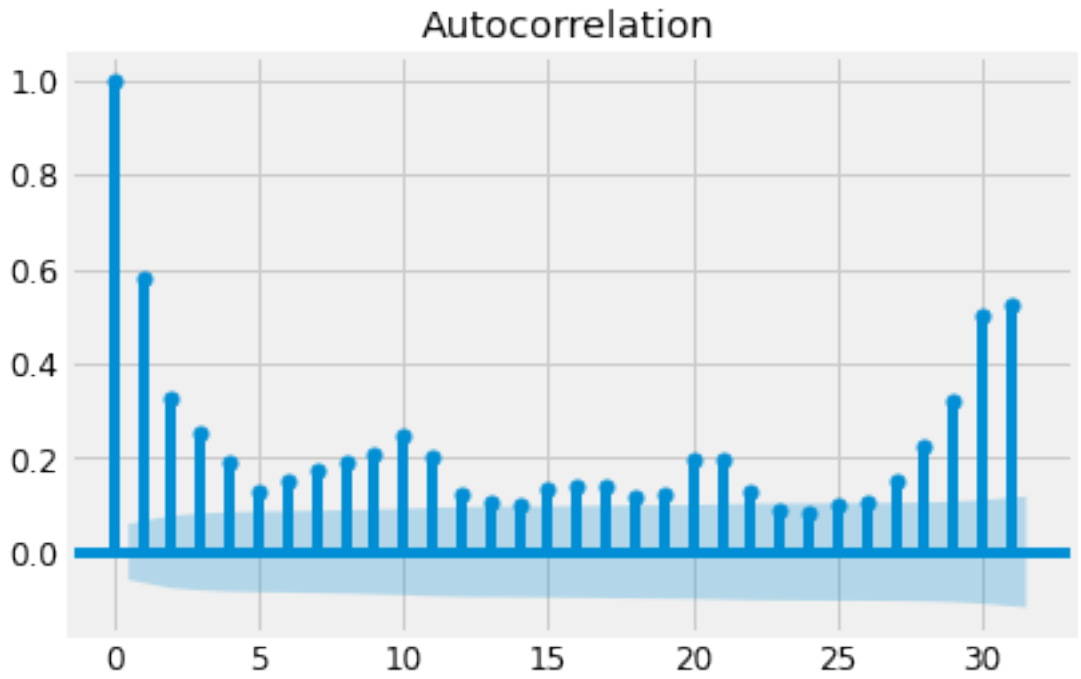
[113]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==========================================================================================
Dep. Variable:                          y   No. Observations:                 1095
Model:                 ARIMA(2, 1, 2)   Log Likelihood               -6276.602
Date:                Sun, 26 Sep 2021   AIC                          12563.203
Time:                        11:07:37   BIC                          12588.191
Sample:                    01-01-2017   HQIC                         12572.659
                         - 12-31-2019
Covariance Type:                  opg
==========================================================================================
                 coef    std err          z      P>|z|        [0.5        0.5]
------------------------------------------------------------------------------------------
ar.L1          0.1494      0.267      0.559      0.576       0.149       0.149
ar.L2          0.0849      0.147      0.576      0.565       0.085       0.085
ma.L1         -0.5720      0.266     -2.151      0.031      -0.572      -0.572
ma.L2         -0.3711      0.257     -1.446      0.148      -0.371      -0.371
```

```
sigma2      5626.4969    144.418     38.960      0.000    5626.497    5626.497
==============================================================================
===
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
1130.75
Prob(Q):                              0.97   Prob(JB):
0.00
Heteroskedasticity (H):               0.56   Skew:
1.32
Prob(H) (two-sided):                  0.00   Kurtosis:
7.22
==============================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
"""
```

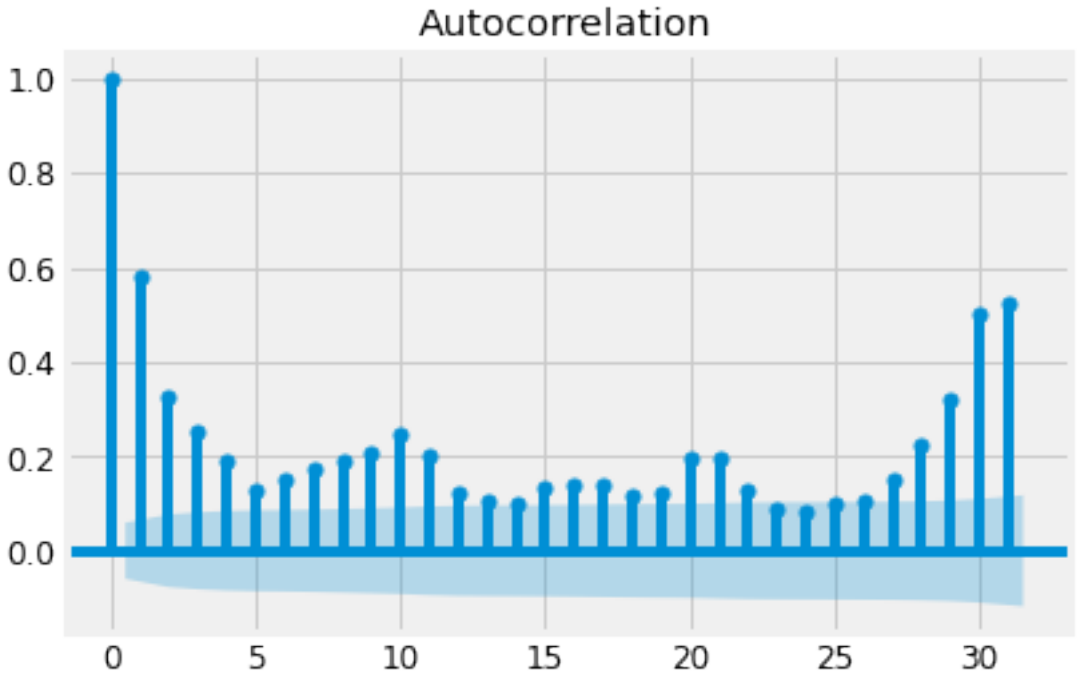
3.1 Calculating number of lags, p and q of ARIMA(p,d,q) using ACF and PACF

[114]:
```
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
fig= plt.figure(figsize=(20,8))
plot_acf(y)
```

[114]:



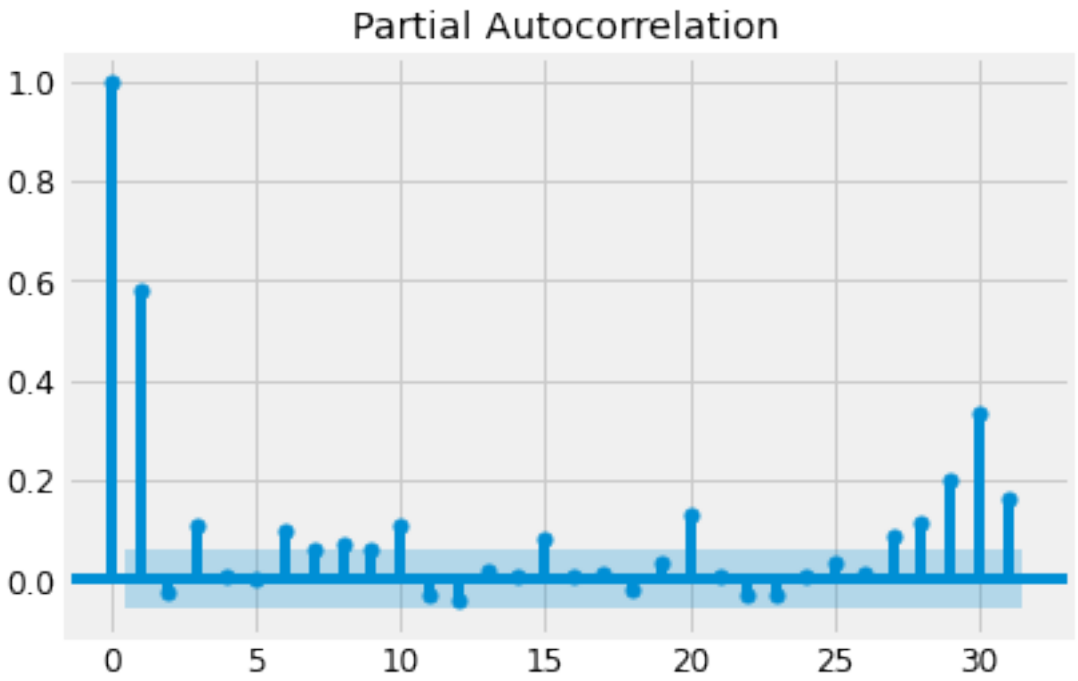
```
<Figure size 1440x576 with 0 Axes>
```

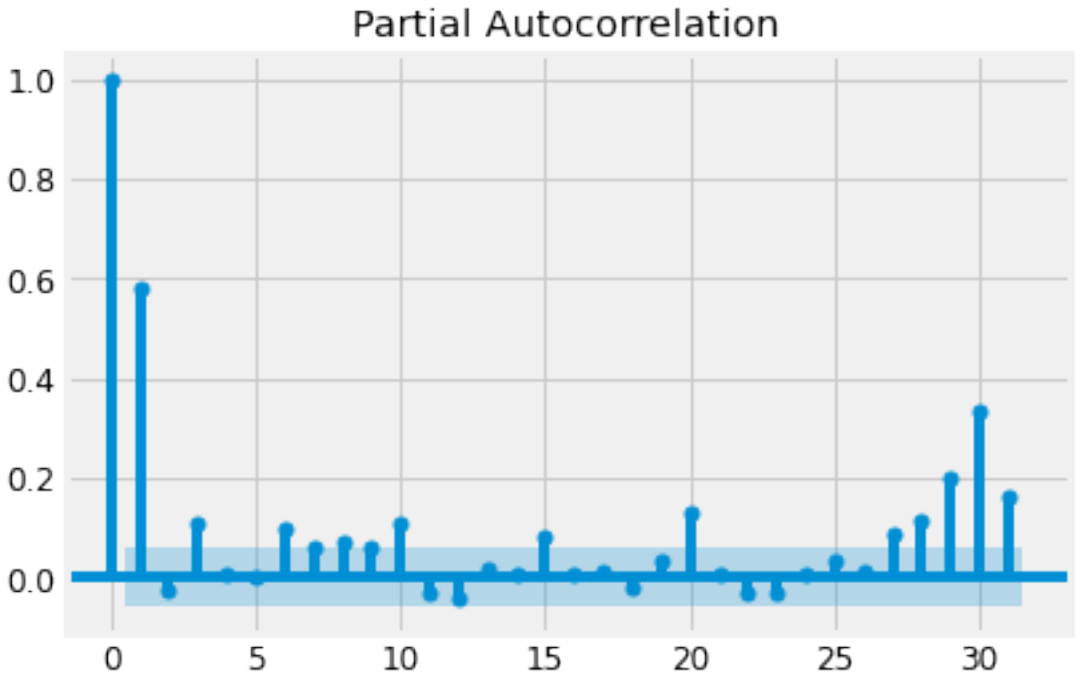


From the ACF plot, the value of q is 5

[115]:
```
plot_pacf(y)
```

[115]:





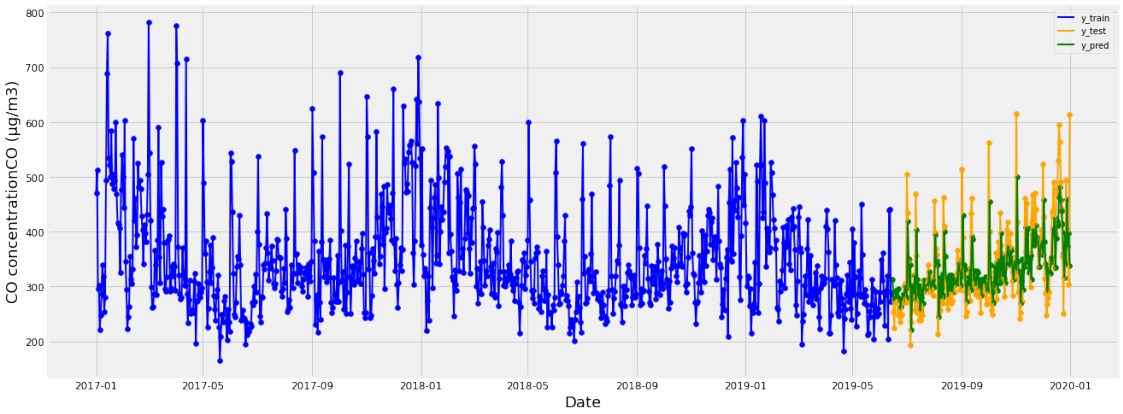
From the PACF, the value of p is 3

[116]:
```
# Determining the value of d
from pmdarima.arima.utils import ndiffs
ndiffs(y,test="adf")
```

[116]: 0

[117]:
```
# Accuracy metrics
def forecast_accuracy(forecast, actual):
    mape = np.mean(np.abs(forecast - actual)/np.abs(actual))  # MAPE
    mae = np.mean(np.abs(forecast - actual))    # MAE
    mpe = np.mean((forecast - actual)/actual)   # MPE
    rmse = np.mean((forecast - actual)**2)**.5  # RMSE

    return( {'mape':mape, 'mae': mae,
            'mpe': mpe, 'rmse':rmse})
```

[118]:
```
# ARMA example
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(y, order=(3, 0, 5))
model_fit = model.fit()
# make prediction
y_pred= model_fit.predict(len(y_train), len(y_train)+199)

fig= plt.figure(figsize=(20,8))

plt.plot(y_train,color='blue',linewidth=2)
plt.scatter(y_train_ind,CO_train, color='blue')
plt.plot(y_test,color='orange',linewidth=2)
plt.scatter(y_test_ind,CO_test, color='orange')
plt.plot(y_pred,color='green',linewidth=2)
plt.scatter(y_test_ind,y_pred, color='green')
plt.legend(['y_train', 'y_test','y_pred'], loc='upper right')
plt.xlabel('Date',fontsize=18)
plt.ylabel( 'CO concentrationCO ( g/m3)',fontsize=18)
plt.grid(True)
#plt.plot(X3,y1,color='orange',linewidth=3)
#plt.title('Sigmoid and sine function')
#plt.grid(True)
plt.show()
smape_loss(y_pred, y_test)
```

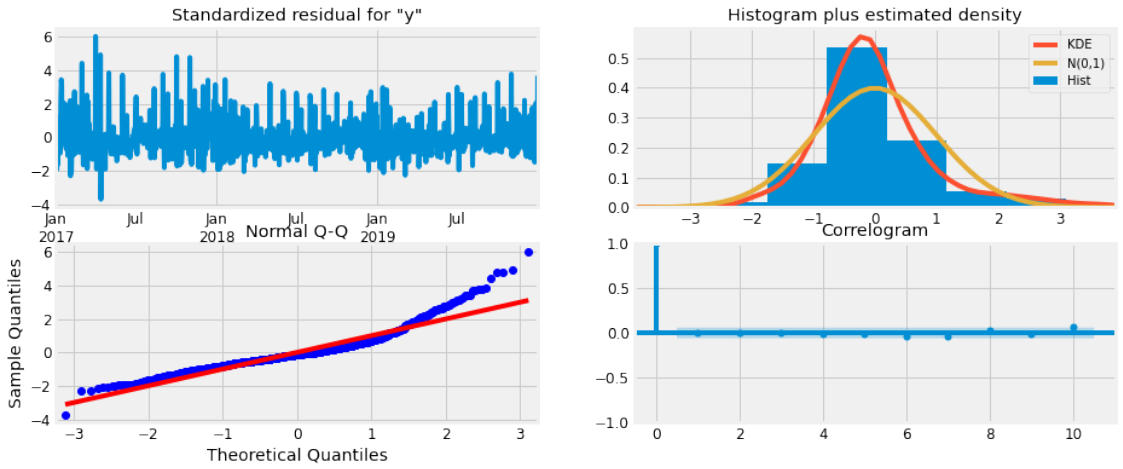


[118]:
```
0.13147110253474364
```

[119]:
```
forecast_accuracy(y_pred, y_test)
```

[119]:
```
{'mape': 0.13079912422978607,
 'mae': 46.110321010027626,
 'mpe': 0.019000126610653284,
 'rmse': 67.21560352529445}
```

[120]:
```
model_fit.plot_diagnostics(figsize=(14,6))
plt.show()
```



More details can be found in here. https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/

3.2 Grid Search

[121]:
```
# ARIMA example
import warnings
warnings.filterwarnings("ignore")
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
from random import random
import itertools
import math
p=range(0,7)
d=range(0,1)
q=range(0,7)
L_model_order=[]
L_loss1=[]
L_loss2=[]
AIC=[]
pdq_comb=list(itertools.product(p,d,q))
for pdq in pdq_comb:
    model = [ARIMA(y, order=pdq ) ]
    model_fit = [model[i].fit() for i in range(len(model))]
    #make prediction
    y_pred = [model_fit[i].predict(len(y_train), len(y_train)+199,␣
 ↪typ='levels') for i in range(len(model)) ]
    loss1=[smape_loss(y_pred[i], y_test)for i in range(len(model)) ]
    loss2=[math.sqrt(mean_squared_error(list(y_pred[i]), list(y_test))) for i␣
 ↪in range(len(model))]
    for i in range(len(model)):
        L_model_order.append(model[i].order)
        L_loss1.append(loss1[i])
        L_loss2.append(loss2[i])
        AIC.append(model_fit[i].aic)
        print(model[i].order,model_fit[i].aic,loss1[i],loss2[i])

for i in range(len(L_loss1)):
    if (L_loss1[i]==min(L_loss1)):
        A=str(L_model_order[i])
        B=str(L_loss1[i])
        C=str(L_loss2[i])
        Model_loss=['\x1b[6;30;43m'+'SMAPE:' + A + B+  '\x1b[0m' for i in␣
 ↪range(5)]
        #A=[print(Happy_New_Year[i]) for i in range(5)]
        B=[print(Model_loss[i]) for i in range(1)]

for i in range(len(L_loss1)):
    if (L_loss2[i]==min(L_loss2)):
        A=str(L_model_order[i])
```

```
        B=str(L_loss1[i])
        C=str(L_loss2[i])
        Model_loss=['\x1b[6;30;46m'+'MSE:' + A +  C+ '\x1b[0m' for i in␣
↪range(5)]
        #A=[print(Happy_New_Year[i]) for i in range(5)]
        B=[print(Model_loss[i]) for i in range(1)]

for i in range(len(L_loss1)):
    if (AIC[i]==min(AIC)):
        A=str(L_model_order[i])
        B=str(L_loss1[i])
        C=str(L_loss2[i])
        D=str(AIC[i])
        Model_loss=['\x1b[6;30;42m'+'AIC:' + A + D+ '\x1b[0m' for i in range(5)]
        #A=[print(Happy_New_Year[i]) for i in range(5)]
        B=[print(Model_loss[i]) for i in range(1)]
```

```
(0, 0, 0) 13067.262587886187 0.18968806034984187 80.70624805571785
(0, 0, 1) 12695.282764865897 0.15732863773163105 72.22545553637582
(0, 0, 2) 12645.335715892492 0.15067636583536312 70.70876801545597
(0, 0, 3) 12629.780549728119 0.14534563239191517 69.2357384840721
(0, 0, 4) 12609.576830500568 0.1422650606721003 69.20673275843556
(0, 0, 5) 12610.289029769407 0.14242842664540334 69.22486419825836
(0, 0, 6) 12610.802998881909 0.14117263648743328 69.1377234591478
(1, 0, 0) 12612.522717168542 0.14075331135314673 68.492870993966
(1, 0, 1) 12613.31919511294 0.1418012863925742 68.81222855126677
(1, 0, 2) 12590.443385156344 0.13426769192812896 68.08407766566246
(1, 0, 3) 12575.801167180012 0.1325504717181964 67.62361118252733
(1, 0, 4) 12574.903242085438 0.13136870058067435 67.2233001674514
(1, 0, 5) 12571.457771835703 0.13120824820384167 67.44124880898607
(1, 0, 6) 12571.486368367368 0.13024632193271538 67.23248899892783
(2, 0, 0) 12613.76698156943 0.14134667955211072 68.68058358250687
(2, 0, 1) 12611.74848224311 0.14203372322000699 68.94261540605183
(2, 0, 2) 12572.253781743995 0.13162988979325368 67.35069108906498
(2, 0, 3) 12573.586341760347 0.13161957233194926 67.366617227206
(2, 0, 4) 12575.165433237462 0.13128564786568536 67.24930473266083
(2, 0, 5) 12571.951350283754 0.12970223906243522 67.23065796187973
(2, 0, 6) 12573.94676823007 0.12972099529338016 67.24339170760004
(3, 0, 0) 12602.50665628726 0.14103440366195283 69.06732224322508
(3, 0, 1) 12604.483785372444 0.1410434813414891 69.0787031720779
(3, 0, 2) 12573.575252988565 0.13149239908417465 67.31846190406225
(3, 0, 3) 12575.827667031579 0.13122812283587534 67.33509380329127
(3, 0, 4) 12576.333028624817 0.13101384793377163 67.38943923855369
(3, 0, 5) 12572.27109448328 0.13147110253474364 67.21560352529445
(3, 0, 6) 12565.43414814473 0.12877657370408158 66.33901303505627
```

```
(4, 0, 0) 12604.487389712625 0.1410426128454656 69.07691569501279
(4, 0, 1) 12605.186163353386 0.14038703072405187 69.04783144221543
(4, 0, 2) 12597.156837324044 0.1402335534653926 68.26352118053765
(4, 0, 3) 12572.107034116656 0.1293394142195895 67.27658712100781
(4, 0, 4) 12573.199640368199 0.12975664716805807 66.62607192208563
(4, 0, 5) 12570.337972448246 0.12849236469477987 66.41357392955524
(4, 0, 6) 12565.123255958395 0.13042812800716175 66.60810995966092
(5, 0, 0) 12606.474550031617 0.14099333354533972 69.06263920439275
(5, 0, 1) 12606.657453903788 0.14016459302204504 69.03868083294613
(5, 0, 2) 12605.264926475431 0.13999993107231676 69.18379586380865
(5, 0, 3) 12572.180383191995 0.12953003295078955 66.3535632024567
(5, 0, 4) 12578.20763968063 0.13286250801515387 67.52368919530194
(5, 0, 5) 12570.179151018394 0.13063645469383509 66.88062409368219
(5, 0, 6) 12566.849515773301 0.13143822783314874 66.76688274463747
(6, 0, 0) 12599.0425237863 0.13927241298744664 68.90675207760208
(6, 0, 1) 12569.049221395102 0.1314235195525035 66.98038585121759
(6, 0, 2) 12570.606655139418 0.1306529124761437 67.05957624322464
(6, 0, 3) 12575.645154219877 0.13088390241790757 67.00712818394021
(6, 0, 4) 12565.606858490073 0.13009287554731483 66.684490858172
(6, 0, 5) 12557.221374565052 0.12982740390338787 65.46884895713187
(6, 0, 6) 12565.42805141899 0.13090220959439233 66.71025883175258
SMAPE:(4, 0, 5)0.12849236469477987
MSE:(6, 0, 5)65.46884895713187
AIC:(6, 0, 5)12557.221374565052
```

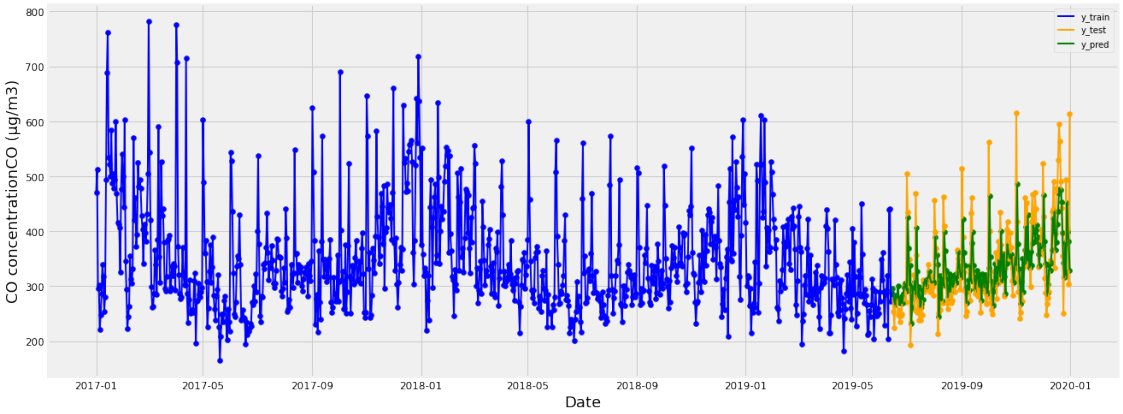
[122]:
```
import pmdarima as pm
model=pm.
 ↪auto_arima(y,m=12,seasonal=False,start_p=0,d=0,start_q=0,max_p=6,max_d=1,max_q=6,␣
 ↪start_P=0,D=0,start_Q=0,max_P=3,max_D=3,max_Q=3,max_order=50,error_action='ignore',scoring=
                )
```

```
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=15996.331, Time=0.03 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=14838.981, Time=0.15 sec
 ARIMA(0,0,2)(0,0,0)[0]             : AIC=14196.746, Time=0.37 sec
 ARIMA(0,0,3)(0,0,0)[0]             : AIC=13877.956, Time=0.90 sec
 ARIMA(0,0,4)(0,0,0)[0]             : AIC=13597.695, Time=0.98 sec
 ARIMA(0,0,5)(0,0,0)[0]             : AIC=13470.813, Time=1.63 sec
 ARIMA(0,0,6)(0,0,0)[0]             : AIC=13343.891, Time=3.40 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=12849.869, Time=0.05 sec
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=12774.746, Time=0.18 sec
 ARIMA(1,0,2)(0,0,0)[0]             : AIC=12597.720, Time=0.46 sec
 ARIMA(1,0,3)(0,0,0)[0]             : AIC=12582.404, Time=0.82 sec
 ARIMA(1,0,4)(0,0,0)[0]             : AIC=12581.396, Time=1.36 sec
 ARIMA(1,0,5)(0,0,0)[0]             : AIC=12577.901, Time=1.74 sec
 ARIMA(1,0,6)(0,0,0)[0]             : AIC=12577.936, Time=2.04 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=12818.381, Time=0.08 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=12588.446, Time=0.60 sec
 ARIMA(2,0,2)(0,0,0)[0]             : AIC=12578.735, Time=0.96 sec
```

```
 ARIMA(2,0,3)(0,0,0)[0]             : AIC=12581.447, Time=1.22 sec
 ARIMA(2,0,4)(0,0,0)[0]             : AIC=12581.710, Time=1.03 sec
 ARIMA(2,0,5)(0,0,0)[0]             : AIC=12578.410, Time=1.91 sec
 ARIMA(2,0,6)(0,0,0)[0]             : AIC=12581.221, Time=2.26 sec
 ARIMA(3,0,0)(0,0,0)[0]             : AIC=12743.101, Time=0.19 sec
 ARIMA(3,0,1)(0,0,0)[0]             : AIC=12580.657, Time=0.97 sec
 ARIMA(3,0,2)(0,0,0)[0]             : AIC=12590.678, Time=1.32 sec
 ARIMA(3,0,3)(0,0,0)[0]             : AIC=12582.353, Time=1.03 sec
 ARIMA(3,0,4)(0,0,0)[0]             : AIC=12595.049, Time=1.76 sec
 ARIMA(3,0,5)(0,0,0)[0]             : AIC=12569.595, Time=2.54 sec
 ARIMA(3,0,6)(0,0,0)[0]             : AIC=12568.532, Time=2.72 sec
 ARIMA(4,0,0)(0,0,0)[0]             : AIC=inf, Time=0.11 sec
 ARIMA(4,0,1)(0,0,0)[0]             : AIC=12581.956, Time=1.23 sec
 ARIMA(4,0,2)(0,0,0)[0]             : AIC=inf, Time=1.56 sec
 ARIMA(4,0,3)(0,0,0)[0]             : AIC=12587.082, Time=1.93 sec
 ARIMA(4,0,4)(0,0,0)[0]             : AIC=inf, Time=2.49 sec
 ARIMA(4,0,5)(0,0,0)[0]             : AIC=12569.855, Time=2.97 sec
 ARIMA(4,0,6)(0,0,0)[0]             : AIC=12570.185, Time=3.70 sec
 ARIMA(5,0,0)(0,0,0)[0]             : AIC=inf, Time=0.18 sec
 ARIMA(5,0,1)(0,0,0)[0]             : AIC=12581.859, Time=1.73 sec
 ARIMA(5,0,2)(0,0,0)[0]             : AIC=inf, Time=2.23 sec
 ARIMA(5,0,3)(0,0,0)[0]             : AIC=inf, Time=2.61 sec
 ARIMA(5,0,4)(0,0,0)[0]             : AIC=inf, Time=2.63 sec
 ARIMA(5,0,5)(0,0,0)[0]             : AIC=12569.249, Time=2.93 sec
 ARIMA(5,0,6)(0,0,0)[0]             : AIC=12574.293, Time=3.18 sec
 ARIMA(6,0,0)(0,0,0)[0]             : AIC=inf, Time=0.23 sec
 ARIMA(6,0,1)(0,0,0)[0]             : AIC=12578.573, Time=2.16 sec
 ARIMA(6,0,2)(0,0,0)[0]             : AIC=inf, Time=3.58 sec
 ARIMA(6,0,3)(0,0,0)[0]             : AIC=12582.906, Time=3.29 sec
 ARIMA(6,0,4)(0,0,0)[0]             : AIC=inf, Time=4.71 sec
 ARIMA(6,0,5)(0,0,0)[0]             : AIC=12568.388, Time=3.78 sec
 ARIMA(6,0,6)(0,0,0)[0]             : AIC=12575.957, Time=4.18 sec

Best model:  ARIMA(6,0,5)(0,0,0)[0]
Total fit time: 84.180 seconds
```

[123]:
```python
# ARMA example
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(y, order=(6, 0, 5))
model_fit = model.fit()
# make prediction
y_pred= model_fit.predict(len(y_train), len(y_train)+199)

fig= plt.figure(figsize=(20,8))

plt.plot(y_train,color='blue',linewidth=2)
plt.scatter(y_train_ind,CO_train, color='blue')
```

```
plt.plot(y_test,color='orange',linewidth=2)
plt.scatter(y_test_ind,CO_test, color='orange')
plt.plot(y_pred,color='green',linewidth=2)
plt.scatter(y_test_ind,y_pred, color='green')
plt.legend(['y_train', 'y_test','y_pred'], loc='upper right')
plt.xlabel('Date',fontsize=18)
plt.ylabel( 'CO concentrationCO ( g/m3)',fontsize=18)
plt.grid(True)
#plt.plot(X3,y1,color='orange',linewidth=3)
#plt.title('Sigmoid and sine function')
#plt.grid(True)
plt.show()
smape_loss(y_pred, y_test)
```

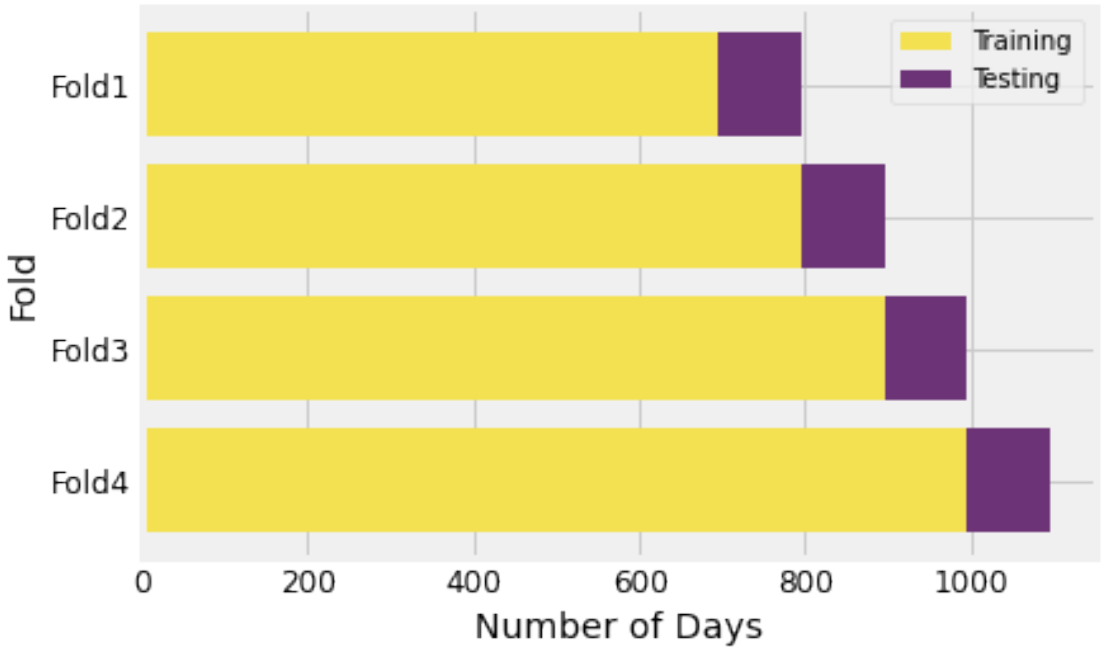


[123]: 0.12982740390338787

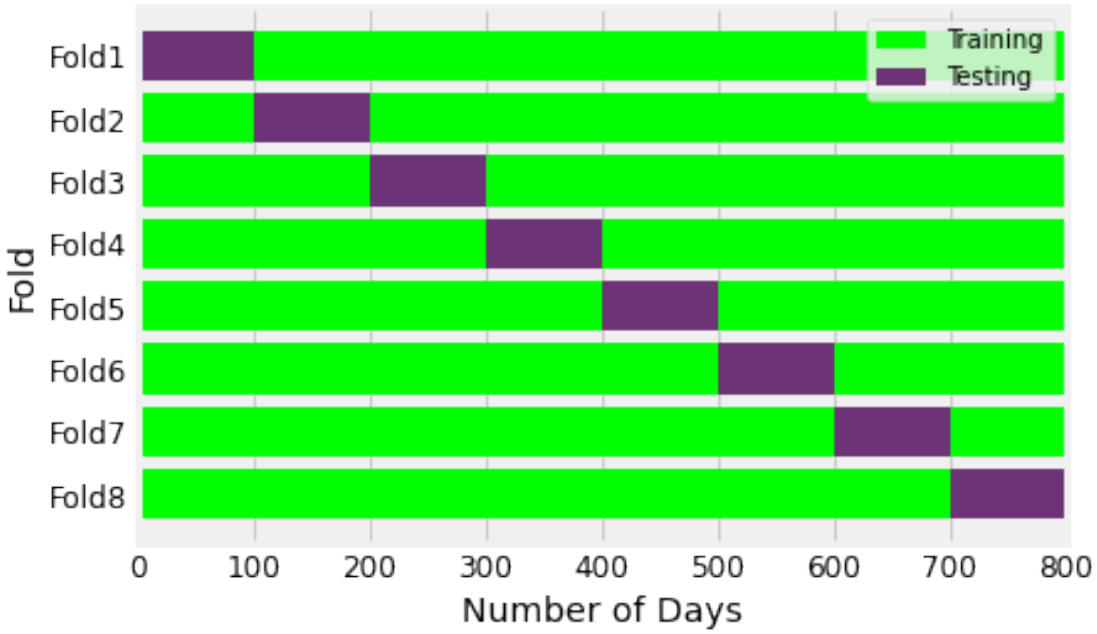
3.3 Rolling Cross-Validation for Time Series

[124]:
```
import matplotlib.pyplot as plt
Fold= ["Fold4","Fold3","Fold2","Fold1"]
Training= [995,895,795,695]
Testing= [100,100,100,100]

plt.barh(Fold,Training, color="#f3e151")
# careful: notice "bottom" parameter became "left"
plt.barh(Fold,Testing, left=Training, color="#6c3376")

# we also need to switch the labels
plt.xlabel('Number of Days')
plt.ylabel('Fold')
plt.legend(['Training', 'Testing'], loc='upper right')
```

```
plt.show()
```



[125]:
```
import matplotlib.pyplot as plt
Fold= ["Fold8","Fold7","Fold6","Fold5","Fold4","Fold3","Fold2","Fold1"]
Training1= [700,600,500,400,300,200,100,0]
Testing= [100,100,100,100,100,100,100,100]
Training2=[0,100,200,300,400,500,600,700]

plt.barh(Fold,Training1, color='lime')
# careful: notice "bottom" parameter became "left"
plt.barh(Fold,Testing, left=Training1, color="#6c3376")
plt.barh(Fold,Training2,left=[i+j for i,j in zip(Training1, Testing)],␣
 ↪color='lime')
# we also need to switch the labels
plt.xlabel('Number of Days')
plt.ylabel('Fold')
plt.legend(['Training', 'Testing'], loc='upper right')
plt.show()
```



[126]:
```
def Cross_valid(data):
    fold1_train=data[0:695]
    fold1_test=data[695:795]
    fold2_train=data[0:795]
    fold2_test=data[795:895]
    fold3_train=data[0:895]
    fold3_test=data[895:995]
    fold4_train=data[0:995]
    fold4_test=data[995:1095]
    train=[fold1_train,fold2_train,fold3_train,fold4_train]
    test= [fold1_test,fold2_test,fold3_test,fold4_test]
    return train,test
```

[127]:
```
Cross_valid(y)[0][0]
```

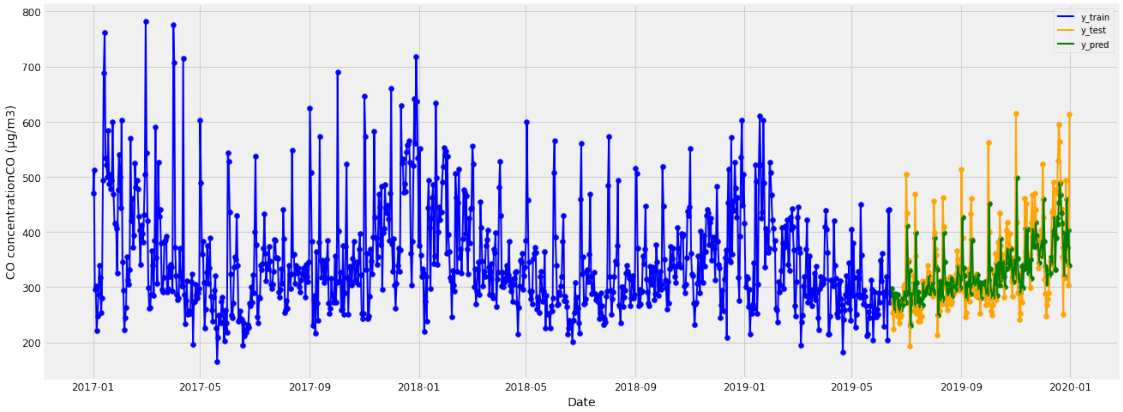
[127]:
```
2017-01-01    470.245972
2017-01-02    512.894531
2017-01-03    296.407288
2017-01-04    302.471497
2017-01-05    220.845215
                  ...
2018-11-22    377.546753
2018-11-23    418.029480
2018-11-24    425.210388
2018-11-25    348.479065
2018-11-26    376.163483
```

```
Freq: D, Length: 695, dtype: float64
```

[128]:
```
#  Cross Validation ARIMA example
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(y, order=(6, 0, 5))
model_fit = model.fit()
# make prediction
y_pred_list= [model_fit.predict(len(Cross_valid(y)[0][i]),␣
 ↪len(Cross_valid(y)[0][i])+99) for i in range(4)]
y_test_list=Cross_valid(y)[1]
Smape_loss=[smape_loss(y_pred_list[i], y_test_list[i]) for i in range(4)]
Smape_loss
```

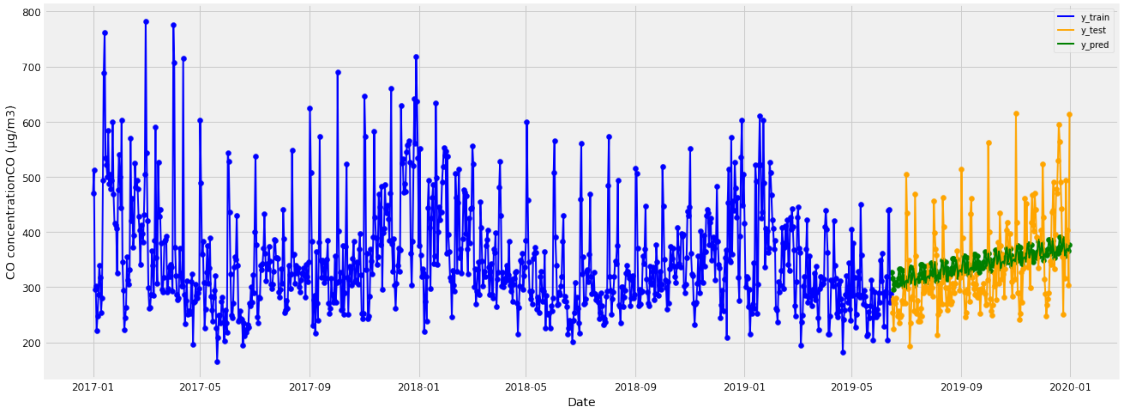
[128]:
```
[0.1680243586490672,
 0.1363483628549973,
 0.11524571567115707,
 0.14440909213561873]
```

[129]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX
# fit model
model = SARIMAX(y, order=(2, 0, 2), seasonal_order=(0, 0, 0, 12))
model_fit = model.fit(disp=False)
# make prediction
y_pred = model_fit.predict(len(y_train), len(y_train)+199)
fig= plt.figure(figsize=(20,8))

plt.plot(y_train,color='blue',linewidth=2)
plt.scatter(y_train_ind,CO_train, color='blue')
plt.plot(y_test,color='orange',linewidth=2)
plt.scatter(y_test_ind,CO_test, color='orange')
plt.plot(y_pred,color='green',linewidth=2)
plt.scatter(y_test_ind,y_pred, color='green')
plt.legend(['y_train', 'y_test','y_pred'], loc='upper right')
plt.xlabel('Date')
plt.ylabel( 'CO concentrationCO ( g/m3)')
plt.grid(True)
#plt.plot(X3,y1,color='orange',linewidth=3)
#plt.title('Sigmoid and sine function')
#plt.grid(True)
plt.show()
smape_loss(y_pred,y_test)
```



[129]: 0.130013804664028

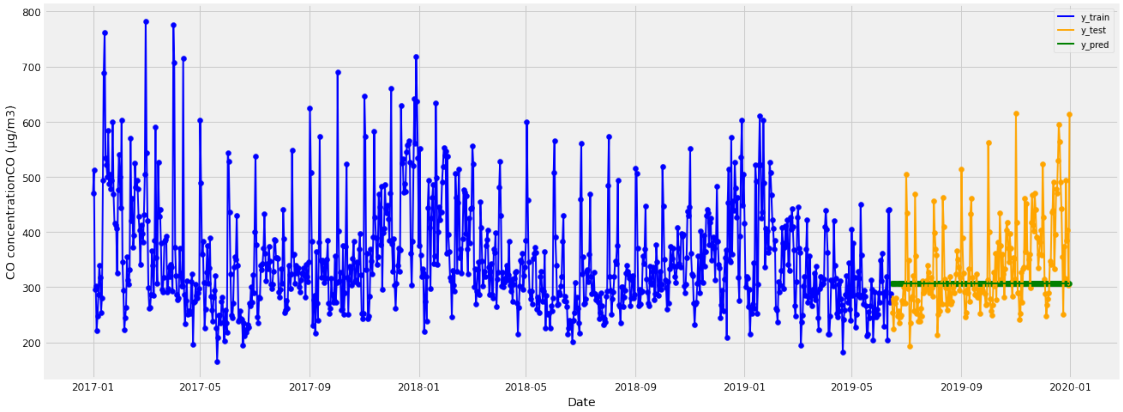
[130]:
```
forecaster = ExponentialSmoothing(trend="add", seasonal='add', sp=12)
forecaster.fit(y_train)
y_pred = forecaster.predict(fh)
fig= plt.figure(figsize=(20,8))
y_train, y_test = temporal_train_test_split(y, test_size=ts)
y_train_ind, y_test_ind = temporal_train_test_split(y.index, test_size=ts)
CO_train, CO_test = temporal_train_test_split(CO, test_size=ts)
#print(y_train.shape[0], y_test.shape[0])
#x_train=[i+1 for i in range(len(y_train))]
#x_test=[i+1 for i in range(len(y_train),len(y))]
plt.plot(y_train,color='blue',linewidth=2)
plt.scatter(y_train_ind,CO_train, color='blue')
plt.plot(y_test,color='orange',linewidth=2)
plt.scatter(y_test_ind,CO_test, color='orange')
plt.plot(y_pred,color='green',linewidth=2)
plt.scatter(y_test_ind,y_pred, color='green')
plt.legend(['y_train', 'y_test','y_pred'], loc='upper right')
plt.xlabel('Date')
plt.ylabel( 'CO concentrationCO ( g/m3)')
plt.grid(True)
#plt.plot(X3,y1,color='orange',linewidth=3)
#plt.title('Sigmoid and sine function')
#plt.grid(True)
plt.show()
smape_loss(y_pred, y_test)
```



[130]: 0.17008622834132442

The exponential smoothing of state space model can also be automated similar to the ets function in R.

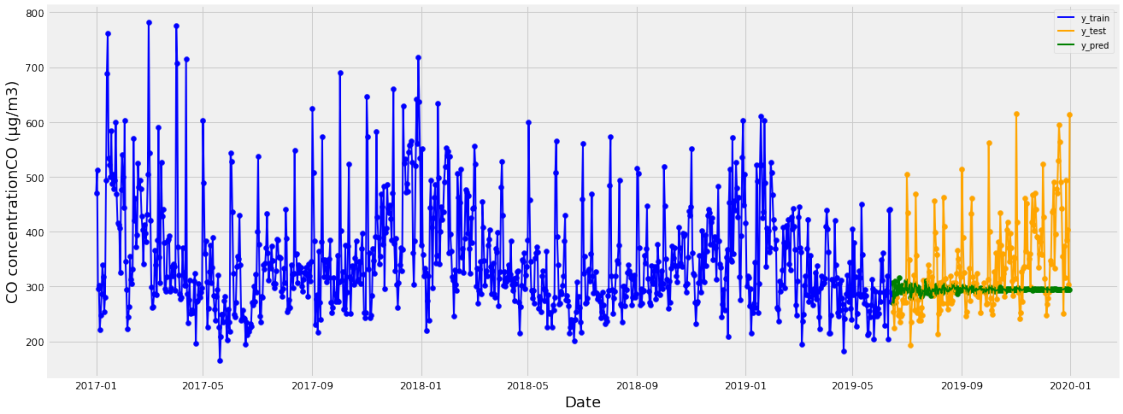
[131]:
```
from sktime.forecasting.ets import AutoETS
forecaster = AutoETS(auto=False, sp=12, n_jobs=-1)
forecaster.fit(y_train)
y_pred = forecaster.predict(fh)
fig= plt.figure(figsize=(20,8))
y_train, y_test = temporal_train_test_split(y, test_size=ts)
y_train_ind, y_test_ind = temporal_train_test_split(y.index, test_size=ts)
CO_train, CO_test = temporal_train_test_split(CO, test_size=ts)
#print(y_train.shape[0], y_test.shape[0])
#x_train=[i+1 for i in range(len(y_train))]
#x_test=[i+1 for i in range(len(y_train),len(y))]
plt.plot(y_train,color='blue',linewidth=2)
plt.scatter(y_train_ind,CO_train, color='blue')
plt.plot(y_test,color='orange',linewidth=2)
plt.scatter(y_test_ind,CO_test, color='orange')
plt.plot(y_pred,color='green',linewidth=2)
plt.scatter(y_test_ind,y_pred, color='green')
plt.legend(['y_train', 'y_test','y_pred'], loc='upper right')
plt.xlabel('Date')
plt.ylabel( 'CO concentrationCO ( g/m3)')
plt.grid(True)
#plt.plot(X3,y1,color='orange',linewidth=3)
#plt.title('Sigmoid and sine function')
#plt.grid(True)
plt.show()
smape_loss(y_pred, y_test)
```



[131]: 0.17216101736888903

Another common model is the ARIMA model. In `sktime`, we interface `pmdarima`, a package for automatically selecting the best ARIMA model. This since searches over a number of possible model parametrisations, it may take a bit longer.

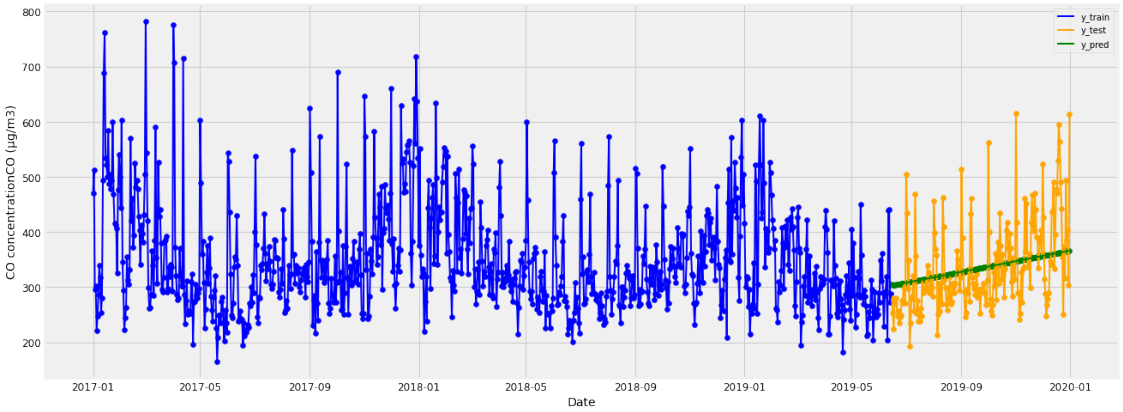
[132]:
```
forecaster = AutoARIMA(sp=12, suppress_warnings=True)
forecaster.fit(y_train)
y_pred = forecaster.predict(fh)
fig= plt.figure(figsize=(20,8))
y_train, y_test = temporal_train_test_split(y, test_size=ts)
y_train_ind, y_test_ind = temporal_train_test_split(y.index, test_size=ts)
CO_train, CO_test = temporal_train_test_split(CO, test_size=ts)
#print(y_train.shape[0], y_test.shape[0])
#x_train=[i+1 for i in range(len(y_train))]
#x_test=[i+1 for i in range(len(y_train),len(y))]
plt.plot(y_train,color='blue',linewidth=2)
plt.scatter(y_train_ind,CO_train, color='blue')
plt.plot(y_test,color='orange',linewidth=2)
plt.scatter(y_test_ind,CO_test, color='orange')
plt.plot(y_pred,color='green',linewidth=2)
plt.scatter(y_test_ind,y_pred, color='green')
plt.legend(['y_train', 'y_test','y_pred'], loc='upper right')
plt.xlabel('Date',fontsize=18)
plt.ylabel( 'CO concentrationCO ( g/m3)',fontsize=18)
plt.grid(True)
#plt.plot(X3,y1,color='orange',linewidth=3)
#plt.title('Sigmoid and sine function')
#plt.grid(True)
plt.show()
smape_loss(y_pred, y_test)
```



[132]: 0.1775422840877976

BATS and TBATS are two other time series forecasting algorithms that are contained in `sktime` by means of wrapping the package `tbats`.

[133]:
```
from sktime.forecasting.bats import BATS

forecaster = BATS(sp=12, use_trend=True, use_box_cox=False)
forecaster.fit(y_train)
y_pred = forecaster.predict(fh)
fig= plt.figure(figsize=(20,8))
y_train, y_test = temporal_train_test_split(y, test_size=ts)
y_train_ind, y_test_ind = temporal_train_test_split(y.index, test_size=ts)
CO_train, CO_test = temporal_train_test_split(CO, test_size=ts)
#print(y_train.shape[0], y_test.shape[0])
#x_train=[i+1 for i in range(len(y_train))]
#x_test=[i+1 for i in range(len(y_train),len(y))]
plt.plot(y_train,color='blue',linewidth=2)
plt.scatter(y_train_ind,CO_train, color='blue')
plt.plot(y_test,color='orange',linewidth=2)
plt.scatter(y_test_ind,CO_test, color='orange')
plt.plot(y_pred,color='green',linewidth=2)
plt.scatter(y_test_ind,y_pred, color='green')
plt.legend(['y_train', 'y_test','y_pred'], loc='upper right')
plt.xlabel('Date')
plt.ylabel( 'CO concentrationCO ( g/m3)')
plt.grid(True)
#plt.plot(X3,y1,color='orange',linewidth=3)
#plt.title('Sigmoid and sine function')
#plt.grid(True)
plt.show()
smape_loss(y_pred, y_test)
```



[133]:
```
0.15920351037807623
```

[134]:
```
y_train
```

[134]:
```
2017-01-01    470.245972
2017-01-02    512.894531
2017-01-03    296.407288
2017-01-04    302.471497
2017-01-05    220.845215
                 ...
2019-06-10    204.702103
2019-06-11    439.315948
2019-06-12    440.410950
2019-06-13    273.354706
2019-06-14    287.598419
Freq: D, Length: 895, dtype: float64
```

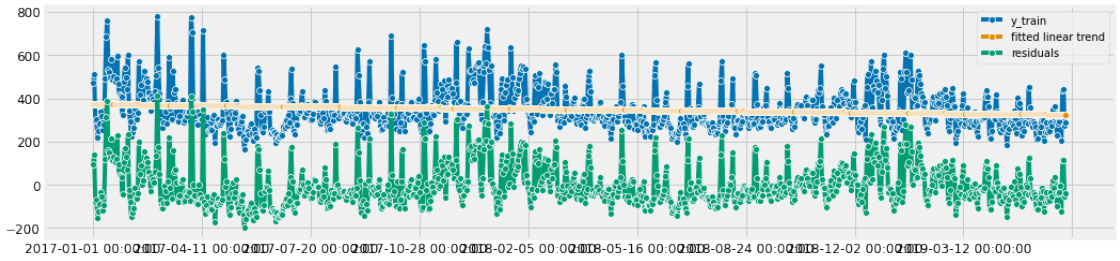
3.3.1 Detrending

Note that so far the reduction approach above does not take any seasonal or trend into account, but we can easily specify a pipeline which first detrends the data.

`sktime` provides a generic detrender, a transformer which uses any forecaster and returns the in-sample residuals of the forecaster's predicted values. For example, to remove the linear trend of a time series, we can write:

[135]:
```
# liner detrending
forecaster = PolynomialTrendForecaster(degree=1)
transformer = Detrender(forecaster=forecaster)
yt = transformer.fit_transform(y_train)

# internally, the Detrender uses the in-sample predictions
```

```
# of the PolynomialTrendForecaster
forecaster = PolynomialTrendForecaster(degree=1)
fh_ins = -np.arange(len(y_train))  # in-sample forecasting horizon
y_pred = forecaster.fit(y_train).predict(fh=fh_ins)

plot_series(y_train, y_pred, yt, labels=["y_train", "fitted linear trend",␣
↪"residuals"]);
```



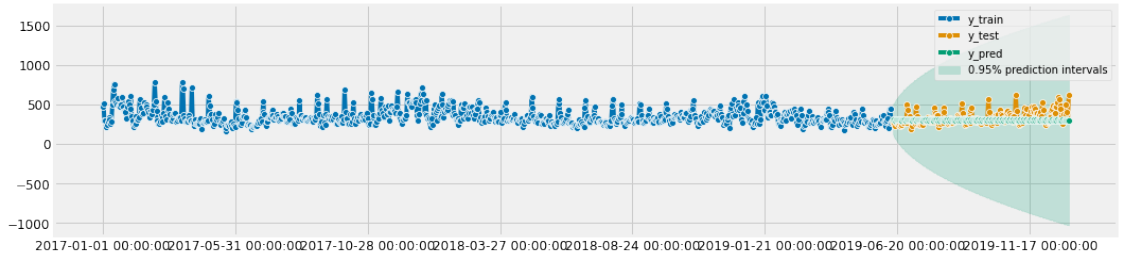
3.4 Prediction intervals

So far, we've only looked at point forecasts. In many cases, we're also interested in prediction intervals. `sktime`'s interface support prediction intervals, but we haven't implemented them for all algorithms yet.

Here, we use the Theta forecasting algorithm:

[136]:
```
forecaster = ThetaForecaster(sp=12)
forecaster.fit(y_train)
alpha = 0.05  # 95% prediction intervals
y_pred, pred_ints = forecaster.predict(fh, return_pred_int=True, alpha=alpha)
fig, ax = plot_series(y_train, y_test, y_pred, labels=["y_train", "y_test",␣
↪"y_pred"])

ax.fill_between(
    ax.get_lines()[-1].get_xdata(),
    pred_ints["lower"],
    pred_ints["upper"],
    alpha=0.2,
    color=ax.get_lines()[-1].get_c(),
    label=f"{1 - alpha}% prediction intervals",
)
ax.legend();
smape_loss(y_test, y_pred)
```

[136]:
```
0.17598386876904176
```



3.5 Forecasting with fbprophet

[137]:
```
import pandas as pd
data1=pd.read_csv("Data_2017.csv")
data2=pd.read_csv("Data_2018.csv")
data3=pd.read_csv("Data_2019.csv")
p_id1=data1['pollutant_id'].to_list()
date1=data1['date_time'].to_list()
p_value1=data1['pollutant_value'].to_list()

p_id2=data2['pollutant_id'].to_list()
date2=data2['date_time'].to_list()
p_value2=data2['pollutant_value'].to_list()
p_id3=data3['pollutant_id'].to_list()
date3=data3['date_time'].to_list()
p_value3=data3['pollutant_value'].to_list()

p_id=p_id1+p_id2+p_id3
date=date1+date2+date3
p_value=p_value1+p_value2+p_value3

date=[date[i] for i in range(len(p_id)) if(p_id[i]==6)]
CO_2017=[p_value[i] for i in range(len(p_id)) if(p_id[i]==6)]

for i in range(len(CO_2017)):
    if (CO_2017[i]=='BDL'):
        CO_2017[i]='nan'
    else:
        pass

for i in range(len(CO_2017)):
    if (CO_2017[i]=='nan'):
        CO_2017[i]='0'
    else:
        pass
```

```
#CO_2017=[int(CO_2017[i]) for i in range(len(CO_2017))]
data=[[date[i], CO_2017[i]] for i in range(len(date))]
data= pd.DataFrame(data, columns = ['Date', 'CO'])
data["CO"] = pd.to_numeric(data["CO"], downcast="float")
column_means = data['CO'].mean()
data['CO'].fillna(column_means)
```

[137]:
```
0         1554.890015
1         1126.829956
2         2066.790039
3         1713.380005
4         1685.310059
             …
210235    1665.099976
210236    2269.500000
210237    2996.699951
210238    2517.300049
210239    1723.000000
Name: CO, Length: 210240, dtype: float32
```

[138]:
```
#Printing the date upto period 'Day'
data['Date'] = pd.to_datetime(data['Date'])
#Drops all rows with missing values
#data=data.dropna()
#Setting the date column as an index column to allow resampling
data=data.set_index('Date')
#Resampling data to give only the mean daily CO concentrations
data=data.resample('D').mean()
data['date'] = data.index
#data['date'] = pd.to_datetime(data['date']).dt.to_period('D')
#mean_value=data['CO'].mean()
data['CO'].values
```

[138]:
```
array([470.24597, 512.89453, 296.4073 , …, 403.23   , 304.0566 ,
       614.26825], dtype=float32)
```

[139]:
```
from warnings import simplefilter

import numpy as np
import pandas as pd

from sktime.datasets import load_airline
from sktime.forecasting.arima import ARIMA, AutoARIMA
from sktime.forecasting.base import ForecastingHorizon
from sktime.forecasting.compose import (
```

```
    EnsembleForecaster,
    ReducedRegressionForecaster,
    TransformedTargetForecaster,
)
from sktime.forecasting.exp_smoothing import ExponentialSmoothing
from sktime.forecasting.model_selection import (
    ForecastingGridSearchCV,
    SlidingWindowSplitter,
    temporal_train_test_split,
)
from sktime.forecasting.naive import NaiveForecaster
from sktime.forecasting.theta import ThetaForecaster
from sktime.forecasting.trend import PolynomialTrendForecaster
from sktime.performance_metrics.forecasting import sMAPE, smape_loss
from sktime.transformations.series.detrend import Deseasonalizer, Detrender
from sktime.utils.plotting import plot_series

simplefilter("ignore", FutureWarning)
%matplotlib inline
import matplotlib.pyplot as plt
```

[140]:
```
from fbprophet import Prophet
from fbprophet.diagnostics import cross_validation
from fbprophet.diagnostics import performance_metrics
from fbprophet.plot import plot_cross_validation_metric
```

[141]:
```
import math
#Deriving the lists of Date and CO columns of the data
Date=data['date'].to_list()
CO  =data['CO'].to_list()
date=[i+1 for i in range(len(CO)) ]
#CO=[math.log(CO[i]) for i in range(len(CO))]
#Generating timeseries data based on the two lists
data1= pd.Series(CO,Date)
#CO=pd.Series(CO)
```

[142]:
```
ts=200
z_train, z_test = temporal_train_test_split(data1, test_size=ts)
Date_train, Date_test = temporal_train_test_split(Date, test_size=ts)
date=Date_train
z=list(z_train)
len(Date_train)
```

[142]: 895

[143]:
```
data2=[[ z[i],date[i]] for i in range(len(z))]
data2= pd.DataFrame(data2, columns = ['y', 'ds'])
```

```
#data['ds']=pd.DatetimeIndex(data['date'])
data2.dtypes
data2['ds']=pd.to_datetime(data2.ds)
```

[144]:
```
train=data2[:895]
test=data2[895:]
```

3.6 Useful resources

https://towardsdatascience.com/a-quick-start-of-time-series-forecasting-with-a-practical-example-using-fb-prophet-31c4447a2274

https://sailajakarra.medium.com/facebook-prophet-for-time-series-cf26be1be274#:~:text=Facebook%20Prophet%

3.7 Model

[145]:
```
from fbprophet import Prophet
m=Prophet(growth='linear',interval_width=0.95,seasonality_mode='additive',␣
 ↪yearly_seasonality=False,
          weekly_seasonality=False,daily_seasonality=True )
m.add_seasonality(name='monthly', period=30.5, fourier_order=30,␣
 ↪prior_scale=10,mode='additive')
m.add_seasonality(name='weakly', period=7, fourier_order=40,␣
 ↪prior_scale=15,mode='additive')
m.add_seasonality(name='daily', period=1, fourier_order=10,␣
 ↪prior_scale=15,mode='additive')
m.add_seasonality(name='yearly', period=365.25, fourier_order=25,␣
 ↪prior_scale=15,mode='additive')
m.add_seasonality(name='quarterly', period=365.25/4, fourier_order=22,␣
 ↪prior_scale=15,mode='additive')
model=m.fit(train)
```

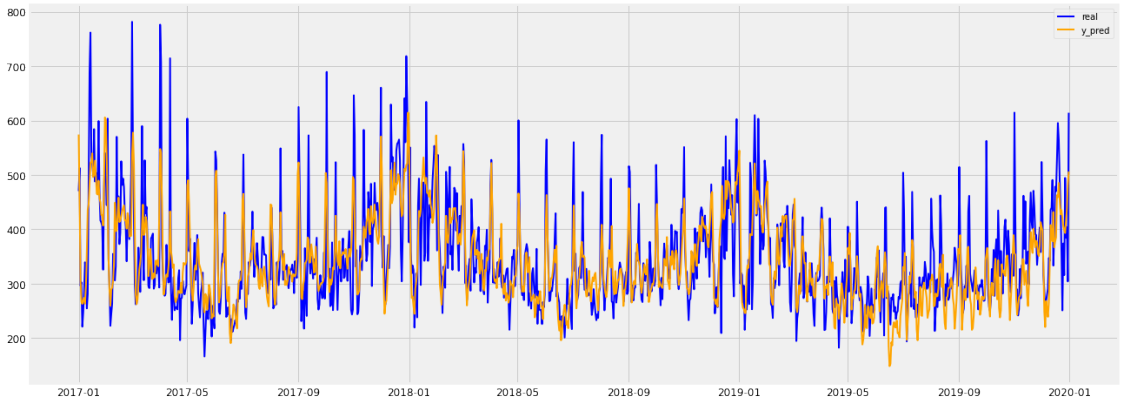
[146]:
```
future=m.make_future_dataframe(periods=200,freq='D')
forecast=m.predict(future)
pred=forecast['yhat'].to_list()
pred= pd.Series(pred,Date)
forecast.head()
```

[146]:

	ds	trend	yhat_lower	yhat_upper	trend_lower	trend_upper	\
0	2017-01-01	460.260113	463.396764	679.174428	460.260113	460.260113	
1	2017-01-02	460.189189	331.450902	559.244682	460.189189	460.189189	
2	2017-01-03	460.118264	196.425845	417.731795	460.118264	460.118264	
3	2017-01-04	460.047339	154.021989	385.127985	460.047339	460.047339	
4	2017-01-05	459.976414	149.767399	376.514940	459.976414	459.976414	

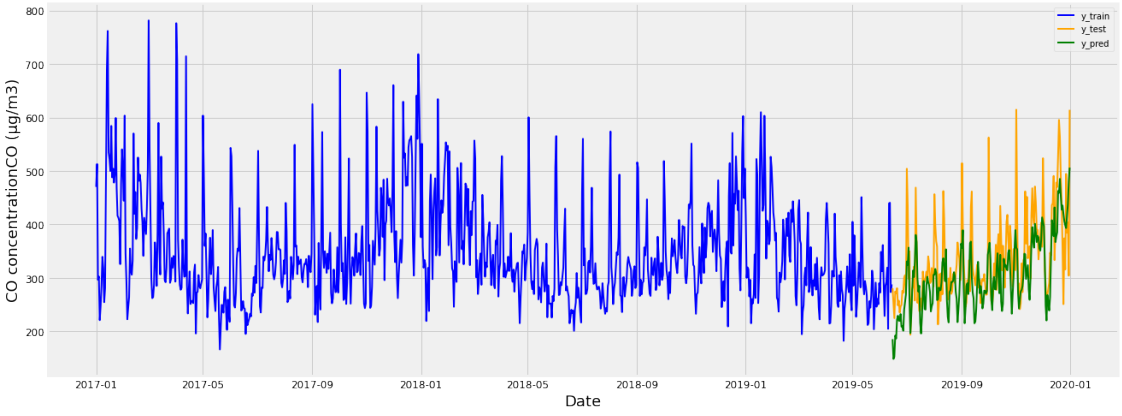
```
   additive_terms  additive_terms_lower  additive_terms_upper      daily  …  \
0      113.908732            113.908732            113.908732 -38.842031  …
1       -7.847722             -7.847722             -7.847722 -38.842031  …
2     -152.325046           -152.325046           -152.325046 -38.842031  …
3     -194.021238           -194.021238           -194.021238 -38.842031  …
4     -197.117476           -197.117476           -197.117476 -38.842031  …

      weekly  weekly_lower  weekly_upper     yearly  yearly_lower  \
0 -59.025985    -59.025985    -59.025985  79.929408     79.929408
1 -46.681325    -46.681325    -46.681325  34.477172     34.477172
2 -53.489355    -53.489355    -53.489355  -7.277813     -7.277813
3 -45.121199    -45.121199    -45.121199 -40.918012    -40.918012
4 -44.376954    -44.376954    -44.376954 -62.967033    -62.967033

   yearly_upper  multiplicative_terms  multiplicative_terms_lower  \
0     79.929408                   0.0                         0.0
1     34.477172                   0.0                         0.0
2     -7.277813                   0.0                         0.0
3    -40.918012                   0.0                         0.0
4    -62.967033                   0.0                         0.0

   multiplicative_terms_upper        yhat
0                         0.0  574.168845
1                         0.0  452.341467
2                         0.0  307.793218
3                         0.0  266.026102
4                         0.0  262.858939

[5 rows x 28 columns]
```

[147]:
```
import matplotlib.pyplot as plt
fig= plt.figure(figsize=(20,8))
plt.plot(data1,color='blue',linewidth=2)
#plt.scatter(y_train_ind,CO_train, color='blue')
plt.plot(pred,color='orange',linewidth=2)
plt.legend(['real','y_pred'], loc='upper right')
#plt.xlabel('Days from Jan 9th to Nov 9th')
#plt.ylabel( 'CO concentrationCO ( g/m3)')
plt.grid(True)
```



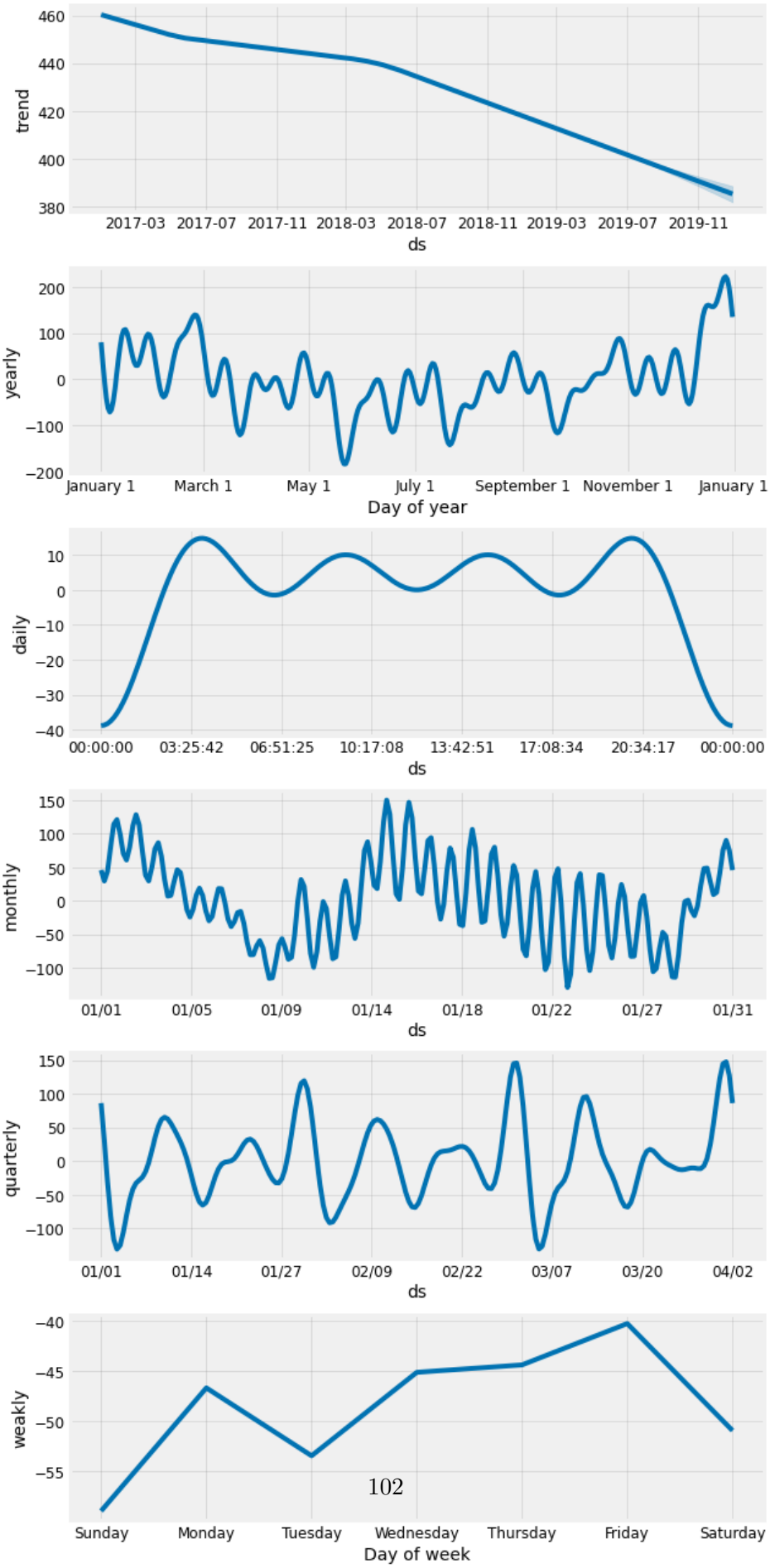
[148]:
```
z_pred=forecast.iloc[895:,:]
z_pred=z_pred['yhat'].to_list()
z_pred= pd.Series(z_pred,Date_test)
```

[149]:
```
fig= plt.figure(figsize=(20,8))
plt.plot(z_train,color='blue',linewidth=2)
plt.plot(z_test,color='orange',linewidth=2)
plt.plot(z_pred,color='green',linewidth=2)
plt.legend(['y_train', 'y_test','y_pred'], loc='upper right')
plt.xlabel('Date',fontsize=18)
plt.ylabel( 'CO concentrationCO ( g/m3)',fontsize=18)
plt.grid(True)
#plt.plot(X3,y1,color='orange',linewidth=3)
#plt.title('Sigmoid and sine function')
#plt.grid(True)
plt.show()
smape_loss(z_pred, z_test)
```



[149]: 0.16115445556136873

[150]:
```
import warnings
warnings.filterwarnings("ignore")
fig=m.plot_components(forecast)
```


trend
460
440
420
400
380
2017-03
2017-07
2017-11
2018-03
2018-07
2018-11
2019-03
2019-07
2019-11
ds
yearly
200
100
0
−100
−200
January 1
March 1
May 1
July 1
September 1
November 1
January 1
Day of year
daily
10
0
−10
−20
−30
−40
00:00:00
03:25:42
06:51:25
10:17:08
13:42:51
17:08:34
20:34:17
00:00:00
ds
monthly
150
100
50
0
−50
−100
01/01
01/05
01/09
01/14
01/18
01/22
01/27
01/31
ds
quarterly
150
100
50
0
−50
−100
01/01
01/14
01/27
02/09
02/22
03/07
03/20
04/02
ds
weakly
−40
−45
−50
−55
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Day of week

[151]:
```
import warnings
warnings.filterwarnings("ignore")
from ipywidgets import IntProgress

# Initial training period.
initial= 600
initial= str(initial)+' days'
#Period length that we perform the cross validation for.
period= 100
period=str(period)+' days'
#Horizon of prediction essentially for each fold.
horizon = 200
horizon=str(horizon)+' days'
fb_cv=cross_validation(m,initial=initial,period=period,
horizon=horizon,parallel="processes")
# Performance Metrics of fb_cv
performance_metrics(fb_cv)
```

```
INFO:fbprophet:Making 1 forecasts with cutoffs between 2018-11-26 00:00:00 and
2018-11-26 00:00:00
INFO:fbprophet:Applying in parallel with
<concurrent.futures.process.ProcessPoolExecutor object at 0x148b0fa30>
```

[151]:

	horizon	mse	rmse	mae	mape	mdape	\
0	20 days	10425.533018	102.105499	76.843033	0.246074	0.174663	
1	21 days	10516.514248	102.550057	78.250362	0.247799	0.174663	
2	22 days	11569.976321	107.563824	83.357909	0.262169	0.198088	
3	23 days	11041.264006	105.077419	80.934881	0.250113	0.189568	
4	24 days	11264.171363	106.132801	82.212462	0.251838	0.194184	
..	…	…	…	…	…	…	
176	196 days	4290.057592	65.498531	51.920686	0.191420	0.151521	
177	197 days	4447.178636	66.687170	53.285699	0.191294	0.151521	
178	198 days	4472.695486	66.878214	53.496574	0.189087	0.146712	
179	199 days	4470.108989	66.858874	53.478255	0.190037	0.146712	
180	200 days	4518.211024	67.217639	54.402464	0.192049	0.146712	

	coverage
0	0.80
1	0.80
2	0.75
3	0.80
4	0.80
..	…
176	0.90
177	0.90

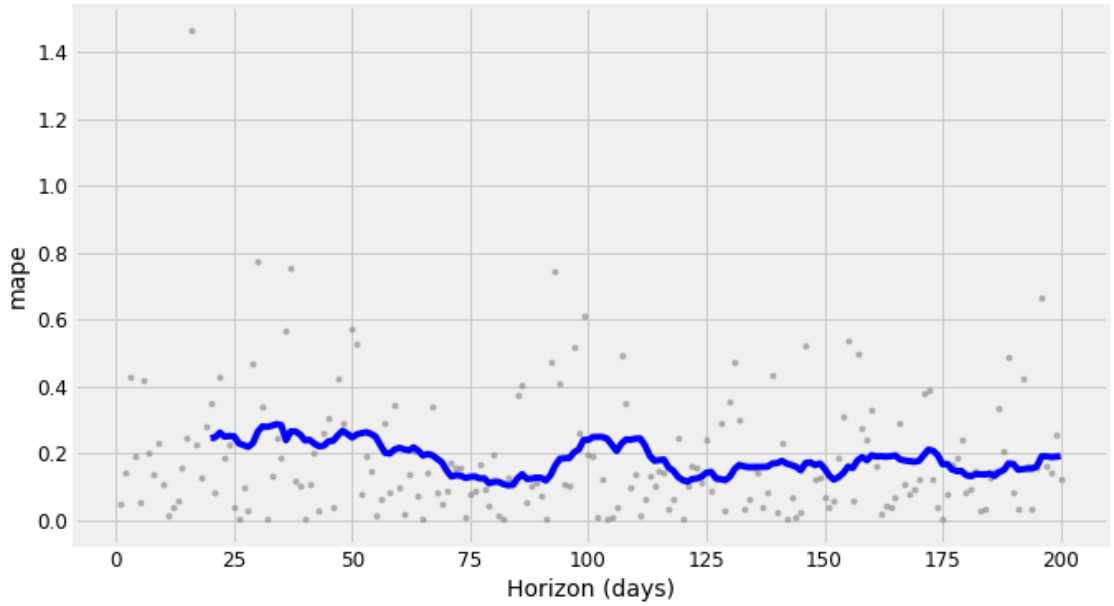
```
178      0.90
179      0.90
180      0.90

[181 rows x 7 columns]
```

[152]:
```
fb_cv.head()
```

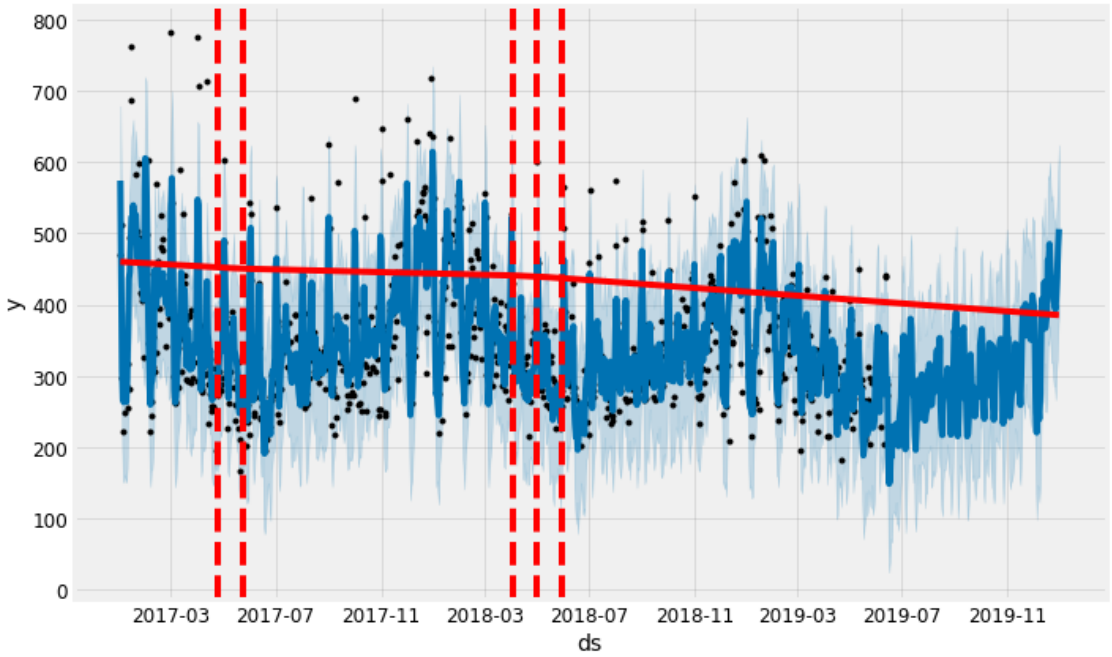
[152]:

	ds	yhat	yhat_lower	yhat_upper	y	cutoff
0	2018-11-27	408.510679	288.756768	519.313051	390.259888	2018-11-26
1	2018-11-28	422.869510	308.331988	533.146504	370.817017	2018-11-26
2	2018-11-29	445.746763	324.707666	562.561704	312.414917	2018-11-26
3	2018-11-30	459.336858	345.252135	582.690122	384.874573	2018-11-26
4	2018-12-01	507.560127	397.530120	616.290739	482.472290	2018-11-26

[153]:
```
plot_cross_validation_metric(fb_cv,'mape');
```



[154]:
```
# changing trend points
from fbprophet.plot import add_changepoints_to_plot
fig=m.plot(forecast)
a=add_changepoints_to_plot(fig.gca(),m,forecast)
```



[155]:
```
from fbprophet.plot import plot_plotly, plot_components_plotly

plot_plotly(m, forecast)
```

[156]:
```
plot_components_plotly(m, forecast)
```

[157]:
```
# Python
#import itertools
#import numpy as np
#import pandas as pd
#cutoffs = pd.to_datetime(['2018-02-15', '2018-08-15', '2019-02-15'])
#param_grid = {
    #'changepoint_prior_scale': [0.001, 0.01, 0.1, 0.5],
    #'seasonality_prior_scale': [0.01, 0.1, 1.0, 10.0],
#}

# Generate all combinations of parameters
#all_params = [dict(zip(param_grid.keys(), v)) for v in itertools.
 ↪product(*param_grid.values())]
#rmses = []  # Store the RMSEs for each params here

# Use cross validation to evaluate all parameters
#for params in all_params:
    #m = Prophet(**params,daily_seasonality=True).fit(train)  # Fit model with␣
 ↪given params
```

```
    #df_cv = cross_validation(m, cutoffs=cutoffs, horizon='200 days',␣
↪parallel=None)
    #df_p = performance_metrics(df_cv, rolling_window=1)
    #rmses.append(df_p['rmse'].values[0])

# Find the best parameters
#tuning_results = pd.DataFrame(all_params)
#tuning_results['rmse'] = rmses
#print(tuning_results)
```

[158]:
```
#best_params = all_params[np.argmin(rmses)]
#print(best_params)
```

[ ]:
```

```

[ ]:
```

```

3.8 Forecasting with LSTM

[159]:
```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from sklearn.preprocessing import MinMaxScaler
```

[160]:
```
data=[(CO[i]-min(CO))/(max(CO)-min(CO)) for i in range(len(CO))]

len(data)
```

[160]:
```
1095
```

[161]:
```
data_train, data_test = temporal_train_test_split(data, test_size=200)
```

[162]:
```
#data=np.array(ts_data)
#scaler=MinMaxScaler(feature_range=(0,1))
#scaled_data=scaler.fit_transform(ts_data)
```

[163]:
```
def prepare_data(ts_data,n_steps):
    X,y=[],[]
    for i in range(len(ts_data)):
        end_ix=i+n_steps
        if end_ix>len(ts_data)-1:
            break
        seq_x,seq_y=ts_data[i:end_ix],ts_data[end_ix]
```

```
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X),np.array(y)
```

[164]:
```
import math
CO_train, CO_test = temporal_train_test_split(CO, test_size=200)
ts_data=CO_train

n_steps=20

X,y=prepare_data(data_train,n_steps)
```

[165]:
```
#print(X),print(y)
```

[166]:
```
# Reshape from [samples,timesteps] to [samples,timesteps,features]
n_features=1
X=X.reshape(X.shape[0],X.shape[1],n_features)
y=y.reshape(X.shape[0],n_features)
```

4 Building LSTM Model

[167]:
```
from keras.models import Sequential
import tensorflow as tf
from keras.layers import LSTM
from keras.layers import Dense, Dropout
model = Sequential()
# 50 neurons in first hidden layer
model.add(LSTM(units=200, input_shape=(X.shape[1], X.shape[2]),␣
 ↪activation='tanh'))
model.add(Dense(units=100,activation='tanh'))
model.add(Dense(units=20,activation='tanh'))
model.add(Dense(1,kernel_initializer='normal', activation='linear'))
model.compile(loss='mae', optimizer=tf.keras.optimizers.Adam(lr=0.01))

model.fit(X,y,epochs=100,verbose=1)
```

```
Epoch 1/100
28/28 [==============================] - 2s 21ms/step - loss: 0.1504
Epoch 2/100
28/28 [==============================] - 1s 21ms/step - loss: 0.1025
Epoch 3/100
28/28 [==============================] - 1s 21ms/step - loss: 0.0926
Epoch 4/100
28/28 [==============================] - 1s 32ms/step - loss: 0.0928
```

```
Epoch 5/100
28/28 [==============================] - 1s 24ms/step - loss: 0.0855
Epoch 6/100
28/28 [==============================] - 1s 22ms/step - loss: 0.0871
Epoch 7/100
28/28 [==============================] - 1s 24ms/step - loss: 0.0908
Epoch 8/100
28/28 [==============================] - 1s 23ms/step - loss: 0.0889
Epoch 9/100
28/28 [==============================] - 1s 25ms/step - loss: 0.0849
Epoch 10/100
28/28 [==============================] - 1s 24ms/step - loss: 0.0908
Epoch 11/100
28/28 [==============================] - 1s 23ms/step - loss: 0.0846
Epoch 12/100
28/28 [==============================] - 1s 23ms/step - loss: 0.0835
Epoch 13/100
28/28 [==============================] - 1s 23ms/step - loss: 0.0823
Epoch 14/100
28/28 [==============================] - 1s 23ms/step - loss: 0.0883
Epoch 15/100
28/28 [==============================] - 1s 23ms/step - loss: 0.0875
Epoch 16/100
28/28 [==============================] - 1s 23ms/step - loss: 0.0868
Epoch 17/100
28/28 [==============================] - 1s 24ms/step - loss: 0.0819
Epoch 18/100
28/28 [==============================] - 1s 23ms/step - loss: 0.0853
Epoch 19/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0884
Epoch 20/100
28/28 [==============================] - 1s 27ms/step - loss: 0.0834
Epoch 21/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0834
Epoch 22/100
28/28 [==============================] - 1s 27ms/step - loss: 0.0810
Epoch 23/100
28/28 [==============================] - 1s 25ms/step - loss: 0.0826
Epoch 24/100
28/28 [==============================] - 1s 25ms/step - loss: 0.0820
Epoch 25/100
28/28 [==============================] - 1s 39ms/step - loss: 0.0809
Epoch 26/100
28/28 [==============================] - 1s 30ms/step - loss: 0.0791
Epoch 27/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0767
Epoch 28/100
28/28 [==============================] - 1s 32ms/step - loss: 0.0802
```

```
Epoch 29/100
28/28 [==============================] - 1s 27ms/step - loss: 0.0763
Epoch 30/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0825
Epoch 31/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0767
Epoch 32/100
28/28 [==============================] - 1s 25ms/step - loss: 0.0835
Epoch 33/100
28/28 [==============================] - 1s 29ms/step - loss: 0.0816
Epoch 34/100
28/28 [==============================] - 1s 27ms/step - loss: 0.0800
Epoch 35/100
28/28 [==============================] - 1s 28ms/step - loss: 0.0881
Epoch 36/100
28/28 [==============================] - 1s 42ms/step - loss: 0.0843
Epoch 37/100
28/28 [==============================] - 1s 37ms/step - loss: 0.0860
Epoch 38/100
28/28 [==============================] - 1s 37ms/step - loss: 0.0866
Epoch 39/100
28/28 [==============================] - 1s 39ms/step - loss: 0.0800
Epoch 40/100
28/28 [==============================] - 1s 28ms/step - loss: 0.0788
Epoch 41/100
28/28 [==============================] - 1s 27ms/step - loss: 0.0810
Epoch 42/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0788
Epoch 43/100
28/28 [==============================] - 1s 28ms/step - loss: 0.0767
Epoch 44/100
28/28 [==============================] - 1s 31ms/step - loss: 0.0849
Epoch 45/100
28/28 [==============================] - 1s 28ms/step - loss: 0.0826
Epoch 46/100
28/28 [==============================] - 1s 27ms/step - loss: 0.0830
Epoch 47/100
28/28 [==============================] - 1s 24ms/step - loss: 0.0746
Epoch 48/100
28/28 [==============================] - 1s 27ms/step - loss: 0.0751
Epoch 49/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0778
Epoch 50/100
28/28 [==============================] - 1s 32ms/step - loss: 0.0770
Epoch 51/100
28/28 [==============================] - 1s 27ms/step - loss: 0.0822
Epoch 52/100
28/28 [==============================] - 1s 24ms/step - loss: 0.0777
```

```
Epoch 53/100
28/28 [==============================] - 1s 27ms/step - loss: 0.0750
Epoch 54/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0782
Epoch 55/100
28/28 [==============================] - 1s 28ms/step - loss: 0.0769
Epoch 56/100
28/28 [==============================] - 1s 28ms/step - loss: 0.0767
Epoch 57/100
28/28 [==============================] - 1s 24ms/step - loss: 0.0842
Epoch 58/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0743
Epoch 59/100
28/28 [==============================] - 1s 29ms/step - loss: 0.0752
Epoch 60/100
28/28 [==============================] - 1s 24ms/step - loss: 0.0781
Epoch 61/100
28/28 [==============================] - 1s 23ms/step - loss: 0.0739
Epoch 62/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0706
Epoch 63/100
28/28 [==============================] - 1s 36ms/step - loss: 0.0769
Epoch 64/100
28/28 [==============================] - 1s 28ms/step - loss: 0.0714
Epoch 65/100
28/28 [==============================] - 1s 23ms/step - loss: 0.0662
Epoch 66/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0739
Epoch 67/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0796
Epoch 68/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0750
Epoch 69/100
28/28 [==============================] - 1s 24ms/step - loss: 0.0775
Epoch 70/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0750
Epoch 71/100
28/28 [==============================] - 1s 27ms/step - loss: 0.0719
Epoch 72/100
28/28 [==============================] - 1s 22ms/step - loss: 0.0747
Epoch 73/100
28/28 [==============================] - 1s 23ms/step - loss: 0.0691
Epoch 74/100
28/28 [==============================] - 1s 23ms/step - loss: 0.0771
Epoch 75/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0724
Epoch 76/100
28/28 [==============================] - 1s 27ms/step - loss: 0.0711
```

```
Epoch 77/100
28/28 [==============================] - 1s 31ms/step - loss: 0.0701
Epoch 78/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0744
Epoch 79/100
28/28 [==============================] - 1s 24ms/step - loss: 0.0726
Epoch 80/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0708
Epoch 81/100
28/28 [==============================] - 1s 27ms/step - loss: 0.0752
Epoch 82/100
28/28 [==============================] - 1s 25ms/step - loss: 0.0665
Epoch 83/100
28/28 [==============================] - 1s 25ms/step - loss: 0.0669
Epoch 84/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0686
Epoch 85/100
28/28 [==============================] - 1s 31ms/step - loss: 0.0672
Epoch 86/100
28/28 [==============================] - 1s 27ms/step - loss: 0.0672
Epoch 87/100
28/28 [==============================] - 1s 23ms/step - loss: 0.0654
Epoch 88/100
28/28 [==============================] - 1s 23ms/step - loss: 0.0702
Epoch 89/100
28/28 [==============================] - 1s 23ms/step - loss: 0.0688
Epoch 90/100
28/28 [==============================] - 1s 23ms/step - loss: 0.0649
Epoch 91/100
28/28 [==============================] - 1s 24ms/step - loss: 0.0674
Epoch 92/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0673
Epoch 93/100
28/28 [==============================] - 1s 26ms/step - loss: 0.0673
Epoch 94/100
28/28 [==============================] - 1s 25ms/step - loss: 0.0685
Epoch 95/100
28/28 [==============================] - 1s 30ms/step - loss: 0.0710
Epoch 96/100
28/28 [==============================] - 1s 36ms/step - loss: 0.0685
Epoch 97/100
28/28 [==============================] - 1s 28ms/step - loss: 0.0618
Epoch 98/100
28/28 [==============================] - 1s 31ms/step - loss: 0.0637
Epoch 99/100
28/28 [==============================] - 1s 24ms/step - loss: 0.0593
Epoch 100/100
28/28 [==============================] - 1s 23ms/step - loss: 0.0677
```

[167]:
```
<tensorflow.python.keras.callbacks.History at 0x160aac340>
```

[168]:
```
import numpy as np
test_data=[data[i] for i in range(895,1095)]
X_test=[]
y_test=[CO[i] for i in range(905,1095)]
y_test=np.array(y_test)
for i in range(10, len(test_data)):
    X_test.append(test_data[i-10:i])
Y_test=np.array(y_test)
y_test.shape
```

[168]:
```
(190,)
```

[169]:
```
X_test=np.array(X_test)
X_test.shape

X_test=np.reshape(X_test,(190,10,1))
X_test.shape
```

[169]:
```
(190, 10, 1)
```

[170]:
```
y_pred=model.predict(X_test)
#y_pred=y_pred.reshape(y_test.shape)
```

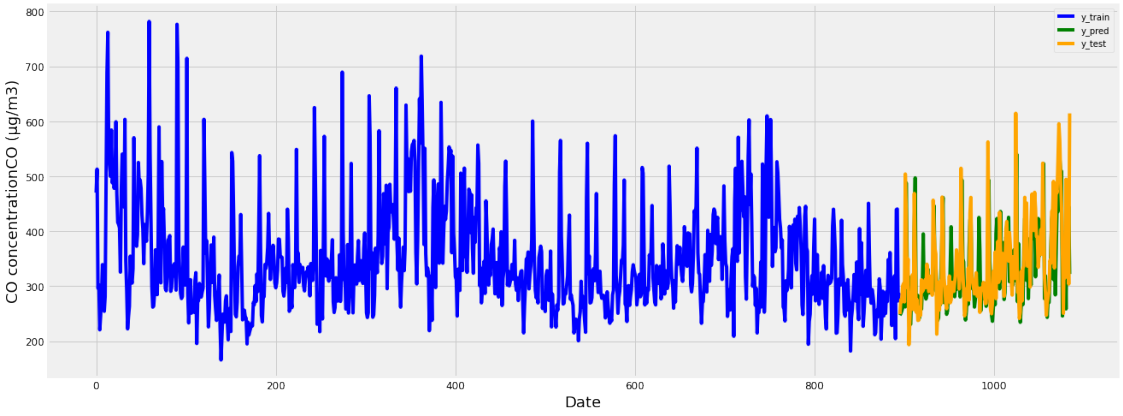
```
WARNING:tensorflow:Model was constructed with shape (None, 20, 1) for input
KerasTensor(type_spec=TensorSpec(shape=(None, 20, 1), dtype=tf.float32,
name='lstm_input'), name='lstm_input', description="created by layer
'lstm_input'"), but it was called on an input with incompatible shape (None, 10,
1).
```

[171]:
```
y_pred=[y_pred[i]*(max(CO)-min(CO))+min(CO) for i in range(190)]
```

[172]:
```
Date_test=[i for i in range(190)]
```

[173]:
```
x1=[i for i in range(895)]
x2=[i for i in range(895,1085)]
fig= plt.figure(figsize=(20,8))
plt.plot(x1,CO_train, color='blue')
plt.plot(x2,y_pred,color='green')
plt.plot(x2,Y_test,color='orange')
plt.legend(['y_train','y_pred', 'y_test'], loc='upper right')
plt.xlabel('Date',fontsize=18)
plt.ylabel( 'CO concentrationCO ( g/m3)',fontsize=18)
plt.grid(True)
plt.show()
y_pred=[sum(list(y_pred[i])) for i in range(190)]
```

```
y_test= pd.Series(y_test,Date_test)
y_pred= pd.Series(y_pred,Date_test)
smape_loss(y_pred, y_test)
```



[173]: 0.14570342492216762

4.0.1 Forecasting with XGBoost

[174]:
```
data= pd.DataFrame(CO, columns = [ 'CO'])
data.head()
```

[174]:

	CO
0	470.245972
1	512.894531
2	296.407288
3	302.471497
4	220.845215

[175]:
```
data["target"]=data.CO.shift(-1)
```

[176]:
```
data.dropna(inplace=True)
```

[177]:
```
def train_test_split(data,perc):
    data=data.values
    n=int(len(data)*(1-perc))
    return data[:n],data[n:]
```

[178]:
```
train, test = train_test_split(data,200/1093)
X=train[:,:-1]
y=train[:,-1]
```

[179]:
```
from xgboost import XGBRegressor
model=XGBRegressor(objective="reg:squarederror",n_estimators=100)
model.fit(X,y)
```

[179]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

[180]:
```
test[2]
```

[180]:
```
array([280.53613281, 253.6010437 ])
```

[181]:
```
val=np.array(test[3,0]).reshape(1,-1)
pred=model.predict(val)
pred[0]
```

[181]:
```
301.99792
```

4.1 Work forward validation

[182]:
```
def xgb_predict(train,val):
    train=np.array(train)
    X,y=train[:,:-1],train[:,-1]
    model=XGBRegressor(objective="reg:squarederror",n_estimators=500)
    model.fit(X,y)
    val=np.array(val).reshape(1,-1)
    pred=model.predict(val)
    return pred[0]
```

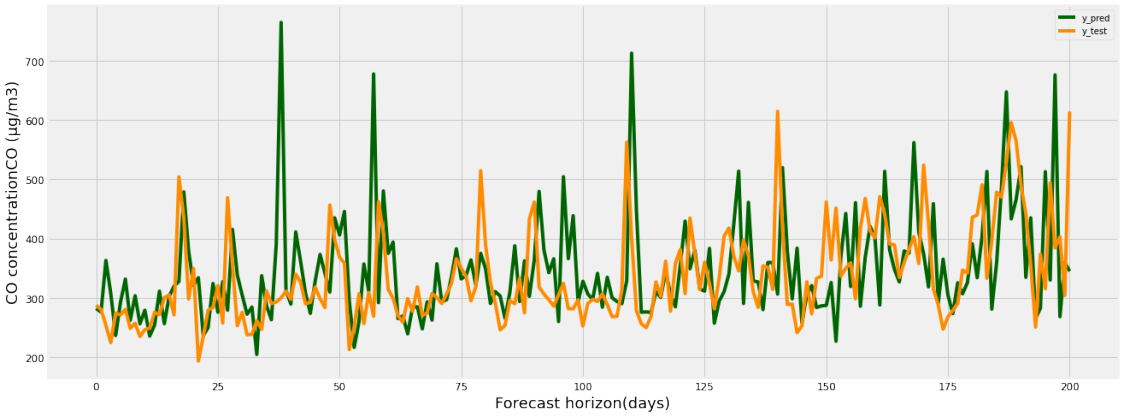
[183]:
```
from sklearn.metrics import mean_squared_error
def validate(data,perc):
    predictions=[]
    train, test = train_test_split(data,perc)
    history=[x for x in train]
    for i in range(len(test)):
        test_X,test_y=test[i,:-1],test[i,-1]
        pred=xgb_predict(history,test_X[0])
        predictions.append(pred)
        history.append(test[i])
    error=mean_squared_error(test[:,-1],predictions,squared=False)
```

```
    return error, test[:,-1] ,predictions
```

[184]:
```
%%time
rmse,y,pred=validate(data,200/1093)
print(rmse)
```

```
99.88808850600822
CPU times: user 6min 50s, sys: 20.4 s, total: 7min 10s
Wall time: 2min 5s
```

[185]:
```
ind=[i for i in range(201)]
y_pred= pd.Series(pred,ind)
y_test= pd.Series(y,ind)
fig= plt.figure(figsize=(20,8))
plt.plot(y_pred,color='darkgreen')
plt.plot(y_test,color='darkorange')
plt.legend(['y_pred','y_test'], loc='upper right')
plt.xlabel('Forecast horizon(days)',fontsize=18)
plt.ylabel( 'CO concentrationCO ( g/m3)',fontsize=18)
plt.grid(True)
plt.show()
smape_loss(y_pred, y_test)
```



[185]: 0.18917575463392064

4.2 Summary

As we have seen, in order to make forecasts, we need to first specify (or build) a model, then fit it to the training data, and finally call predict to generate forecasts for the given forecasting horizon.

- `sktime` comes with several forecasting algorithms (or forecasters) and tools for composite

model building. All forecaster share a common interface. Forecasters are trained on a single series of data and make forecasts for the provided forecasting horizon.

- `sktime` has a number of statistical forecasting algorithms, based on implementations in statsmodels. For example, to use exponential smoothing with an additive trend component and multiplicative seasonality, we can write the following.

4.3 Useful resources

- For more details, take a look at our paper on forecasting with sktime in which we discuss the forecasting API in more detail and use it to replicate and extend the M4 study.
- For a good introduction to forecasting, see Hyndman, Rob J., and George Athanasopoulos. Forecasting: principles and practice. OTexts, 2018.
- For comparative benchmarking studies/forecasting competitions, see the M4 competition and the currently running M5 competition.

[186]:
```
import xlwt
from tempfile import TemporaryFile
book = xlwt.Workbook()
sheet1 = book.add_sheet('sheet1')

supersecretdata = CO

for i,e in enumerate(supersecretdata):
    sheet1.write(i,1,e)

name = "random.xls"
book.save(name)
book.save(TemporaryFile())
```

[ ]:

[ ]: