

Sensor Calibration

September 30, 2021

1 DATA

2 CO DATA

```
[1]: import pandas as pd
import scipy.io
import numpy as np
from sktime.performance_metrics.forecasting import sMAPE, smape_loss
mat = scipy.io.loadmat('COfinal.mat')
#cardio_df = pd.DataFrame(mat)

len(mat['COfinal'])
L_COfinal=[]
for i in range(len(mat['COfinal'])):
    L_COfinal.append(list(mat['COfinal'][i]))

CO_data=pd.DataFrame(L_COfinal,columns=['Lab1', 'Lab2', 'Lab3', 'Lab4', 'Temp', 'RH', 'Ref', 'Time'])
Time=CO_data['Time'].to_list()
Time=np.array(Time)
Date=pd.to_datetime(Time-719529,unit='d').round('h')
CO_data['Date'] = Date.tolist()
CO_data=CO_data.set_index('Date')
CO_data.drop('Time',axis = 1, inplace = True)

mat = scipy.io.loadmat('CO_raw.mat')
L_CO_raw=[]
for i in range(len(mat['CO_raw'])):
    L_CO_raw.append(list(mat['CO_raw'][i]))

CO_raw=pd.DataFrame(L_CO_raw,columns=['WE', 'AE', 'Temp', 'RH', 'Time'])

CO_data.insert(loc = 0,
               column = 'WE',
               value = CO_raw['WE'].to_list())
```

```

CO_data.insert(loc = 1,
               column = 'AE',
               value = CO_raw['AE'].to_list())
CO_data=CO_data.interpolate()

WE=np.array(CO_data['WE'].to_list())
AE=np.array(CO_data['AE'].to_list())
Signal=list(WE-AE)
CO_data.insert(loc = 2,
               column = 'Signal',
               value = Signal)

CO_data.head()

```

[1]:

	WE	AE	Signal	Lab1	\
Date					
2019-10-02 12:00:00	1023.970000	39.543125	984.426875	3571.592599	
2019-10-02 13:00:00	1023.970000	40.757143	983.212857	3534.556213	
2019-10-02 16:00:00	1023.970000	40.073881	983.896119	3714.254704	
2019-10-02 17:00:00	941.693433	209.369701	732.323731	2296.396571	
2019-10-03 16:00:00	488.950000	320.808571	168.141429	555.780140	

	Lab2	Lab3	Lab4	Temp	\
Date					
2019-10-02 12:00:00	2272.635909	1941.086340	618.283953	26.378438	
2019-10-02 13:00:00	2261.997806	1938.583211	609.013503	25.502791	
2019-10-02 16:00:00	2291.131285	1939.991999	679.326053	30.827910	
2019-10-02 17:00:00	1639.208945	1421.286044	497.011998	30.047164	
2019-10-03 16:00:00	390.151856	258.023564	-445.692120	29.441429	

	RH	Ref			
Date					
2019-10-02 12:00:00	58.063437	206.858886			
2019-10-02 13:00:00	59.868837	261.703907			
2019-10-02 16:00:00	49.008060	268.918360			
2019-10-02 17:00:00	51.259851	319.195427			
2019-10-03 16:00:00	52.018571	280.104590			

2.1 NO2 DATA

[2]:

```

import pandas as pd
import scipy.io
import numpy as np
mat = scipy.io.loadmat('NO2final.mat')
#cardio_df = pd.DataFrame(mat)

```

```

len(mat['N02final'])
L_N02final=[]
for i in range(len(mat['N02final'])):
    L_N02final.append(list(mat['N02final'][i]))

N02_data=pd.DataFrame(L_N02final,columns=['Lab1',  

    'Lab2','Lab3','Lab4','Temp','RH','Ref','Time'])
Time=N02_data['Time'].to_list()
Time=np.array(Time)
Date=pd.to_datetime(Time-719529,unit='d').round('h')
N02_data['Date'] = Date.tolist()
N02_data=N02_data.set_index('Date')
N02_data.drop('Time',axis = 1, inplace = True)

mat = scipy.io.loadmat('N02_raw.mat')

L_N02_raw=[]
for i in range(len(mat['N02_raw'])):
    L_N02_raw.append(list(mat['N02_raw'][i]))

N02_raw=pd.DataFrame(L_N02_raw,columns=['WE', 'AE','Temp','RH','Time'])
N02_raw.head()
N02_data.insert(loc = 0,
    column = 'WE',
    value = N02_raw['WE'].to_list())

N02_data.insert(loc = 1,
    column = 'AE',
    value = N02_raw['AE'].to_list())

N02_data=N02_data.interpolate()
WE=np.array(N02_data['WE'].to_list())
AE=np.array(N02_data['AE'].to_list())
Signal=list(WE-AE)
N02_data.insert(loc = 2,
    column = 'Signal',
    value = Signal)
N02_data.head()

```

[2]:

	WE	AE	Signal	Lab1	\
Date					
2019-10-02 12:00:00	157.696563	149.846563	7.850000	460.448301	
2019-10-02 13:00:00	0.586429	39.937143	-39.350714	1186.045510	
2019-10-02 16:00:00	17.601343	40.504328	-22.902985	1293.158435	
2019-10-02 17:00:00	190.049701	180.774030	9.275672	219.575228	

2019-10-03 16:00:00	226.502857	202.540000	23.962857	86.725922			
	Lab2	Lab3	Lab4	Temp	RH	\	
Date							
2019-10-02 12:00:00	11.259601	-54.281210	-844.685734	26.378438	58.063437		
2019-10-02 13:00:00	-64.843326	-173.647930	-1383.373409	25.502791	59.868837		
2019-10-02 16:00:00	11.194037	-180.488333	-1212.985070	30.827910	49.008060		
2019-10-02 17:00:00	-6.131452	-65.833055	-658.643041	30.047164	51.259851		
2019-10-03 16:00:00	23.809224	-16.484097	-552.026071	29.441429	52.018571		
	Ref						
Date							
2019-10-02 12:00:00	9.613288						
2019-10-02 13:00:00	15.181394						
2019-10-02 16:00:00	13.091546						
2019-10-02 17:00:00	18.506848						
2019-10-03 16:00:00	14.396420						

```
[3]: def MBE(true,pred):
    true=np.array(true)
    pred=np.array(pred)
    mbe=np.mean(true-pred)
    return mbe
def CRMSE(true,pred):
    true=np.array(true)
    pred=np.array(pred)
    crmse=np.sqrt(np.mean(((true-np.mean(true))-(pred-np.mean(pred)))**2))
    if np.std(pred)>np.std(true):
        crmse=crmse
    else:
        crmse=-crmse
    return crmse
```

2.2 SO2 DATA

```
[4]: import pandas as pd
import scipy.io
import numpy as np
mat = scipy.io.loadmat('So2final.mat')

L_So2final=[]
for i in range(len(mat['So2final'])):
    L_So2final.append(list(mat['So2final'][i]))
```

```

S02_data=pd.DataFrame(L_S02final,columns=['Lab1',  

                                         'Lab2','Lab3','Lab4','Temp','RH','Ref','Time'])  

Time=S02_data['Time'].to_list()  

Time=np.array(Time)  

Date=pd.to_datetime(Time-719529,unit='d').round('h')  

S02_data['Date'] = Date.tolist()  

S02_data=S02_data.set_index('Date')  

S02_data.drop('Time',axis = 1, inplace = True)  
  

mat = scipy.io.loadmat('S02_raw.mat')  
  

L_S02_raw=[]
for i in range(len(mat['S02_raw'])):
    L_S02_raw.append(list(mat['S02_raw'][i]))  
  

S02_raw=pd.DataFrame(L_S02_raw,columns=['WE', 'AE','Temp','RH','Time'])
S02_raw.head()
S02_data.insert(loc = 0,
                 column = 'WE',
                 value = S02_raw['WE'].to_list())  
  

S02_data.insert(loc = 1,
                 column = 'AE',
                 value = S02_raw['AE'].to_list())  
  

S02_data=S02_data.interpolate()
WE=np.array(S02_data['WE'].to_list())
AE=np.array(S02_data['AE'].to_list())
Signal=list(WE-AE)
S02_data.insert(loc = 2,
                 column = 'Signal',
                 value = Signal)  
  

S02_data.head()

```

[4] :

	WE	AE	Signal	Lab1	\
Date					
2019-10-02 12:00:00	511.980000	375.956250	136.023750	336.491001	
2019-10-02 13:00:00	511.980000	508.330476	3.649524	-339.232168	
2019-10-02 16:00:00	511.980000	442.351642	69.628358	-77.358106	
2019-10-02 17:00:00	435.447463	352.494627	82.952836	221.907307	
2019-10-03 16:00:00	358.812857	336.112857	22.700000	84.566512	

	Lab2	Lab3	Lab4	Temp	\
Date					
2019-10-02 12:00:00	395.131511	258.535708	-624.848616	26.378438	

2019-10-02 13:00:00	-68.687897	-313.588071	-619.380130	25.502791
2019-10-02 16:00:00	108.340330	-219.979244	-680.245645	30.827910
2019-10-02 17:00:00	244.067240	121.645425	-897.011152	30.047164
2019-10-03 16:00:00	77.784764	-11.080530	-1118.485141	29.441429

Date	RH	Ref
2019-10-02 12:00:00	58.063437	1.701245
2019-10-02 13:00:00	59.868837	1.495635
2019-10-02 16:00:00	49.008060	1.102313
2019-10-02 17:00:00	51.259851	1.383004
2019-10-03 16:00:00	52.018571	1.402984

3 O3 DATA

```
[5]: import pandas as pd
import scipy.io
import numpy as np
mat= scipy.io.loadmat('O3.mat')
L_O3final=[]
for i in range(len(mat['O3final_1'])):
    L_O3final.append(list(mat['O3final_1'][i]))

O3_data=pd.DataFrame(L_O3final,columns=['Lab1','Lab2','Lab3','Lab4','Temp','RH','Ref','Time'])
Time=O3_data['Time'].to_list()
Time=np.array(Time)
Date=pd.to_datetime(Time-719529,unit='d').round('h')
O3_data['Date'] = Date.tolist()
O3_data=O3_data.set_index('Date')
O3_data.drop('Time',axis = 1, inplace = True)

mat = scipy.io.loadmat('O3_raw.mat')

L_O3_raw=[]
for i in range(len(mat['O3_raw'])):
    L_O3_raw.append(list(mat['O3_raw'][i]))

O3_raw=pd.DataFrame(L_O3_raw,columns=['WE', 'AE','Temp','RH','Time'])
O3_raw.head()
O3_data.insert(loc = 0,
               column = 'WE',
               value = O3_raw['WE'].to_list())

O3_data.insert(loc = 1,
```

```

        column = 'AE',
        value = O3_raw['AE'].to_list()

O3_data=O3_data.interpolate()
WE=np.array(O3_data['WE'].to_list())
AE=np.array(O3_data['AE'].to_list())
Signal=list(WE-AE)
O3_data.insert(loc = 2,
               column = 'Signal',
               value = Signal)
O3_data.head()

```

[5]:

	WE	AE	Signal	Lab1	\
Date					
2019-10-02 12:00:00	229.020000	232.625625	-3.605625	621.625704	
2019-10-02 13:00:00	284.284762	207.033333	77.251429	1835.510709	
2019-10-02 16:00:00	199.551940	131.729254	67.822687	2195.698656	
2019-10-02 17:00:00	234.016418	225.462090	8.554328	386.251890	
2019-10-03 16:00:00	234.671429	229.311429	5.360000	200.100352	

	Lab2	Lab3	Lab4	Temp	\
Date					
2019-10-02 12:00:00	67.962148	-12.822516	-987.063805	26.378438	
2019-10-02 13:00:00	293.683054	159.572059	-1331.518231	25.502791	
2019-10-02 16:00:00	615.567710	379.302767	-893.337914	30.827910	
2019-10-02 17:00:00	108.049071	34.461730	-696.227364	30.047164	
2019-10-03 16:00:00	122.550133	72.885162	-587.216200	29.441429	

	RH	Ref			
Date					
2019-10-02 12:00:00	58.063437	52.449447			
2019-10-02 13:00:00	59.868837	50.464425			
2019-10-02 16:00:00	49.008060	37.972231			
2019-10-02 17:00:00	51.259851	33.446343			
2019-10-03 16:00:00	52.018571	33.071706			

[6]: C0_data.shape

[6]: (6021, 10)

[7]: C0_data.describe()

[7]:

	WE	AE	Signal	Lab1	Lab2	\
count	6021.000000	6021.000000	6021.000000	6021.000000	6021.000000	
mean	489.868443	325.037641	164.830802	460.445775	351.621354	
std	147.893193	22.357098	145.524186	316.990546	298.778134	

min	269.495048	38.934737	-59.665104	-76.574116	-103.290653
25%	399.638542	316.795833	72.162250	246.967288	162.828332
50%	438.460648	328.552083	114.690455	375.497026	248.319472
75%	515.414062	339.209804	192.807355	573.671805	413.089960
max	1023.970000	377.300333	985.035263	3967.688054	2345.947440

	Lab3	Lab4	Temp	RH	Ref
count	6021.000000	6021.000000	6021.000000	6021.000000	6021.000000
mean	229.061278	-539.062229	18.258360	67.305000	663.749977
std	298.561008	295.565104	7.601853	18.454202	677.801714
min	-211.680627	-912.704297	1.035000	11.428168	-1095.570697
25%	41.157070	-728.592009	12.408678	53.576341	285.212286
50%	124.692402	-638.382797	16.872568	72.236338	466.332961
75%	284.461578	-467.551093	22.849585	82.938548	759.000870
max	1941.711657	854.177980	44.748056	93.012486	6836.814023

4 DATA VISUALIZATION

```
[8]: import seaborn as sns
import matplotlib.pyplot as plt
co_data=CO_data
co_data=co_data[(co_data[co_data.columns] >= 0).all(axis=1)]
#CO_data=CO_data.resample('D').mean()
co_data=co_data.dropna()
co_data.shape

L1=co_data['Lab1'].to_list()
L2=co_data['Lab2'].to_list()
L3=co_data['Lab3'].to_list()
L4=co_data['Lab4'].to_list()
Ref=co_data['Ref'].to_list()
Conc=L1+L2+L3+L4+Ref
model1=['Lab1' for i in range(len(L1))]
model2=['Lab2' for i in range(len(L1))]
model3=['Lab3' for i in range(len(L1))]
model4=['Lab4' for i in range(len(L1))]
Ref=['Ref' for i in range(len(L1))]
model=model1+model2+model3+model4+Ref

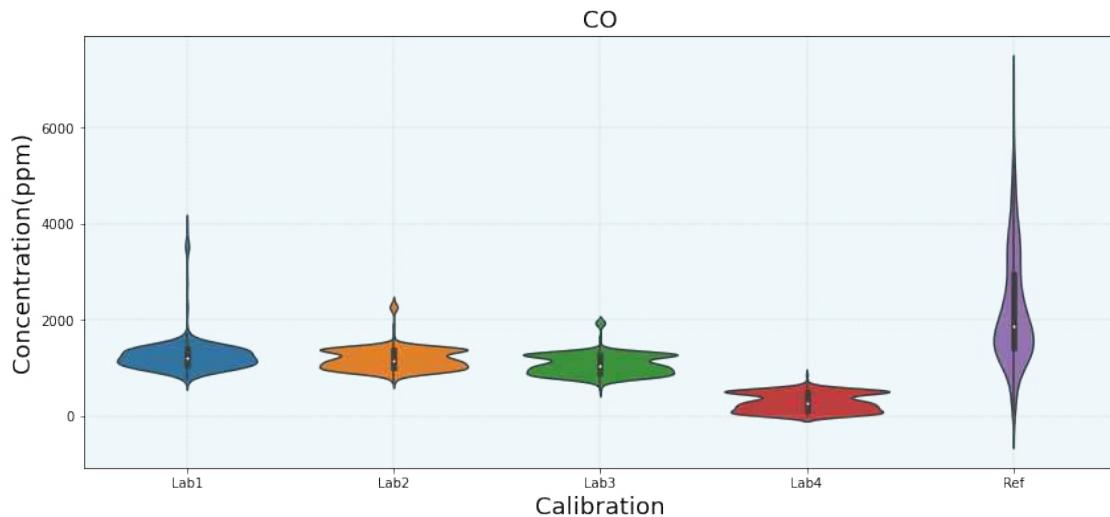
data=[[model[i],Conc[i]] for i in range(len(model))]
data=pd.DataFrame(data=data,columns=['model','Concentration(ppm)'])
fig= plt.figure(figsize=(14,6))
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.2)
```

```

sns.violinplot(x="model", y="Concentration(ppm)", data=data)
plt.grid(linestyle='-.', linewidth=0.3)
plt.xlabel('Calibration', fontsize=18)
plt.ylabel('Concentration(ppm)', fontsize=18)
plt.title('CO', fontsize=18)

```

[8]: Text(0.5, 1.0, 'CO')



```

[9]: no2_data=N02_data
no2_data=no2_data.drop(['Lab4'],axis=1)
no2_data=no2_data[(no2_data[no2_data.columns] >= 0).all(axis=1)]
#CO_data=CO_data.resample('D').mean()
no2_data=no2_data.dropna()
L1=no2_data['Lab1'].to_list()
for i in range(len(L1)):
    if L1[i]>400:
        L1[i]=np.mean(np.array(L1))

L2=no2_data['Lab2'].to_list()
L3=no2_data['Lab3'].to_list()
for i in range(len(L1)):
    if L3[i]<10:
        L3[i]=np.mean(np.array(L3))
#L4=no2_data['Lab4'].to_list()
Ref=no2_data['Ref'].to_list()
Conc=L1+L2+L3+Ref
model1=['Lab1' for i in range(len(L1))]
model2=['Lab2' for i in range(len(L1))]
model3=['Lab3' for i in range(len(L1))]

```

```

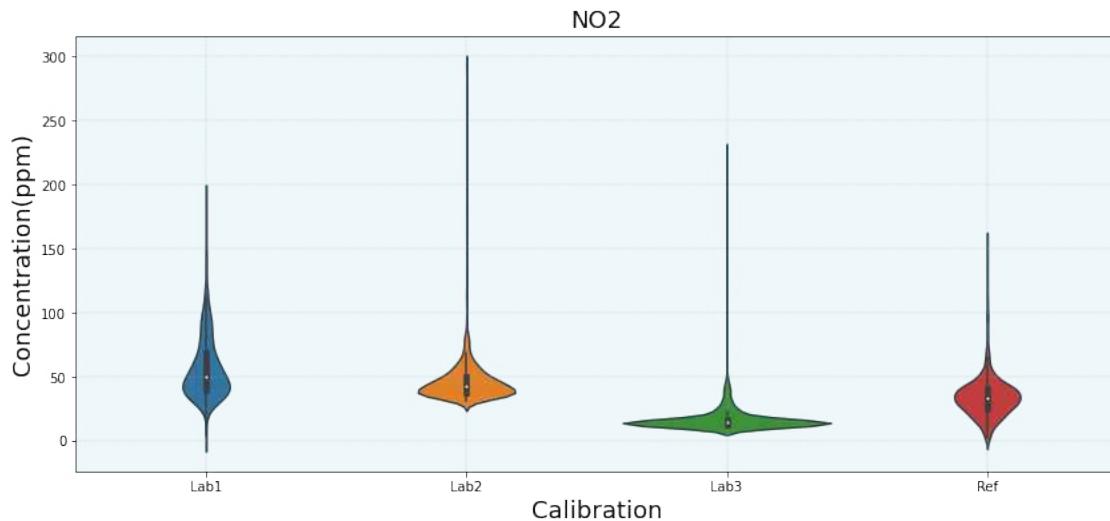
#model4=['Lab4' for i in range(len(L1))]
print(min(L3))
Ref=['Ref' for i in range(len(L1))]
model=model1+model2+model3+Ref

data=[[model[i],Conc[i]] for i in range(len(model))]
data=pd.DataFrame(data=data,columns=['model','Concentration(ppm)'])
fig= plt.figure(figsize=(14,6))
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.2)
sns.violinplot(x="model", y="Concentration(ppm)", data=data)
plt.grid(linestyle='-.', linewidth=0.3)
plt.xlabel('Calibration', fontsize=18)
plt.ylabel('Concentration(ppm)', fontsize=18)
plt.title('NO2', fontsize=18)

```

10.001758318900059

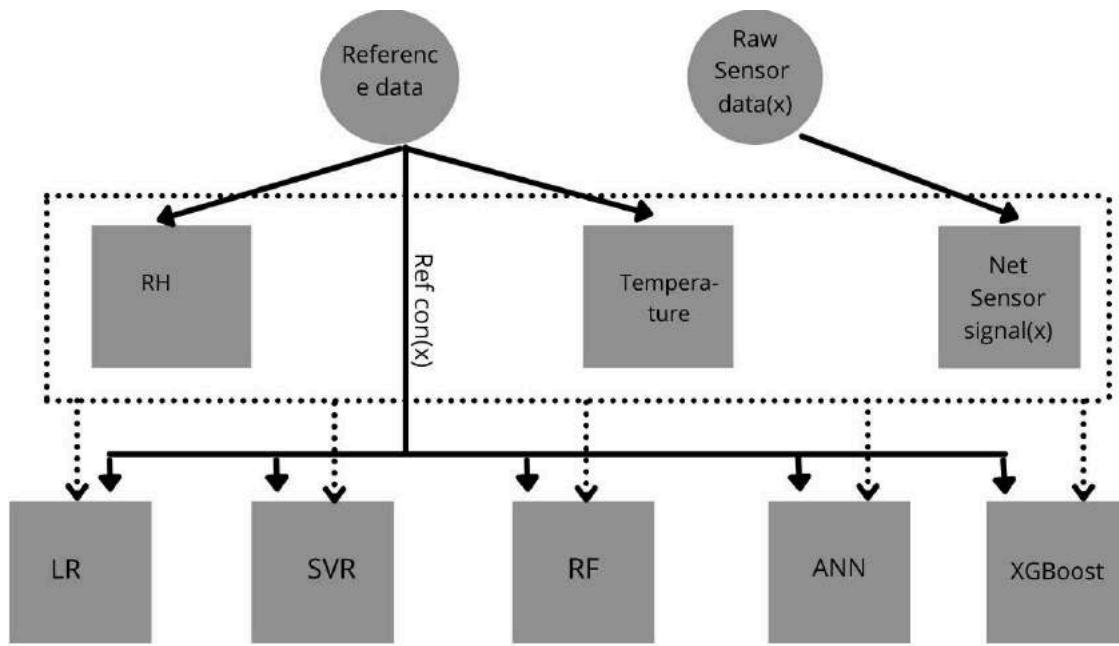
[9]: Text(0.5, 1.0, 'NO2')



5 Model Architectures

[10]: import matplotlib.pyplot as plt
fig= plt.figure(figsize=(10,6))
from IPython.display import Image
Image('Architecture2.png')

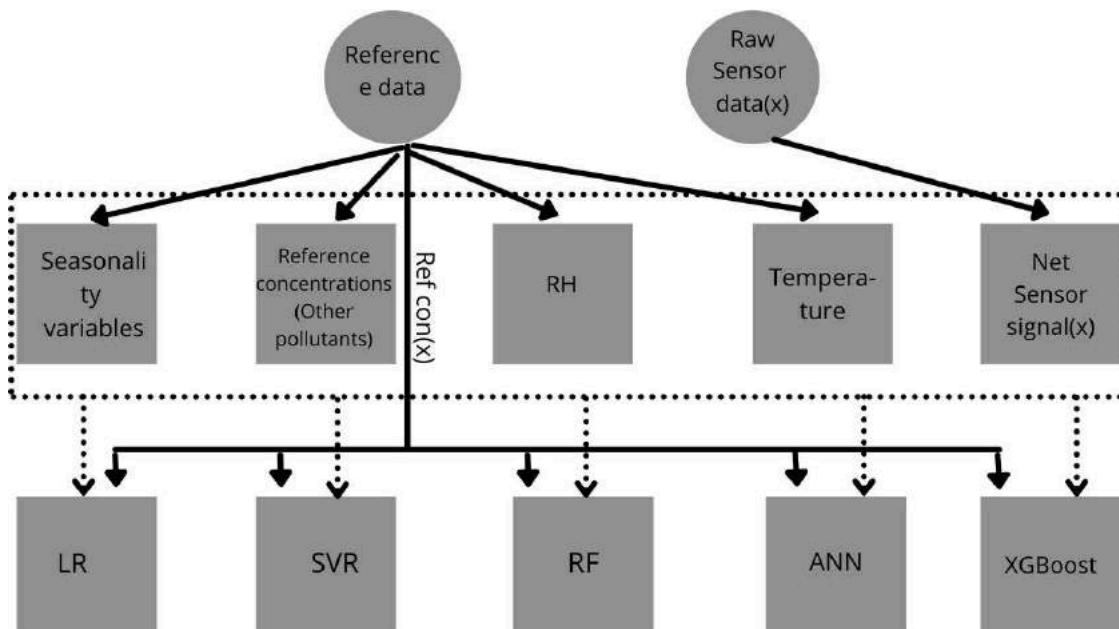
[10] :



<Figure size 720x432 with 0 Axes>

[138] : `Image('Architecture.png')`

[138] :



6 CO CALIBRATION

```
[12]: CO_data['Month']=CO_data.index.month
CO_data['Day_of_week']=CO_data.index.dayofweek
CO_data['Day']=CO_data.index.day
CO_data['Hour']=CO_data.index.hour
CO_data['Ref_N02']=N02_data['Ref'].to_list()
CO_data['Ref_S02']=S02_data['Ref'].to_list()
CO_data['Ref_O3']=O3_data['Ref'].to_list()
CO_Data=CO_data[['Signal','Ref_N02','Ref_S02','Ref_O3','Lab1','Temp','RH','Ref','Month','Day_of_week','Day','Hour']]
CO_Data=CO_Data[(CO_Data[CO_Data.columns] >= 0).all(axis=1)]
#CO_data=CO_data.resample('D').mean()
CO_Data=CO_Data.dropna()
CO_Data.shape
```

```
[12]: (5895, 12)
```

6.1 Model 1: Linear Regression

```
[13]: from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error as mae
import sklearn.metrics as sm
import matplotlib.pyplot as plt
X=CO_Data[['Signal','Ref_N02','Ref_S02','Ref_O3','Temp','RH','Month','Day_of_week','Day','Hour']]
y=CO_Data['Ref']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
len(y_test)
```

```
[13]: 1179
```

```
[14]: lr = LinearRegression()
model = lr.fit(X_train, y_train)
pred = model.predict(X_test)
lab1=CO_Data['Lab1'].to_list()[len(y_train):]

index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index=index)
Y_test

Pred=pd.Series(pred,index=index)
Lab1=pd.Series(lab1,index=index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round(smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
```

```

RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_lr_C0=sMAPE_lr
RMSE_lr_C0=round(RMSE_lr/np.mean(np.array(y_test)),2)
Pearson_lr_C0=Pearson_lr
sMAPE_lab_C0=sMAPE_lab
RMSE_lab_C0=round(RMSE_lab/np.mean(np.array(lab1)),2)
Pearson_lab_C0=Pearson_lab
R2_lr_C0=round(sm.r2_score(y_test, pred), 2)
R2_lab_C0=round(sm.r2_score(y_test, lab1), 2)
RMSE_Lr_C0=RMSE_lr
RMSE_Lab_C0=RMSE_lab

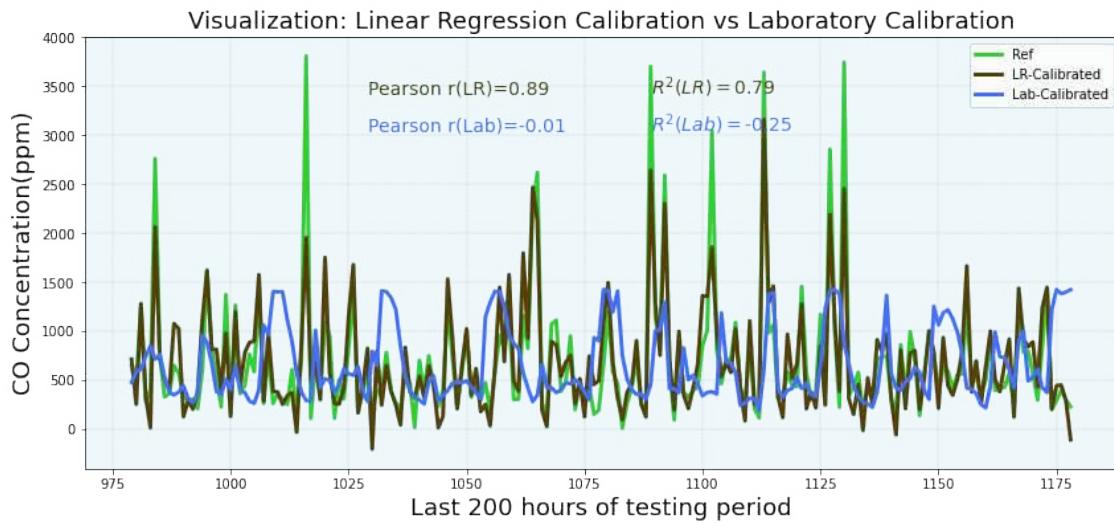
A=len(y_test)-200
D=max(y_test[A:])-0.2*max(y_test[A:])
C=max(y_test[A:])-0.1*max(y_test[A:])
B=A+120

```

```

[15]: fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.2)
plt.plot(index[A:],y_test[A:], color='limegreen', linewidth=3)
plt.plot(index[A:],pred[A:], color='#513e00', linewidth=3)
plt.plot(index[A:],lab1[A:], color='#426eff', linewidth=3)
plt.legend(['Ref', 'LR-Calibrated', 'Lab-Calibrated'], loc = 2, bbox_to_anchor=(0.85,1))
plt.ylabel('CO Concentration(ppm)', fontsize=18)
plt.text(B-10, C, r'$R^2(LR)=$'+str(R2_lr_C0), fontsize = 14, color="#513e00")
plt.text(B-10, D, r'$R^2(Lab)=$' +str(R2_lab_C0), fontsize = 14, color="#426eff")
plt.text(B-70, C, 'Pearson r(LR)='+str(Pearson_lr), fontsize = 14, color="#513e00")
plt.text(B-70, D, 'Pearson r(Lab)='+str(Pearson_lab), fontsize = 14, color="#426eff")
plt.xlabel('Last 200 hours of testing period',fontsize=18)
plt.title('Visualization: Linear Regression Calibration vs Laboratory Calibration',fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



```
[16]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred), 2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_test, pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))
pred_lr=pred
MBE_LR_CO=MBE(pred,y_test)/np.std(y_test)
CRMSE_LR_CO=CRMSE(y_test,pred)/np.std(y_test)
MBE_LAB_CO=MBE(lab1,y_test)/np.std(y_test)
CRMSE_LAB_CO=CRMSE(y_test,lab1)/np.std(y_test)
```

Regressor model performance:
 Mean absolute error(MAE) = 196.29
 Mean squared error(MSE) = 109535.34
 Median absolute error = 125.74
 Explain variance score = 0.79
 R2 score = 0.79

6.2 Model 2 : Support Vector Regression (SVR)

```
[17]: from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
regressor = SVR(kernel = 'linear')
regressor.fit(X_train, y_train)
pred = regressor.predict(X_test)
for i in range(len(Pred)):
    if pred[i]<0:
        pred[i]=np.mean(np.array(pred))
pred_svr=pred
```



```
[18]: lab1=CO_Data['Lab1'].to_list()[len(y_train):]
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index =index)
Y_test
Pred=pd.Series(pred,index =index)

Lab1=pd.Series(lab1,index =index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_svr_CO=sMAPE_lr
RMSE_svr_CO=round(RMSE_lr/np.mean(np.array(y_test)),2)
Pearson_svr_CO=Pearson_lr
R2_svr_CO=round(sm.r2_score(y_test, pred), 2)
RMSE_Svr_CO=RMSE_lr
```



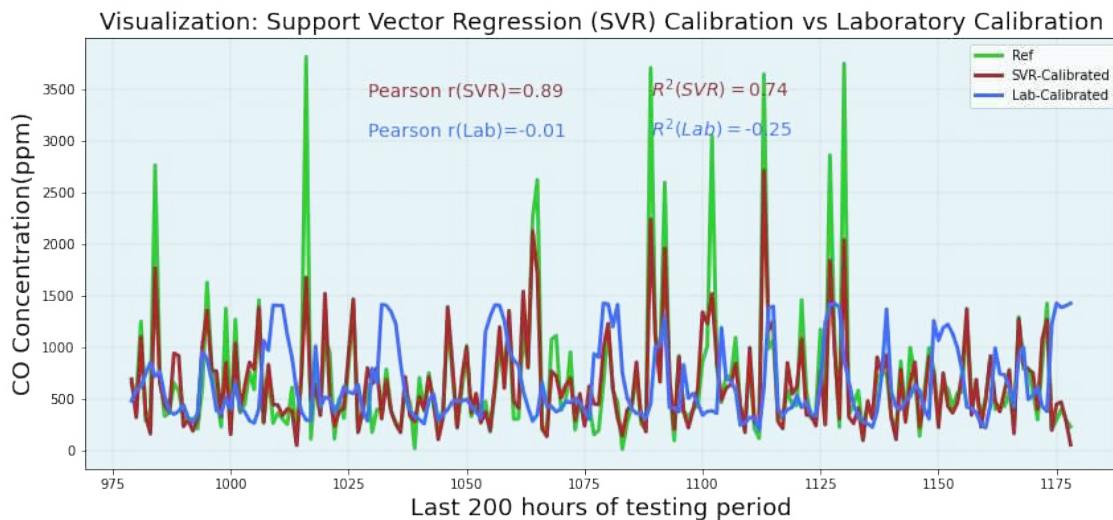
```
[19]: A=len(y_test)-200
D=max(y_test[A:])-0.2*max(y_test[A:])
C=max(y_test[A:])-0.1*max(y_test[A:])
B=A+120

fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.plot(index[A:],y_test[A:], color='limegreen',linewidth=3)
plt.plot(index[A:],pred[A:], color='brown',linewidth=3)
plt.plot(index[A:],lab1[A:], color='#426eff',linewidth=3)
plt.legend(['Ref', 'SVR-Calibrated', 'Lab-Calibrated'], loc = 2, bbox_to_anchor=(0.85,1))
```

```

plt.ylabel('CO Concentration(ppm)', fontsize=18)
plt.text(B-10, C, r'$R^2(SVR)=' + str(R2_svr_CO), fontsize = 14, color='brown')
plt.text(B-10, D, r'$R^2(Lab)=' + str(R2_lab_CO), fontsize = 14, color='brown')
plt.text(B-70, C, 'Pearson r(SVR)=' + str(Pearson_lr), fontsize = 14, color='brown')
plt.text(B-70, D, 'Pearson r(Lab)=' + str(Pearson_lab), fontsize = 14, color='brown')
plt.xlabel('Last 200 hours of testing period', fontsize=18)
plt.title('Visualization: Support Vector Regression (SVR) Calibration vs Laboratory Calibration', fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



```

[20]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred), 2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred), 2))
print("Explained variance score =", round(sm.explained_variance_score(y_test, pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))
pred_svr=pred
MBE_SVR_CO=MBE(pred,y_test)/np.std(y_test)
CRMSE_SVR_CO=CRMSE(y_test,pred)/np.std(y_test)

```

Regressor model performance:

```

Mean absolute error(MAE) = 182.71
Mean squared error(MSE) = 138275.78
Median absolute error = 97.84
Explain variance score = 0.74
R2 score = 0.74

```

6.3 Model 3 : Random Forest

```
[21]: from sklearn.ensemble import RandomForestRegressor

# create regressor object
regressor = RandomForestRegressor(n_estimators = 500, random_state = 0)

# fit the regressor with x and y data
regressor.fit(X_train, y_train)
```

```
[21]: RandomForestRegressor(n_estimators=500, random_state=0)
```

```
[22]: features_C0=regressor.feature_importances_
pred = regressor.predict(X_test)
lab1=C0_Data['Lab1'].to_list()[:len(y_train):]
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index =index)
Y_test
Pred=pd.Series(pred,index =index)
Lab1=pd.Series(lab1,index =index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_rf_C0=sMAPE_lr
RMSE_rf_C0=round(RMSE_lr/np.mean(np.array(y_test)),2)
Pearson_rf_C0=Pearson_lr
R2_rf_C0=round(sm.r2_score(y_test, pred), 2)
RMSE_Rf_C0=RMSE_lr

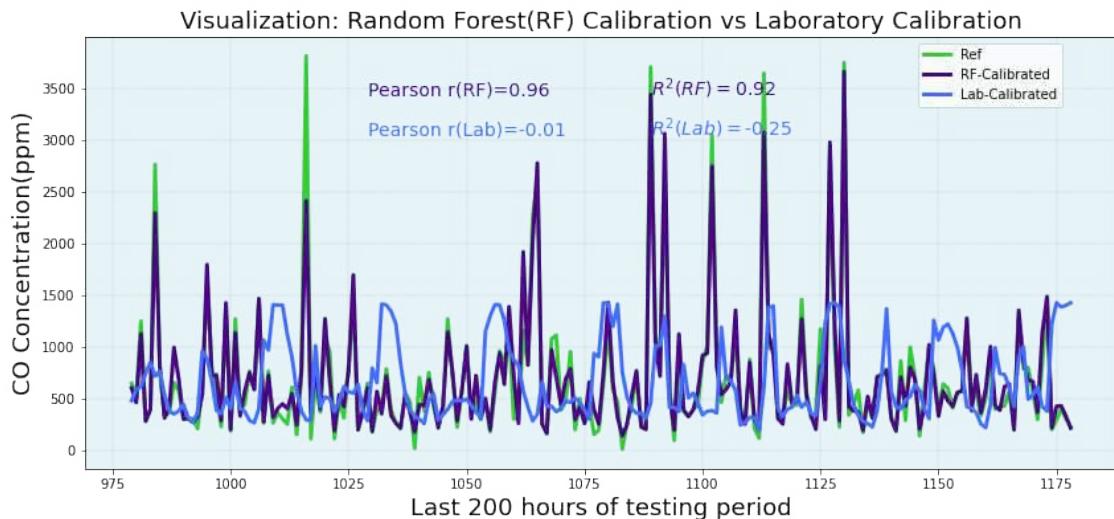
A=len(y_test)-200
D=max(y_test[A:])-0.2*max(y_test[A:])
C=max(y_test[A:])-0.1*max(y_test[A:])
B=A+120
```

```
[23]: fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
```

```

ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.plot(index[A:],y_test[A:], color='limegreen', linewidth=3)
plt.plot(index[A:],pred[A:], color='indigo', linewidth=3)
plt.plot(index[A:],lab1[A:], color='#426eff', linewidth=3)
plt.legend(['Ref', 'RF-Calibrated', 'Lab-Calibrated'], loc = 2, bbox_to_anchor=(0.8,1))
plt.ylabel('CO Concentration(ppm)', fontsize=18)
plt.text(B-10, C, r'$R^2(RF)=$'+str(R2_rf_CO), fontsize = 14, color='indigo')
plt.text(B-10, D, r'$R^2(Lab)=$'+str(R2_lab_CO), fontsize = 14, color='indigo')
plt.text(B-70, C, 'Pearson r(RF)=' + str(Pearson_lr), fontsize = 14, color='indigo')
plt.text(B-70, D, 'Pearson r(Lab)=' + str(Pearson_lab), fontsize = 14, color='indigo')
plt.xlabel('Last 200 hours of testing period', fontsize=18)
plt.title('Visualization: Random Forest(RF) Calibration vs Laboratory Calibration', fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



```

[24]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred), 2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred), 2))

```

```

print("Explain variance score =", round(sm.explained_variance_score(y_test, pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))
pred_rf=pred
MBE_RF_CO=MBE(pred,y_test)/np.std(y_test)
CRMSE_RF_CO=CRMSE(y_test,pred)/np.std(y_test)

```

Regressor model performance:

```

Mean absolute error(MAE) = 107.2
Mean squared error(MSE) = 39184.72
Median absolute error = 53.64
Explain variance score = 0.92
R2 score = 0.92

```

6.4 Model 5: ANN

```
[25]: from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
from sklearn.preprocessing import StandardScaler
model = Sequential()
model.add(Dense(3, input_shape = (10,),kernel_initializer='normal', activation='linear'))
model.add(Dense(128,kernel_initializer='normal', activation= 'relu'))
model.add(Dense(128, kernel_initializer='normal',activation= 'relu'))
model.add(Dense(100, kernel_initializer='normal',activation= 'relu'))
model.add(Dense(1,kernel_initializer='normal',activation='linear',))
sgd = optimizers.Adam(learning_rate=0.01)

model.compile(optimizer = sgd, loss = 'mean_squared_error', metrics= ['mse','mae'])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 3)	33
dense_1 (Dense)	(None, 128)	512
dense_2 (Dense)	(None, 128)	16512
dense_3 (Dense)	(None, 100)	12900
dense_4 (Dense)	(None, 1)	101

```
=====
Total params: 30,058
Trainable params: 30,058
Non-trainable params: 0
-----
```

```
[26]: scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled=scaler.transform(X_train)
X_test_scaled=scaler.transform(X_test)
hist=model.fit(X_train_scaled, y_train, batch_size= 10, epochs=200, verbose=0) #, validation_split=0.2
```

```
[27]: train_pred = model.predict(X_train_scaled)
test_pred = model.predict(X_test_scaled)
pred=[]
for i in range(len(test_pred)):
    pred.append(sum(list(test_pred[i])))

lab1=CO_Data['Lab1'].to_list()[len(y_train):]
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index =index)
Y_test
Pred=pd.Series(pred,index =index)
Lab1=pd.Series(lab1,index =index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_ann_CO=sMAPE_lr
RMSE_ann_CO=round(RMSE_lr/np.mean(np.array(y_test)),2)
Pearson_ann_CO=Pearson_lr
R2_ann_CO=round(sm.r2_score(y_test, pred), 2)
RMSE_Ann_CO=RMSE_lr

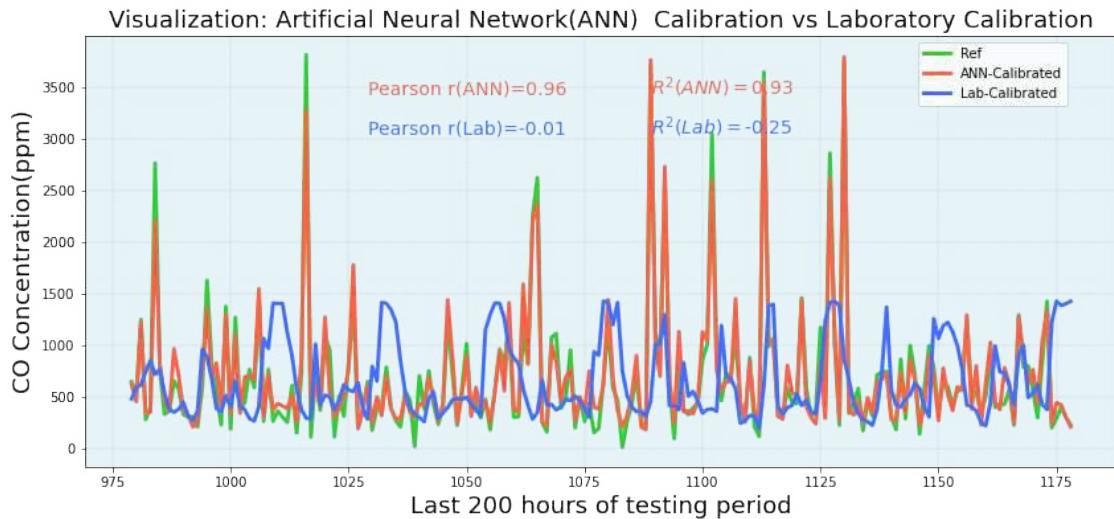
A=len(y_test)-200
D=max(y_test[A:])-0.2*max(y_test[A:])
C=max(y_test[A:])-0.1*max(y_test[A:])
B=A+120
```

```
[28]: fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
```

```

ax.patch.set_alpha(0.3)
plt.plot(index[A:],y_test[A:], color='limegreen', linewidth=3)
plt.plot(index[A:],pred[A:], color='tomato', linewidth=3)
plt.plot(index[A:],lab1[A:], color='#426eff', linewidth=3)
plt.legend(['Ref', 'ANN-Calibrated', 'Lab-Calibrated'], loc = 2, bbox_to_anchor=(0.8,1))
plt.ylabel('CO Concentration(ppm)', fontsize=18)
plt.text(B-10, C, r'$R^2(ANN)=$'+str(R2_ann_CO), fontsize = 14, color='tomato')
plt.text(B-10, D, r'$R^2(Lab)=$'+str(R2_lab_CO), fontsize = 14, color='#426eff')
plt.text(B-70, C, 'Pearson r(ANN)='+str(Pearson_lr), fontsize = 14, color='tomato')
plt.text(B-70, D, 'Pearson r(Lab)='+str(Pearson_lab), fontsize = 14, color='#426eff')
plt.xlabel('Last 200 hours of testing period', fontsize=18)
plt.title('Visualization: Artificial Neural Network(ANN) Calibration vs Laboratory Calibration', fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



```

[29]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred), 2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred), 2))

```

```

print("Explain variance score =", round(sm.explained_variance_score(y_test,
    pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))
pred_ann=pred
MBE_ANN_CO=MBE(pred,y_test)/np.std(y_test)
CRMSE_ANN_CO=CRMSE(y_test,pred)/np.std(y_test)

```

Regressor model performance:

```

Mean absolute error(MAE) = 117.89
Mean squared error(MSE) = 38779.88
Median absolute error = 71.6
Explain variance score = 0.93
R2 score = 0.93

```

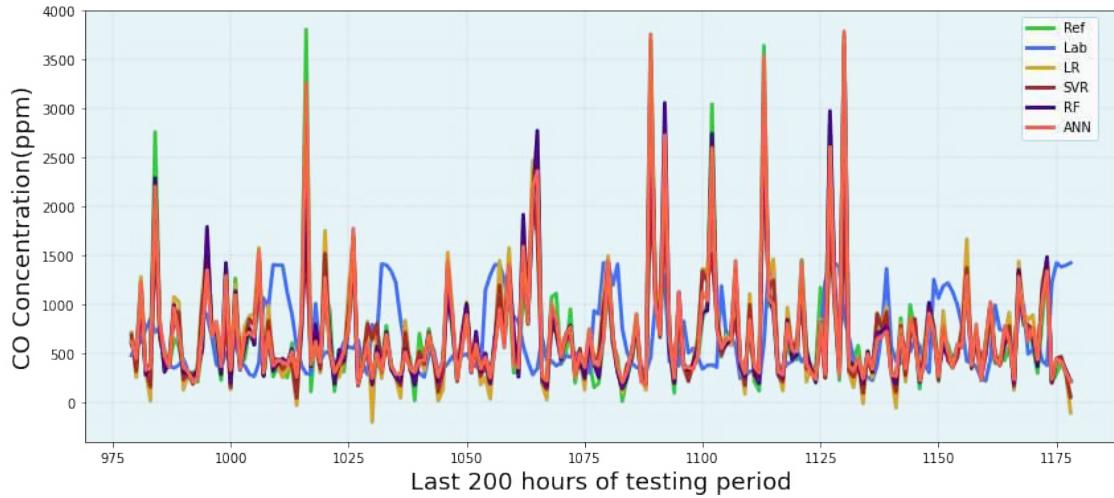
```

[30]: A=len(y_test)-200

fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.plot(index[A:],y_test[A:], color='limegreen', linewidth=3)
plt.plot(index[A:],lab1[A:], color="#426eff", linewidth=3)
plt.plot(index[A:],pred_lr[A:], color='goldenrod', linewidth=3)
plt.plot(index[A:],pred_svr[A:], color='brown', linewidth=3)
plt.plot(index[A:],pred_rf[A:], color='indigo', linewidth=3)
plt.plot(index[A:],pred_ann[A:], color='tomato', linewidth=3)

plt.xlabel('Last 200 hours of testing period', fontsize=18)
plt.ylabel('CO Concentration(ppm)', fontsize=18)
plt.legend(['Ref', 'Lab', 'LR','SVR','RF','ANN'], loc = 2, bbox_to_anchor = (0.
    -9,1))
# plt.title('CO Sensor', fontsize=18 )
plt.grid(linestyle='-.', linewidth=0.3)

```



7 Model 6: XGBoost

```
[31]: from xgboost import XGBRegressor
from numpy import absolute
from pandas import read_csv
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
# create an xgboost regression model
#n_estimators=10000, max_depth=5, eta=0.01, subsample=0.9, colsample_bytree=0.
#→4, alpha=10
model = XGBRegressor(n_estimators=10000, max_depth=5, eta=0.01, subsample=0.9,
                      colsample_bytree=0.4, alpha=10)
model.fit(X_train, y_train)
```

```
[31]: XGBRegressor(alpha=10, base_score=0.5, booster='gbtree', colsample_bylevel=1,
                  colsample_bynode=1, colsample_bytree=0.4, eta=0.01, gamma=0,
                  gpu_id=-1, importance_type='gain', interaction_constraints='',
                  learning_rate=0.00999999978, max_delta_step=0, max_depth=5,
                  min_child_weight=1, missing=nan, monotone_constraints='()',
                  n_estimators=10000, n_jobs=4, num_parallel_tree=1, random_state=0,
                  reg_alpha=10, reg_lambda=1, scale_pos_weight=1, subsample=0.9,
                  tree_method='exact', validate_parameters=1, verbosity=None)
```

```
[32]: pred = model.predict(X_test)
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index =index)
Y_test
```

```

Pred=pd.Series(pred,index =index)
Lab1=pd.Series(lab1,index =index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_xgb_CO=sMAPE_lr
RMSE_xgb_CO=RMSE_lr/np.mean(np.array(y_test))
Pearson_xgb_CO=Pearson_lr
R2_xgb_CO=round(sm.r2_score(y_test, pred), 2)
RMSE_Xgb_CO=RMSE_lr

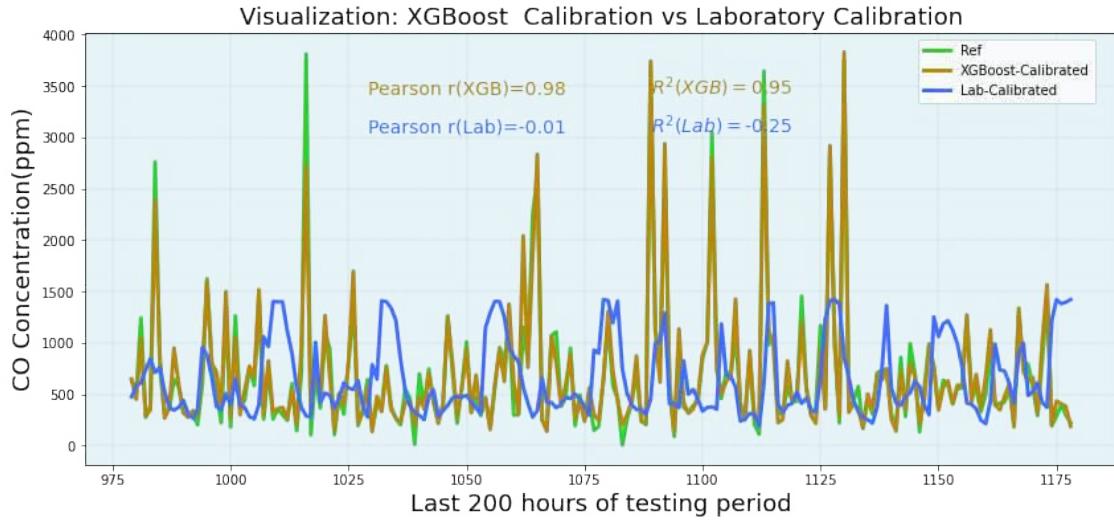
A=len(y_test)-200
D=max(y_test[A:])-0.2*max(y_test[A:])
C=max(y_test[A:])-0.1*max(y_test[A:])
B=A+120

```

```

[33]: fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.plot(index[A:],y_test[A:], color='limegreen',linewidth=3)
plt.plot(index[A:],pred[A:], color='darkgoldenrod',linewidth=3)
plt.plot(index[A:],lab1[A:], color="#426eff",linewidth=3)
plt.legend(['Ref', 'XGBoost-Calibrated', 'Lab-Calibrated'], loc = 2,
   bbox_to_anchor = (0.8,1))
plt.ylabel('CO Concentration(ppm)',fontsize=18)
plt.text(B-10, C,r'$R^2(XGB)=$'+str(R2_xgb_CO) , fontsize = 14,
   color='darkgoldenrod')
plt.text(B-10, D,r'$R^2(Lab)=$'+str(R2_lab_CO) , fontsize = 14,
   color="#426eff")
plt.text(B-70, C, 'Pearson r(XGB)='+str(Pearson_lr), fontsize = 14,
   color='darkgoldenrod')
plt.text(B-70, D, 'Pearson r(Lab)='+str(Pearson_lab), fontsize = 14,
   color="#426eff")
plt.xlabel('Last 200 hours of testing period',fontsize=18)
plt.title('Visualization: XGBoost Calibration vs Laboratory
   Calibration',fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



```
[34]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred), 2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_test, pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))
pred_xgb=pred
MBE_XGB_CO=MBE(pred,y_test)/np.std(y_test)
CRMSE_XGB_CO=CRMSE(y_test,pred)/np.std(y_test)
```

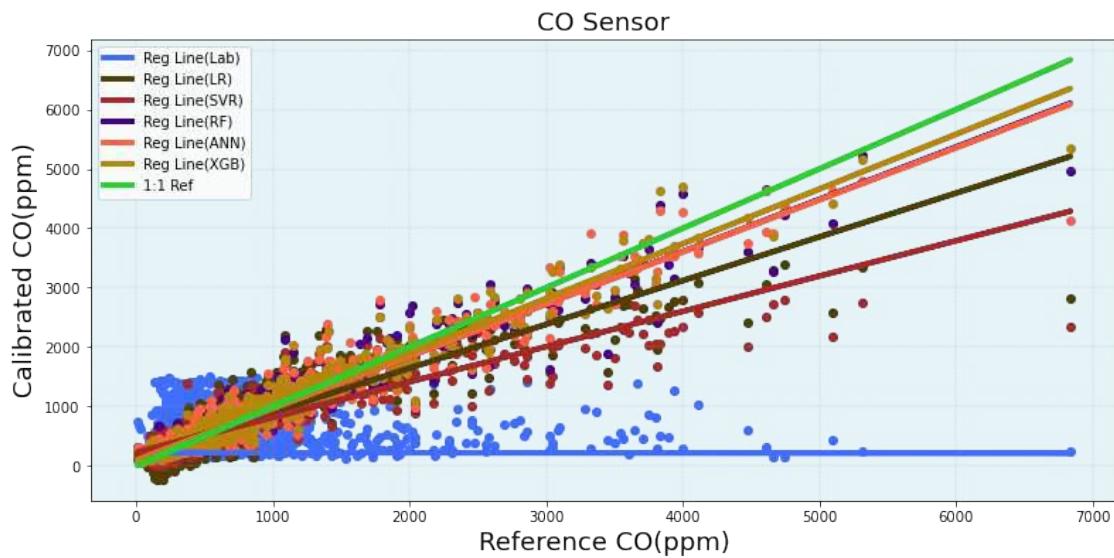
Regressor model performance:
 Mean absolute error(MAE) = 92.14
 Mean squared error(MSE) = 25190.83
 Median absolute error = 49.79
 Explain variance score = 0.95
 R2 score = 0.95

```
[35]: fig= plt.figure(figsize=(13,6))
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
m0, b0 = np.polyfit(np.array(y_test), 0.4*np.array(lab1), 1)
plt.plot(np.array(y_test), m0*np.array(y_test) + b0,color="#426eff", linewidth=4)
m1, b1 = np.polyfit(np.array(y_test), np.array(pred_lr), 1)
plt.plot(np.array(y_test), m1*np.array(y_test) + b1,color="#513e00", linewidth=4)
```

```

m2, b2 = np.polyfit(np.array(y_test), np.array(pred_svr), 1)
plt.plot(np.array(y_test), m2*np.array(y_test) + b2,color='brown',linewidth=4)
m3, b3 = np.polyfit(np.array(y_test), np.array(pred_rf), 1)
plt.plot(np.array(y_test), m3*np.array(y_test) + b3,color='indigo',linewidth=4)
m4, b4 = np.polyfit(np.array(y_test), np.array(pred_ann), 1)
plt.plot(np.array(y_test), m4*np.array(y_test) + b4,color='tomato',linewidth=4)
m5, b5 = np.polyfit(np.array(y_test), np.array(pred_xgb), 1)
plt.plot(np.array(y_test), m5*np.array(y_test) + b5,color='darkgoldenrod',linewidth=4)
plt.scatter(np.array(y_test),np.array(lab1),color="#426eff")
plt.scatter(np.array(y_test),np.array(pred_lr),color="#513e00")
plt.scatter(np.array(y_test),np.array(pred_svr),color='brown')
plt.scatter(np.array(y_test),np.array(pred_rf),color='indigo')
plt.scatter(np.array(y_test),np.array(pred_ann),color='tomato')
plt.scatter(np.array(y_test),np.array(pred_xgb),color='darkgoldenrod')
ax.plot(y_test,y_test, c ="limegreen",linewidth=4)
plt.xlabel('Reference CO(ppm)',fontsize=18)
plt.ylabel('Calibrated CO(ppm)',fontsize=18)
plt.legend(['Reg Line(Lab)', 'Reg Line(LR)', 'Reg Line(SVR)', 'Reg Line(RF)', 'Reg Line(ANN)', 'Reg Line(XGB)', '1:1 Ref'], loc = 2, bbox_to_anchor = (0,1))
plt.title('CO Sensor', fontsize=18 )
plt.grid(linestyle='-.', linewidth=0.3)

```



8 NO2 Calibration

```
[36]: import pandas as pd
import scipy.io
import numpy as np
mat = scipy.io.loadmat('NO2final.mat')
#cardio_df = pd.DataFrame(mat)

len(mat['NO2final'])
L_NO2final=[]
for i in range(len(mat['NO2final'])):
    L_NO2final.append(list(mat['NO2final'][i]))

NO2_data=pd.DataFrame(L_NO2final,columns=['Lab1',_
                                         'Lab2','Lab3','Lab4','Temp','RH','Ref','Time'])
Time=NO2_data['Time'].to_list()
Time=np.array(Time)
Date=pd.to_datetime(Time-719529,unit='d').round('h')
NO2_data['Date'] = Date.tolist()
NO2_data=NO2_data.set_index('Date')
NO2_data.drop('Time',axis = 1, inplace = True)

mat = scipy.io.loadmat('NO2_raw.mat')

L_NO2_raw=[]
for i in range(len(mat['NO2_raw'])):
    L_NO2_raw.append(list(mat['NO2_raw'][i]))

NO2_raw=pd.DataFrame(L_NO2_raw,columns=['WE', 'AE', 'Temp', 'RH', 'Time'])
NO2_raw.head()
NO2_data.insert(loc = 0,
                column = 'WE',
                value = NO2_raw['WE'].to_list())

NO2_data.insert(loc = 1,
                column = 'AE',
                value = NO2_raw['AE'].to_list())

NO2_data=NO2_data.interpolate()
WE=np.array(NO2_data['WE'].to_list())
AE=np.array(NO2_data['AE'].to_list())
Signal=list(WE-AE)
NO2_data.insert(loc = 2,
                column = 'Signal',
                value = Signal)
NO2_data.head()
```

[36] :

	WE	AE	Signal	Lab1	\	
Date						
2019-10-02 12:00:00	157.696563	149.846563	7.850000	460.448301		
2019-10-02 13:00:00	0.586429	39.937143	-39.350714	1186.045510		
2019-10-02 16:00:00	17.601343	40.504328	-22.902985	1293.158435		
2019-10-02 17:00:00	190.049701	180.774030	9.275672	219.575228		
2019-10-03 16:00:00	226.502857	202.540000	23.962857	86.725922		
	Lab2	Lab3	Lab4	Temp	RH	\
Date						
2019-10-02 12:00:00	11.259601	-54.281210	-844.685734	26.378438	58.063437	
2019-10-02 13:00:00	-64.843326	-173.647930	-1383.373409	25.502791	59.868837	
2019-10-02 16:00:00	11.194037	-180.488333	-1212.985070	30.827910	49.008060	
2019-10-02 17:00:00	-6.131452	-65.833055	-658.643041	30.047164	51.259851	
2019-10-03 16:00:00	23.809224	-16.484097	-552.026071	29.441429	52.018571	
	Ref					
Date						
2019-10-02 12:00:00	9.613288					
2019-10-02 13:00:00	15.181394					
2019-10-02 16:00:00	13.091546					
2019-10-02 17:00:00	18.506848					
2019-10-03 16:00:00	14.396420					

[37] : NO2_data.shape

[37] : (6021, 10)

[38] : NO2_data.describe()

	WE	AE	Signal	Lab1	Lab2	\
count	6021.000000	6021.000000	6021.000000	6021.000000	6021.000000	
mean	229.111036	208.110467	21.000569	25.722637	7.072620	
std	12.722230	6.938218	8.591080	55.107618	29.587763	
min	0.564000	38.872000	-41.667777	-280.068781	-164.140427	
25%	220.486982	205.346944	15.064583	7.989353	-12.228035	
50%	233.389663	209.662979	23.171917	21.358645	14.460956	
75%	236.966050	211.015514	26.593889	38.074513	26.521991	
max	255.366500	227.651463	93.160938	1461.434904	293.708249	
	Lab3	Lab4	Temp	RH	Ref	
count	6021.000000	6021.000000	6021.000000	6021.000000	6021.000000	
mean	-28.377323	-697.352025	18.257658	67.307475	24.310285	
std	31.418703	111.688152	7.601533	18.453782	18.579753	
min	-235.559257	-1393.920200	1.037101	11.428168	1.050620	
25%	-49.497866	-769.552964	12.409531	53.636250	11.589200	
50%	-19.292374	-722.425997	16.868500	72.245143	19.618169	

```

75%      -7.293931   -655.081265    22.844245    82.935250    31.467247
max      226.584223   -165.049809    44.748056    93.012486    154.545243

```

```

[39]: NO2_data['Month']=NO2_data.index.month
NO2_data['Day_of_week']=NO2_data.index.dayofweek
NO2_data['Day']=NO2_data.index.day
NO2_data['Hour']=NO2_data.index.hour
NO2_data['Ref_CO']=CO_data['Ref'].to_list()
NO2_data['Ref_SO2']=SO2_data['Ref'].to_list()
NO2_data['Ref_O3']=O3_data['Ref'].to_list()
NO2_Data=NO2_data[['Signal','Ref_CO','Ref_SO2','Ref_O3','Lab1','Temp','RH','Ref','Month','Day','Hour']]
NO2_Data=NO2_Data[(NO2_Data[NO2_Data.columns] >= 0).all(axis=1)]
#CO_data=CO_data.resample('D').mean()
NO2_Data=NO2_Data.dropna()
NO2_Data.shape

```

[39]: (4691, 12)

8.1 Model 1: Linear Regression (LR)

```

[40]: from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error as mae
import sklearn.metrics as sm
import matplotlib.pyplot as plt
X=NO2_Data[['Signal','Ref_CO','Ref_SO2','Ref_O3','Temp','RH','Month','Day_of_week','Day','Hour']]
y=NO2_Data['Ref']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
len(X_test)

```

[40]: 939

```

[41]: lr = LinearRegression()
model = lr.fit(X_train, y_train)
pred = model.predict(X_test)
lab1=NO2_Data['Lab1'].to_list()[len(y_train):]
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index =index)
Y_test
Pred=pd.Series(pred,index =index)
Lab1=pd.Series(lab1,index =index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)

```

```

Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_lr_N02=sMAPE_lr
RMSE_lr_N02=round(RMSE_lr/np.mean(np.array(y_test)),2)
Pearson_lr_N02=Pearson_lr
sMAPE_lab_N02=sMAPE_lab
RMSE_lab_N02=round(RMSE_lab/np.mean(np.array(lab1)),2)
Pearson_lab_N02=Pearson_lab
R2_lr_N02=round(sm.r2_score(y_test, pred), 2)
R2_lab_N02=round(sm.r2_score(y_test, lab1), 2)
RMSE_Lr_N02=RMSE_lr
RMSE_Lab_N02=RMSE_lab

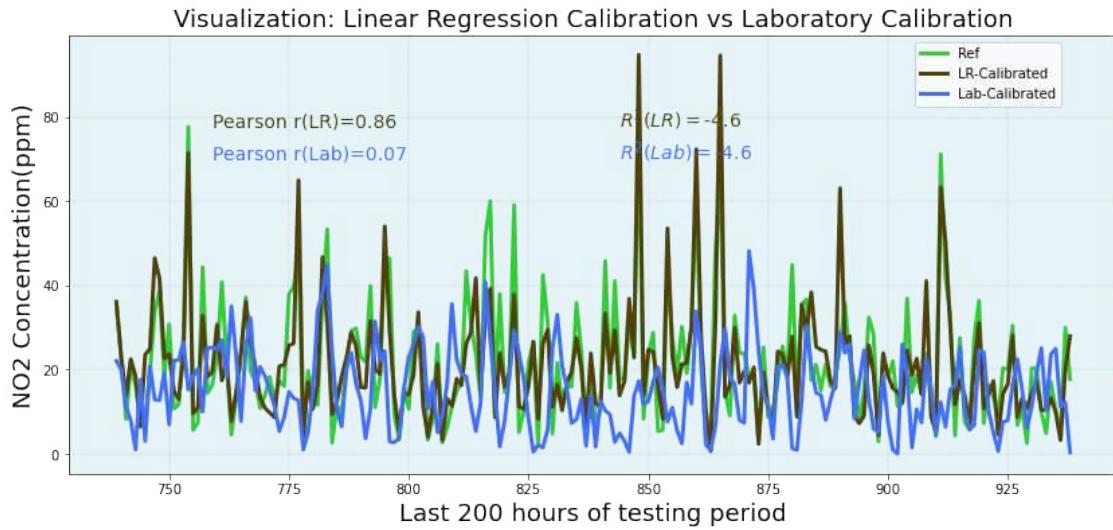
A=len(y_test)-200
B=A+120
D=max(y_test[A:])-0.1*max(y_test[A:])
C=max(y_test[A:])-0*max(y_test[A:])

```

```

[42]: fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.plot(index[A:],y_test[A:], color='limegreen',linewidth=3)
plt.plot(index[A:],pred[A:], color='#513e00',linewidth=3)
plt.plot(index[A:],lab1[A:], color='#426eff',linewidth=3)
plt.legend(['Ref', 'LR-Calibrated', 'Lab-Calibrated'], loc = 2, bbox_to_anchor=[0.8,1])
plt.ylabel('N02 Concentration(ppm)',fontsize=18)
plt.text(B-15, C,r'$R^2$(LR)=$'+str(R2_lab_N02) , fontsize = 14,color="#513e00")
plt.text(B-15, D,r'$R^2$(Lab)=$'+str(R2_lab_N02) , fontsize = 14,color="#426eff")
plt.text(B-100, C, 'Pearson r(LR)=' +str(Pearson_lr), fontsize = 14,color="#513e00")
plt.text(B-100, D, 'Pearson r(Lab)=' +str(Pearson_lab), fontsize = 14,color="#426eff")
plt.xlabel('Last 200 hours of testing period',fontsize=18)
plt.title('Visualization: Linear Regression Calibration vs Laboratory Calibration',fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



```
[43]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred), 2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_test, pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))

MBE_LR_NO2=MBE(pred,y_test)/np.std(y_test)
CRMSE_LR_NO2=CRMSE(y_test,pred)/np.std(y_test)
MBE_LAB_NO2=MBE(lab1,y_test)/(2.6*np.std(y_test))
CRMSE_LAB_NO2=CRMSE(y_test,lab1)/(2.6*np.std(y_test))
pred_lr=pred
```

Regressor model performance:
 Mean absolute error(MAE) = 6.52
 Mean squared error(MSE) = 89.55
 Median absolute error = 4.52
 Explain variance score = 0.73
 R2 score = 0.73

8.2 Model 2: Support Vector Regression (SVR)

```
[44]: from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
regressor = SVR(kernel = 'poly')
regressor.fit(X_train, y_train)
pred = regressor.predict(X_test)
for i in range(len(Pred)):
    if pred[i]<0:
        pred[i]=np.mean(np.array(pred))

[45]: lab1=N02_Data['Lab1'].to_list()[len(y_train):]
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index =index)
Y_test
Pred=pd.Series(pred,index =index)
Lab1=pd.Series(lab1,index =index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_svr_N02=sMAPE_lr
RMSE_svr_N02=round(RMSE_lr/np.mean(np.array(y_test)),2)
Pearson_svr_N02=Pearson_lr
R2_svr_N02=round(sm.r2_score(y_test, pred), 2)
RMSE_Svr_N02=RMSE_lr

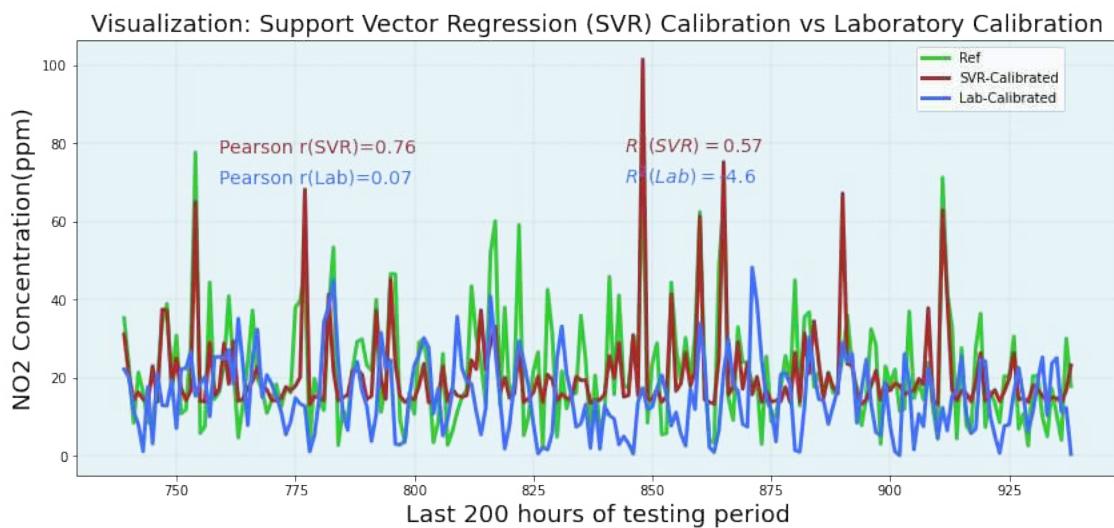
A=len(y_test)-200
D=max(y_test[A:])-0.1*max(y_test[A:])
C=max(y_test[A:])-0*max(y_test[A:])
B=A+120

[46]: fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.plot(index[A:],y_test[A:], color='limegreen',linewidth=3)
plt.plot(index[A:],pred[A:], color='brown',linewidth=3)
plt.plot(index[A:],lab1[A:], color="#426eff",linewidth=3)
plt.legend(['Ref', 'SVR-Calibrated', 'Lab-Calibrated'], loc = 2, bbox_to_anchor=(0.8,1))
plt.ylabel('N02 Concentration(ppm)',fontsize=18)
```

```

plt.text(B-15, C, r'$R^2(SVR)=' + str(R2_svr_N02), fontsize = 14, u
    ↪color='brown')
plt.text(B-15, D, r'$R^2(Lab)=' + str(R2_lab_N02), fontsize = 14, u
    ↪color='#426eff')
plt.text(B-100, C, 'Pearson r(SVR)=' + str(Pearson_lr), fontsize = 14, u
    ↪color='brown')
plt.text(B-100, D, 'Pearson r(Lab)=' + str(Pearson_lab), fontsize = 14, u
    ↪color='#426eff')
plt.xlabel('Last 200 hours of testing period',fontsize=18)
plt.title('Visualization: Support Vector Regression (SVR) Calibration vs Laboratory Calibration',fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



```

[47]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred), u
    ↪2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred), u
    ↪2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred), u
    ↪2))
print("Explained variance score =", round(sm.explained_variance_score(y_test, u
    ↪pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))

MBE_SVR_N02=MBE(pred,y_test)/np.std(y_test)
CRMSE_SVR_N02=CRMSE(y_test,pred)/np.std(y_test)
pred_svr=pred

```

```

Regressor model performance:
Mean absolute error(MAE) = 8.39
Mean squared error(MSE) = 141.95
Median absolute error = 6.39
Explain variance score = 0.57
R2 score = 0.57

```

8.3 Model 3: Random Forest

```
[48]: from sklearn.ensemble import RandomForestRegressor

# create regressor object
regressor = RandomForestRegressor(n_estimators = 500, random_state = 0)

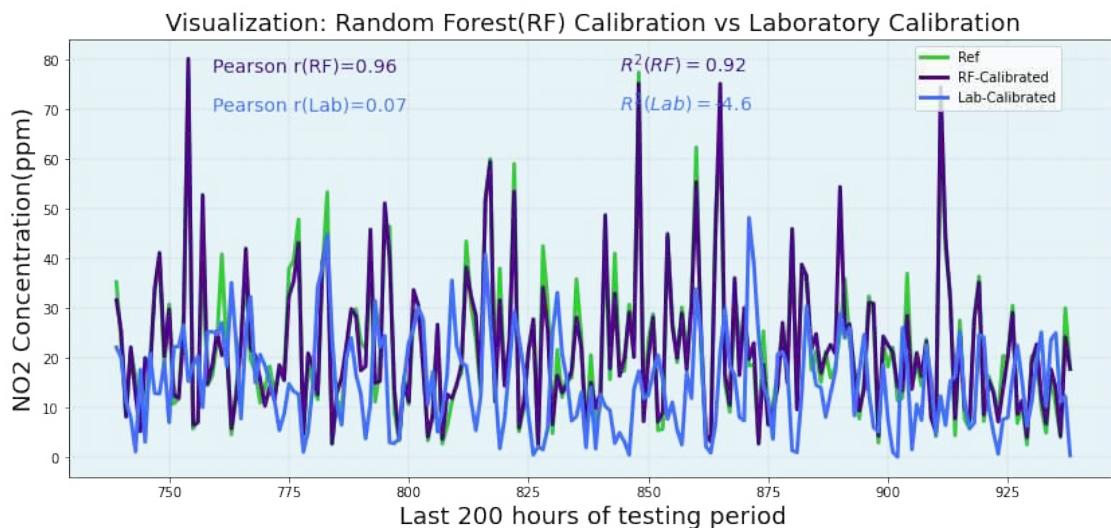
# fit the regressor with x and y data
regressor.fit(X_train, y_train)
```

```
[48]: RandomForestRegressor(n_estimators=500, random_state=0)
```

```
[49]: features_N02=regressor.feature_importances_
pred = regressor.predict(X_test)
lab1=N02_Data['Lab1'].to_list()[:len(y_train):]
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index =index)
Y_test
Pred=pd.Series(pred,index =index)
Lab1=pd.Series(lab1,index =index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_rf_N02=sMAPE_lr
RMSE_rf_N02=round(RMSE_lr/np.mean(np.array(y_test)),2)
Pearson_rf_N02=Pearson_lr
R2_rf_N02=round(sm.r2_score(y_test, pred), 2)
RMSE_Rf_N02=RMSE_lr

A=len(y_test)-200
D=max(y_test[A:])-0.1*max(y_test[A:])
C=max(y_test[A:])-0*max(y_test[A:])
B=A+120
```

```
[50]: fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.plot(index[A:],y_test[A:], color='limegreen',linewidth=3)
plt.plot(index[A:],pred[A:], color='indigo',linewidth=3)
plt.plot(index[A:],lab1[A:], color="#426eff",linewidth=3)
plt.legend(['Ref', 'RF-Calibrated', 'Lab-Calibrated'], loc = 2, bbox_to_anchor=(0.8,1))
plt.ylabel('NO2 Concentration(ppm)',fontsize=18)
plt.text(B-15, C,r'$R^2(RF)=$'+str(R2_rf_NO2) , fontsize = 14, color='indigo')
plt.text(B-15, D,r'$R^2(Lab)=$'+str(R2_lab_NO2) , fontsize = 14,color="#426eff")
plt.text(B-100, C, 'Pearson r(RF)='+str(Pearson_lr), fontsize = 14,color='indigo')
plt.text(B-100, D, 'Pearson r(Lab)='+str(Pearson_lab), fontsize = 14,color="#426eff")
plt.xlabel('Last 200 hours of testing period',fontsize=18)
plt.title('Visualization: Random Forest(RF) Calibration vs Laboratory Calibration',fontsize=18)
plt.grid(linestyle='-.',linewidth=0.3)
plt.show()
```



```
[51]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred),2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred),2))
```

```

print("Median absolute error =", round(sm.median_absolute_error(y_test, pred), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_test, pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))

MBE_RF_N02=MBE(pred,y_test)/np.std(y_test)
CRMSE_RF_N02=CRMSE(y_test,pred)/np.std(y_test)
pred_rf=pred

```

Regressor model performance:

Mean absolute error(MAE) = 3.09
 Mean squared error(MSE) = 26.56
 Median absolute error = 1.93
 Explain variance score = 0.92
 R2 score = 0.92

8.4 Model 4 : ANN

```
[52]: from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
from sklearn.preprocessing import StandardScaler
model = Sequential()
model.add(Dense(6, input_shape = (10,),kernel_initializer='normal', activation='linear'))
model.add(Dense(128,kernel_initializer='normal', activation= 'relu'))
model.add(Dense(128, kernel_initializer='normal',activation= 'relu'))
model.add(Dense(100, kernel_initializer='normal',activation= 'relu'))
model.add(Dense(1,kernel_initializer='normal',activation='linear',))
sgd = optimizers.Adam(learning_rate=0.01)

model.compile(optimizer = sgd, loss = 'mean_squared_error', metrics= ['mse','mae'])
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_5 (Dense)	(None, 6)	66
dense_6 (Dense)	(None, 128)	896
dense_7 (Dense)	(None, 128)	16512

```

dense_8 (Dense)           (None, 100)      12900
-----
dense_9 (Dense)           (None, 1)        101
=====
Total params: 30,475
Trainable params: 30,475
Non-trainable params: 0
-----
```

```
[53]: scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled=scaler.transform(X_train)
X_test_scaled=scaler.transform(X_test)
model.fit(X_train_scaled, y_train, batch_size= 100, epochs=200, verbose= 0)
```

```
[53]: <tensorflow.python.keras.callbacks.History at 0x179644850>
```

```
[54]: train_pred = model.predict(X_train_scaled)
test_pred = model.predict(X_test_scaled)
pred=[]
for i in range(len(test_pred)):
    pred.append(sum(list(test_pred[i])))
len(y_test)
```

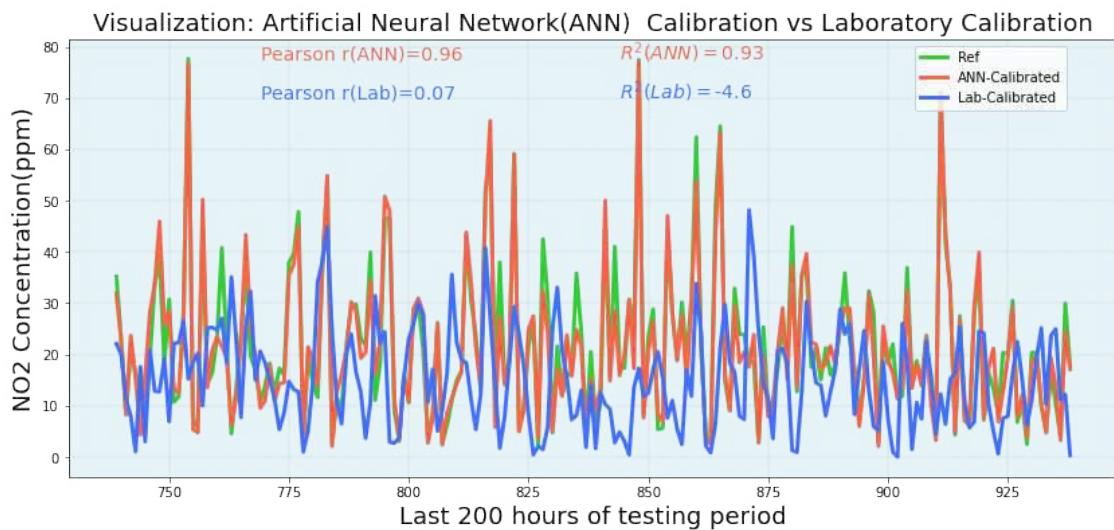
```
[54]: 939
```

```
[55]: lab1=N02_Data['Lab1'].to_list()[len(y_train):]
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index =index)
Y_test
Pred=pd.Series(pred,index =index)
Lab1=pd.Series(lab1,index =index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_ann_N02=sMAPE_lr
RMSE_ann_N02=round(RMSE_lr/np.mean(np.array(y_test)),2)
Pearson_ann_N02=Pearson_lr
R2_ann_N02=round(sm.r2_score(y_test, pred), 2)
RMSE_Ann_N02=RMSE_lr

A=len(y_test)-200
D=max(y_test[A:])-0.1*max(y_test[A:])
```

```
C=max(y_test[A:])-0*max(y_test[A:])
B=A+120
```

```
[56]: fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.plot(index[A:],y_test[A:], color='limegreen',linewidth=3)
plt.plot(index[A:],pred[A:], color='tomato',linewidth=3)
plt.plot(index[A:],lab1[A:], color="#426eff",linewidth=3)
plt.legend(['Ref', 'ANN-Calibrated', 'Lab-Calibrated'], loc = 2, bbox_to_anchor=(0.8,1))
plt.ylabel('NO2 Concentration(ppm)',fontsize=18)
plt.text(B-15, C, r'$R^2(ANN)=$'+str(R2_ann_N02), fontsize = 14,color='tomato')
plt.text(B-15, D, r'$R^2(Lab)=$'+str(R2_lab_N02), fontsize = 14,color="#426eff")
plt.text(B-90, C, 'Pearson r(ANN)='+str(Pearson_lr), fontsize = 14,color='tomato')
plt.text(B-90, D, 'Pearson r(Lab)='+str(Pearson_lab), fontsize = 14,color="#426eff")
plt.xlabel('Last 200 hours of testing period',fontsize=18)
plt.title('Visualization: Artificial Neural Network(ANN) Calibration vs Laboratory Calibration',fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()
```



```
[57]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred), 2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_test, pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))

MBE_ANN_N02=MBE(pred,y_test)/np.std(y_test)
CRMSE_ANN_N02=CRMSE(y_test,pred)/(np.std(y_test))
pred_ann=pred
```

Regressor model performance:
 Mean absolute error(MAE) = 2.93
 Mean squared error(MSE) = 22.99
 Median absolute error = 2.01
 Explain variance score = 0.93
 R2 score = 0.93

9 Model 5: XGBoost

```
[58]: from xgboost import XGBRegressor
from numpy import absolute
from pandas import read_csv
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
# create an xgboost regression model
#n_estimators=10000, max_depth=5, eta=0.01, subsample=0.9, colsample_bytree=0.4, alpha=10
model = XGBRegressor(n_estimators=10000, max_depth=5, eta=0.01, subsample=0.9,
                      colsample_bytree=0.4, alpha=10)

model.fit(X_train,y_train)
```

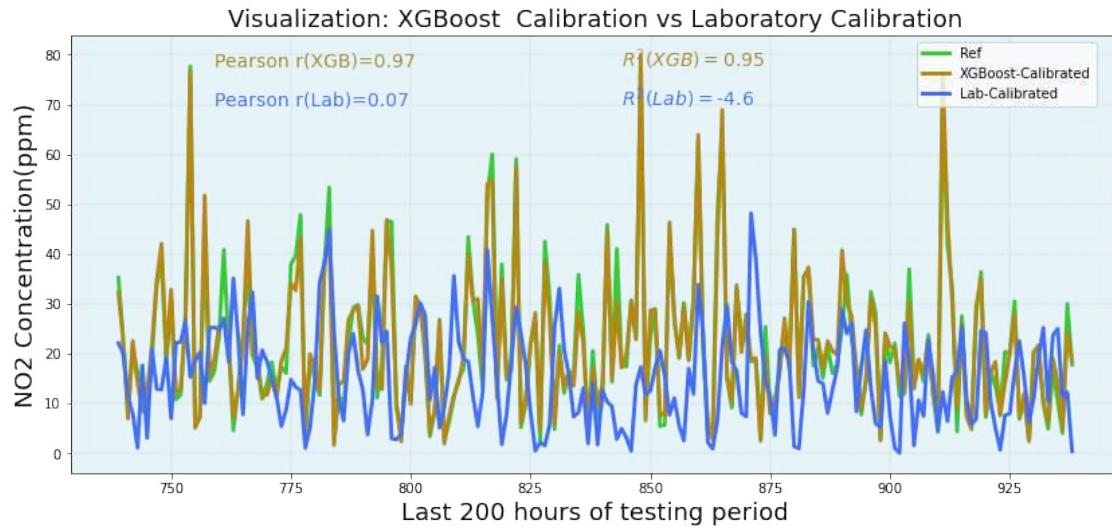
```
[58]: XGBRegressor(alpha=10, base_score=0.5, booster='gbtree', colsample_bylevel=1,
                  colsample_bynode=1, colsample_bytree=0.4, eta=0.01, gamma=0,
                  gpu_id=-1, importance_type='gain', interaction_constraints='',
                  learning_rate=0.00999999978, max_delta_step=0, max_depth=5,
                  min_child_weight=1, missing=nan, monotone_constraints='()',
                  n_estimators=10000, n_jobs=4, num_parallel_tree=1, random_state=0,
                  reg_alpha=10, reg_lambda=1, scale_pos_weight=1, subsample=0.9,
                  tree_method='exact', validate_parameters=1, verbosity=None)
```

```
[59]: pred = model.predict(X_test)
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index =index)
Y_test
Pred=pd.Series(pred,index =index)
Lab1=pd.Series(lab1,index =index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_xgb_N02=sMAPE_lr
RMSE_xgb_N02=RMSE_lr/np.mean(np.array(y_test))
Pearson_xgb_N02=Pearson_lr
R2_xgb_N02=round(sm.r2_score(y_test, pred), 2)
RMSE_Xgb_N02=RMSE_lr

A=len(y_test)-200
D=max(y_test[A:])-0.1*max(y_test[A:])
C=max(y_test[A:])-0*max(y_test[A:])
B=A+120
```

```
[60]: fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.plot(index[A:],y_test[A:], color='limegreen',linewidth=3)
plt.plot(index[A:],pred[A:], color='darkgoldenrod',linewidth=3)
plt.plot(index[A:],lab1[A:], color="#426eff",linewidth=3)
plt.legend(['Ref', 'XGBoost-Calibrated', 'Lab-Calibrated'], loc = 2,
bbox_to_anchor = (0.8,1))
plt.ylabel('N02 Concentration(ppm)',fontsize=18)
plt.text(B-15, C, r'$R^2$(XGB)='+str(R2_xgb_N02), fontsize = 14,
color='darkgoldenrod')
plt.text(B-15, D, r'$R^2$(Lab)='+str(R2_lab_N02), fontsize = 14,
color="#426eff")
plt.text(B-100, C, 'Pearson r(XGB)='+str(Pearson_lr), fontsize = 14,
color='darkgoldenrod')
plt.text(B-100, D, 'Pearson r(Lab)='+str(Pearson_lab), fontsize = 14,
color="#426eff")
plt.xlabel('Last 200 hours of testing period',fontsize=18)
plt.title('Visualization: XGBoost Calibration vs Laboratory',
Calibration',fontsize=18)
```

```
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()
```



```
[61]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred), 2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_test, pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))

MBE_XGB_NO2=MBE(pred,y_test)/np.std(y_test)
CRMSE_XGB_NO2=CRMSE(y_test,pred)/np.std(y_test)
pred_xgb=pred
```

Regressor model performance:
 Mean absolute error(MAE) = 2.58
 Mean squared error(MSE) = 17.93
 Median absolute error = 1.73
 Explain variance score = 0.95
 R2 score = 0.95

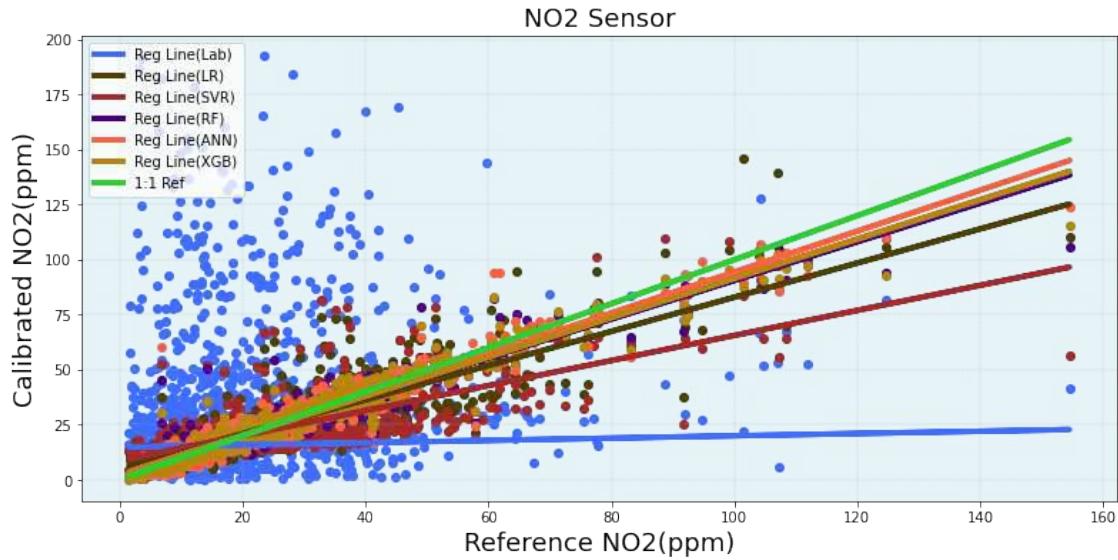
```
[62]: fig= plt.figure(figsize=(13,6))
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
```

```

m0, b0 = np.polyfit(np.array(y_test), 0.4*np.array(lab1), 1)
plt.plot(np.array(y_test), m0*np.array(y_test) + b0,color='#426eff',linewidth=4)
m1, b1 = np.polyfit(np.array(y_test), np.array(pred_lr), 1)
plt.plot(np.array(y_test), m1*np.array(y_test) + b1,color='#513e00',linewidth=4)
m2, b2 = np.polyfit(np.array(y_test), np.array(pred_svr), 1)
plt.plot(np.array(y_test), m2*np.array(y_test) + b2,color='brown',linewidth=4)
m3, b3 = np.polyfit(np.array(y_test), np.array(pred_rf), 1)
plt.plot(np.array(y_test), m3*np.array(y_test) + b3,color='indigo',linewidth=4)
m4, b4 = np.polyfit(np.array(y_test), np.array(pred_ann), 1)
plt.plot(np.array(y_test), m4*np.array(y_test) + b4,color='tomato',linewidth=4)
m5, b5 = np.polyfit(np.array(y_test), np.array(pred_xgb), 1)
plt.plot(np.array(y_test), m5*np.array(y_test) + b5,color='darkgoldenrod',linewidth=4)

plt.scatter(np.array(y_test),np.array(lab1),color='#426eff')
plt.scatter(np.array(y_test),np.array(pred_lr),color='#513e00')
plt.scatter(np.array(y_test),np.array(pred_svr),color='brown')
plt.scatter(np.array(y_test),np.array(pred_rf),color='indigo')
plt.scatter(np.array(y_test),np.array(pred_ann),color='tomato')
plt.scatter(np.array(y_test),np.array(pred_xgb),color='darkgoldenrod')
ax.plot(y_test,y_test, c ="limegreen",linewidth=4)
plt.xlabel('Reference NO2(ppm)',fontsize=18)
plt.ylabel('Calibrated NO2(ppm)',fontsize=18)
plt.legend(['Reg Line(Lab)', 'Reg Line(LR)', 'Reg Line(SVR)', 'Reg Line(RF)', 'Reg Line(ANN)', 'Reg Line(XGB)', '1:1 Ref'], loc = 2, bbox_to_anchor = (0,1))
plt.title('NO2 Sensor',fontsize=18 )
plt.grid(linestyle='-.', linewidth=0.3)

```



```
[63]: mean_test=np.array(np.mean(np.array(y_test)))
mean_pred=np.array(np.mean(np.array(pred)))
mean_lab=np.array(np.mean(np.array(lab1)))
mean_test,mean_pred,mean_lab

[63]: (array(22.93707451), array(22.844624, dtype=float32), array(39.95566787))

[64]: y_test[0],pred[0],lab1[0]

[64]: (7.727682433333333, 9.642745, 112.51357800527329)
```

SO2 Calibration

```
[65]: import pandas as pd
import scipy.io
import numpy as np
mat = scipy.io.loadmat('So2final.mat')

L_S02final=[]
for i in range(len(mat['S02final'])):
    L_S02final.append(list(mat['S02final'][i]))

S02_data=pd.DataFrame(L_S02final,columns=['Lab1',  
                                         'Lab2','Lab3','Lab4','Temp','RH','Ref','Time'])
Time=S02_data['Time'].to_list()
Time=np.array(Time)
Date=pd.to_datetime(Time-719529,unit='d').round('h')
S02_data['Date'] = Date.tolist()
S02_data=S02_data.set_index('Date')
S02_data.drop('Time',axis = 1, inplace = True)

mat = scipy.io.loadmat('S02_raw.mat')

L_S02_raw=[]
for i in range(len(mat['S02_raw'])):
    L_S02_raw.append(list(mat['S02_raw'][i]))

S02_raw=pd.DataFrame(L_S02_raw,columns=['WE', 'AE','Temp','RH','Time'])
S02_raw.head()
S02_data.insert(loc = 0,
                column = 'WE',
                value = S02_raw['WE'].to_list())

S02_data.insert(loc = 1,
                column = 'AE',
                value = S02_raw['AE'].to_list())
```

```

S02_data=S02_data.interpolate()
WE=np.array(S02_data['WE'].to_list())
AE=np.array(S02_data['AE'].to_list())
Signal=list(WE-AE)
S02_data.insert(loc = 2,
                 column = 'Signal',
                 value = Signal)

S02_data.head()

```

[65]:

	WE	AE	Signal	Lab1	\
Date					
2019-10-02 12:00:00	511.980000	375.956250	136.023750	336.491001	
2019-10-02 13:00:00	511.980000	508.330476	3.649524	-339.232168	
2019-10-02 16:00:00	511.980000	442.351642	69.628358	-77.358106	
2019-10-02 17:00:00	435.447463	352.494627	82.952836	221.907307	
2019-10-03 16:00:00	358.812857	336.112857	22.700000	84.566512	

	Lab2	Lab3	Lab4	Temp	\
Date					
2019-10-02 12:00:00	395.131511	258.535708	-624.848616	26.378438	
2019-10-02 13:00:00	-68.687897	-313.588071	-619.380130	25.502791	
2019-10-02 16:00:00	108.340330	-219.979244	-680.245645	30.827910	
2019-10-02 17:00:00	244.067240	121.645425	-897.011152	30.047164	
2019-10-03 16:00:00	77.784764	-11.080530	-1118.485141	29.441429	

	RH	Ref
Date		
2019-10-02 12:00:00	58.063437	1.701245
2019-10-02 13:00:00	59.868837	1.495635
2019-10-02 16:00:00	49.008060	1.102313
2019-10-02 17:00:00	51.259851	1.383004
2019-10-03 16:00:00	52.018571	1.402984

[66]: S02_data.shape

[66]: (6021, 10)

[67]: S02_data.describe()

[67]:

	WE	AE	Signal	Lab1	Lab2	\
count	6021.000000	6021.000000	6021.000000	6021.000000	6021.000000	
mean	346.317991	343.672928	2.645063	4.426280	10.876447	
std	11.844320	5.652805	9.278214	31.465055	27.662360	
min	327.665107	336.112857	-19.034635	-961.004985	-505.040176	
25%	339.505000	341.720921	-3.212714	-10.826914	-5.415598	

```

50%      343.395667   342.801650    0.383073    0.567498    5.306895
75%      349.409378   344.366921    5.712667   15.114913   20.328606
max      511.984380   511.984380  137.776860  413.434407  419.568327

```

	Lab3	Lab4	Temp	RH	Ref
count	6021.000000	6021.000000	6021.000000	6021.000000	6021.000000
mean	-89.010607	-1120.030522	18.258139	67.305016	1.715263
std	34.530699	64.509983	7.601563	18.453999	0.995402
min	-1302.528007	-1777.717366	1.035000	11.428168	-0.742694
25%	-103.105554	-1127.334289	12.409531	53.576341	1.092307
50%	-92.472035	-1115.412344	16.872568	72.220802	1.523944
75%	-78.691378	-1099.959520	22.849585	82.938548	2.061193
max	322.268946	-599.346047	44.748056	93.012486	18.687856

```

[68]: S02_data['Month']=S02_data.index.month
S02_data['Day_of_week']=S02_data.index.dayofweek
S02_data['Day']=S02_data.index.day
S02_data['Hour']=S02_data.index.hour

S02_data['Ref_CO']=CO_data['Ref'].to_list()
S02_data['Ref_NO2']=NO2_data['Ref'].to_list()
S02_data['Ref_O3']=O3_data['Ref'].to_list()
S02_Data=S02_data[['Signal','Ref_CO','Ref_NO2','Ref_O3','Lab1','Temp','RH','Ref','Month','Day']]
S02_Data=S02_Data[(S02_Data[S02_Data.columns] >= 0).all(axis=1)]
#CO_data=CO_data.resample('D').mean()
S02_Data=S02_Data.dropna()
S02_Data.shape

```

[68]: (2794, 12)

10 Model 1: Linear Regression (LR)

```

[69]: from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error as mae
import sklearn.metrics as sm
import matplotlib.pyplot as plt
X=S02_Data[['Signal','Ref_CO','Ref_NO2','Ref_O3','Temp','RH','Month','Day_of_week','Day','Hour']]
y=S02_Data['Ref']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
len(X_test)

```

[69]: 559

```
[70]: lr = LinearRegression()
model = lr.fit(X_train, y_train)
pred = model.predict(X_test)
lab1=S02_Data['Lab1'].to_list()[:len(y_train):]
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index=index)
Y_test
Pred=pd.Series(pred,index=index)
Lab1=pd.Series(lab1,index=index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round(smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_lr_S02=sMAPE_lr
RMSE_lr_S02=RMSE_lr/np.mean(np.array(y_test))
Pearson_lr_S02=Pearson_lr
sMAPE_lab_S02=sMAPE_lab
RMSE_lab_S02=RMSE_lab/np.mean(np.array(lab1))
Pearson_lab_S02=Pearson_lab
R2_lr_S02=round(sm.r2_score(y_test, pred), 2)
R2_lab_S02=round(sm.r2_score(y_test, lab1), 2)
RMSE_Lr_S02=RMSE_lr
RMSE_Lab_S02=RMSE_lab
```

10.1 Scaling Laboratory Calibration

For the purpose of visual comparison with the ref and calibrated measurements, the lab measurement was scaled by a factor of 0.05

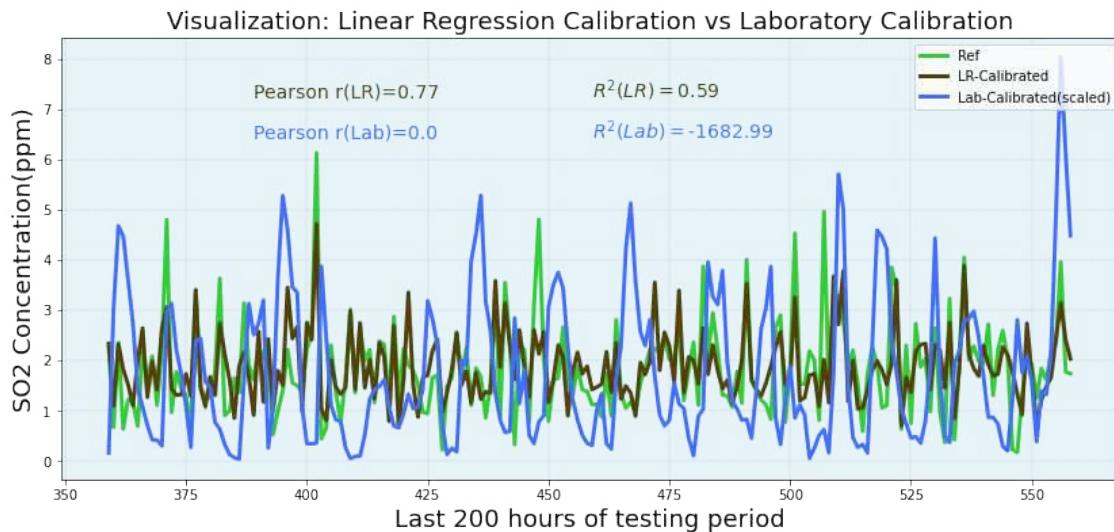
```
[71]: LAB1=0.05*np.array(Lab1)
A=len(y_test)-200
D=max(LAB1[A:])-0.2*max(LAB1[A:])
C=max(LAB1[A:])-0.1*max(LAB1[A:])
B=A+120
```

```
[72]: fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.plot(index[A:],y_test[A:], color='limegreen', linewidth=3)
plt.plot(index[A:],pred[A:], color="#513e00", linewidth=3)
plt.plot(index[A:],LAB1[A:], color="#426eff", linewidth=3)
```

```

plt.legend(['Ref', 'LR-Calibrated', 'Lab-Calibrated(scaled)'], loc = 2,
          bbox_to_anchor = (0.8,1))
plt.ylabel('SO2 Concentration(ppm)', fontsize=18)
plt.text(B-20, C, r'$R^2(LR)=$'+str(R2_lr_SO2), fontsize = 14,
         color='#513e00')
plt.text(B-20, D, r'$R^2(Lab)=$'+str(R2_lab_SO2), fontsize = 14,
         color='#426eff')
plt.text(B-90, C, 'Pearson r(LR)='+str(Pearson_lr), fontsize = 14,
         color='#513e00')
plt.text(B-90, D, 'Pearson r(Lab)='+str(Pearson_lab), fontsize = 14,
         color='#426eff')
plt.xlabel('Last 200 hours of testing period', fontsize=18)
plt.title('Visualization: Linear Regression Calibration vs Laboratory Calibration', fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



```

[73]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred),
                                          2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred),
                                          2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred),
                                         2))
print("Explained variance score =", round(sm.explained_variance_score(y_test,
                                                                      pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))
MBE_LR_SO2=MBE(pred,y_test)/np.std(y_test)

```

```

CRMSE_LR_S02=CRMSE(y_test,pred)/np.std(y_test)
MBE_LAB_S02=MBE(lab1,y_test)/(25*np.std(y_test))
CRMSE_LAB_S02=CRMSE(y_test,lab1)/25*(np.std(y_test))
pred_lr=pred

```

Regressor model performance:
 Mean absolute error(MAE) = 0.46
 Mean squared error(MSE) = 0.37
 Median absolute error = 0.38
 Explain variance score = 0.59
 R2 score = 0.59

11 Model 2: SVR

```
[74]: from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
regressor = SVR(kernel = 'poly',degree=3)
regressor.fit(X_train, y_train)
pred = regressor.predict(X_test)
```

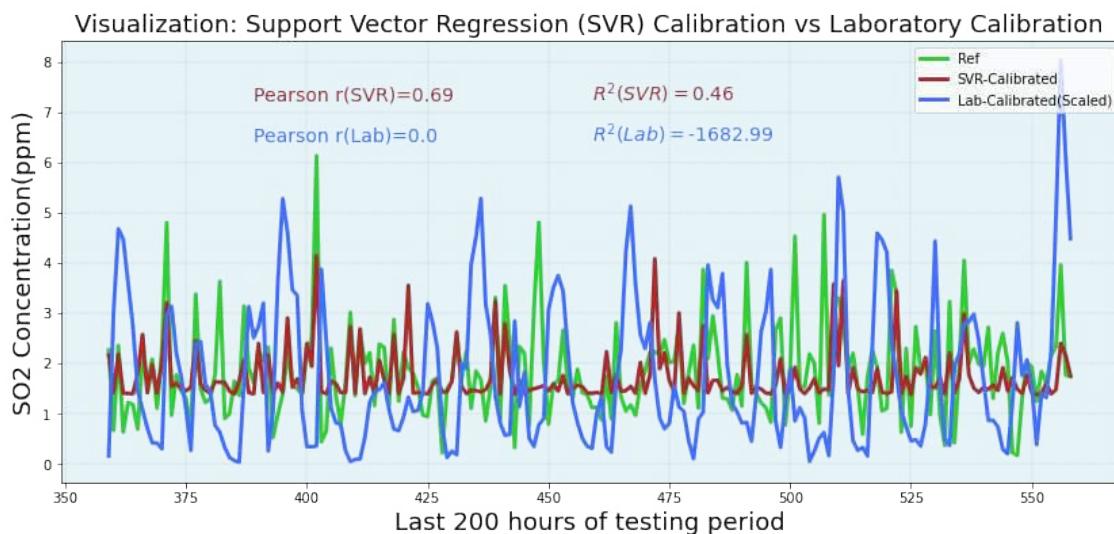
```
[75]: lab1=S02_Data['Lab1'].to_list()[len(y_train):]
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index =index)
Y_test
Pred=pd.Series(pred,index =index)
Lab1=pd.Series(lab1,index =index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_svr_S02=sMAPE_lr
RMSE_svr_S02=RMSE_lr/np.mean(np.array(y_test))
Pearson_svr_S02=Pearson_lr
R2_svr_S02=round(sm.r2_score(y_test, pred), 2)
RMSE_Svr_S02=RMSE_lr
```

```
[76]: fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.plot(index[A:],y_test[A:], color='limegreen',linewidth=3)
```

```

plt.plot(index[A:],pred[A:], color='brown',linewidth=3)
plt.plot(index[A:],LAB1[A:], color="#426eff",linewidth=3)
plt.legend(['Ref', 'SVR-Calibrated', 'Lab-Calibrated(Scaled)'], loc = 2,
           bbox_to_anchor = (0.8,1))
plt.ylabel('SO2 Concentration(ppm)',fontsize=18)
plt.text(B-20, C,r'$R^2$(SVR)=$'+str(R2_svr_SO2) , fontsize = 14,
         color='brown')
plt.text(B-20, D, r'$R^2$(Lab)=$'+str(R2_lab_SO2), fontsize = 14,
         color="#426eff")
plt.text(B-90, C, 'Pearson r(SVR)='+str(Pearson_lr), fontsize = 14,
         color='brown')
plt.text(B-90, D, 'Pearson r(Lab)='+str(Pearson_lab), fontsize = 14,
         color="#426eff")
plt.xlabel('Last 200 hours of testing period',fontsize=18)
plt.title('Visualization: Support Vector Regression (SVR) Calibration vs Laboratory Calibration',fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



```

[77]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred),
                                          2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred),
                                          2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred),
                                         2))
print("Explained variance score =", round(sm.explained_variance_score(y_test,
                                                                     pred), 2))

```

```

print("R2 score =", round(sm.r2_score(y_test, pred), 2))

MBE_SVR_S02=MBE(pred,y_test)/np.std(y_test)
CRMSE_SVR_S02=CRMSE(y_test,pred)/np.std(y_test)
pred_svr=pred

```

Regressor model performance:
 Mean absolute error(MAE) = 0.51
 Mean squared error(MSE) = 0.48
 Median absolute error = 0.4
 Explain variance score = 0.47
 R2 score = 0.46

Model 3: Random Forest

```
[78]: from sklearn.ensemble import RandomForestRegressor

# create regressor object
regressor = RandomForestRegressor(n_estimators = 500, random_state = 0)

# fit the regressor with x and y data
regressor.fit(X_train, y_train)
```

```
[78]: RandomForestRegressor(n_estimators=500, random_state=0)
```

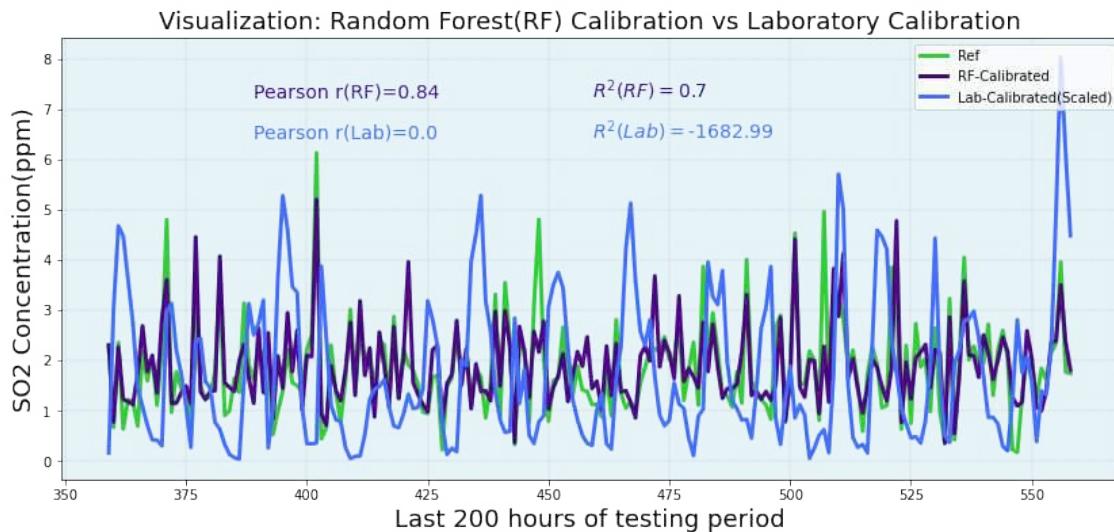
```
[79]: features_S02=regressor.feature_importances_
pred = regressor.predict(X_test)
lab1=S02_Data['Lab1'].to_list()[len(y_train):]
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index =index)
Y_test
Pred=pd.Series(pred,index =index)
Lab1=pd.Series(lab1,index =index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_rf_S02=sMAPE_lr
RMSE_rf_S02=RMSE_lr/np.mean(np.array(y_test))
Pearson_rf_S02=Pearson_lr
R2_rf_S02=round(sm.r2_score(y_test, pred), 2)
RMSE_Rf_S02=RMSE_lr
```

```
[80]: fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
```

```

ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.plot(index[A:],y_test[A:], color='limegreen', linewidth=3)
plt.plot(index[A:],pred[A:], color='indigo', linewidth=3)
plt.plot(index[A:],LAB1[A:], color='#426eff', linewidth=3)
plt.legend(['Ref', 'RF-Calibrated', 'Lab-Calibrated(Scaled)'], loc = 2,
           bbox_to_anchor = (0.8,1))
plt.ylabel('SO2 Concentration(ppm)', fontsize=18)
plt.text(B-20, C,r'$R^2(RF)=$'+str(R2_rf_SO2) , fontsize = 14, color='indigo')
plt.text(B-20, D,r'$R^2(Lab)=$'+str(R2_lab_SO2) , fontsize = 14,
         color="#426eff")
plt.text(B-90, C, 'Pearson r(RF)='+str(Pearson_lr), fontsize = 14,
         color='indigo')
plt.text(B-90, D, 'Pearson r(Lab)='+str(Pearson_lab), fontsize = 14,
         color="#426eff")
plt.xlabel('Last 200 hours of testing period', fontsize=18)
plt.title('Visualization: Random Forest(RF) Calibration vs Laboratory Calibration', fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



```

[81]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred),
                                          2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred),
                                          2))

```

```

print("Median absolute error =", round(sm.median_absolute_error(y_test, pred), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_test, pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))

MBE_RF_S02=MBE(pred,y_test)/np.std(y_test)
CRMSE_RF_S02=CRMSE(y_test,pred)/np.std(y_test)
pred_rf=pred

```

Regressor model performance:

Mean absolute error(MAE) = 0.37
 Mean squared error(MSE) = 0.27
 Median absolute error = 0.29
 Explain variance score = 0.7
 R2 score = 0.7

12 Model 4 : ANN

```
[82]: from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
from sklearn.preprocessing import StandardScaler
model = Sequential()
model.add(Dense(6, input_shape = (10,),kernel_initializer='normal', activation='linear'))
model.add(Dense(128,kernel_initializer='normal', activation= 'relu'))
model.add(Dense(50, kernel_initializer='normal',activation= 'relu'))
#model.add(Dense(100, kernel_initializer='normal',activation= 'relu'))
model.add(Dense(1,kernel_initializer='normal',activation='linear',))
sgd = optimizers.Adam(learning_rate=0.01)

model.compile(optimizer = sgd, loss = 'mean_squared_error', metrics= ['mse','mae'])
model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_10 (Dense)	(None, 6)	66
dense_11 (Dense)	(None, 128)	896
dense_12 (Dense)	(None, 50)	6450

```
dense_13 (Dense)           (None, 1)      51
=====
Total params: 7,463
Trainable params: 7,463
Non-trainable params: 0
```

```
[83]: scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled=scaler.transform(X_train)
X_test_scaled=scaler.transform(X_test)
model.fit(X_train_scaled, y_train, batch_size= 200, epochs=100, verbose= 0)
```

```
[83]: <tensorflow.python.keras.callbacks.History at 0x191376eb0>
```

```
[84]: train_pred = model.predict(X_train_scaled)
test_pred = model.predict(X_test_scaled)
pred=[]
for i in range(len(test_pred)):
    pred.append(sum(list(test_pred[i])))
len(y_test)
```

```
[84]: 559
```

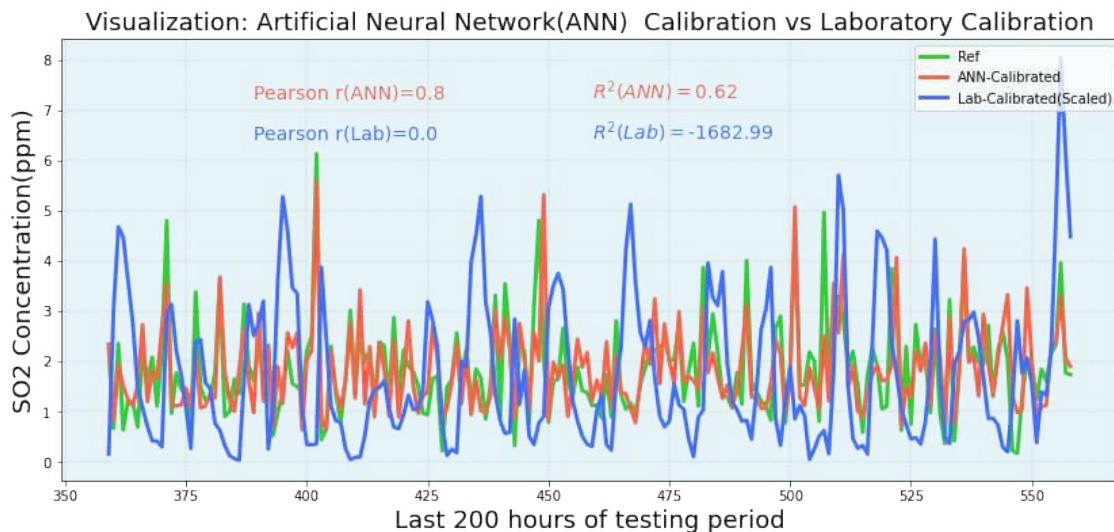
```
[85]: lab1=S02_Data['Lab1'].to_list()[len(y_train):]
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index =index)
Y_test
Pred=pd.Series(pred,index =index)
Lab1=pd.Series(lab1,index =index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_ann_S02=sMAPE_lr
RMSE_ann_S02=RMSE_lr/np.mean(np.array(y_test))
Pearson_ann_S02=Pearson_lr
R2_ann_S02=round(sm.r2_score(y_test, pred), 2)
RMSE_Ann_S02=RMSE_lr
```

```
[86]: fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
```

```

ax.patch.set_alpha(0.3)
plt.plot(index[A:],y_test[A:], color='limegreen', linewidth=3)
plt.plot(index[A:],pred[A:], color='tomato', linewidth=3)
plt.plot(index[A:],LAB1[A:], color='#426eff', linewidth=3)
plt.legend(['Ref', 'ANN-Calibrated', 'Lab-Calibrated(Scaled)'], loc = 2,
           bbox_to_anchor = (0.8,1))
plt.ylabel('SO2 Concentration(ppm)', fontsize=18)
plt.text(B-20, C,r'$R^2(ANN)=$'+str(R2_ann_SO2) , fontsize = 14,
         color='tomato')
plt.text(B-20, D,r'$R^2(Lab)=$'+str(R2_lab_SO2) , fontsize = 14,
         color='#426eff')
plt.text(B-90, C, 'Pearson r(ANN)='+str(Pearson_lr), fontsize = 14,
         color='tomato')
plt.text(B-90, D, 'Pearson r(Lab)='+str(Pearson_lab), fontsize = 14,
         color='#426eff')
plt.xlabel('Last 200 hours of testing period', fontsize=18)
plt.title('Visualization: Artificial Neural Network(ANN) Calibration vs Laboratory Calibration', fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



```
[87]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred),
                                          2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred),
                                          2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred),
                                         2))
```

```

print("Explain variance score =", round(sm.explained_variance_score(y_test, pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))

MBE_ANN_SO2=MBE(pred,y_test)/np.std(y_test)
CRMSE_ANN_SO2=CRMSE(y_test,pred)/np.std(y_test)
pred_ann=pred

```

Regressor model performance:
Mean absolute error(MAE) = 0.43
Mean squared error(MSE) = 0.34
Median absolute error = 0.31
Explain variance score = 0.62
R2 score = 0.62

13 Model 5 : XGBoost

```
[88]: from xgboost import XGBRegressor
from numpy import absolute
from pandas import read_csv
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
# create an xgboost regression model
#n_estimators=10000, max_depth=5, eta=0.01, subsample=0.9, colsample_bytree=0.4, alpha=10
model = XGBRegressor(n_estimators=10000, max_depth=5, eta=0.01, subsample=0.9,
                      colsample_bytree=0.4, alpha=10)

model.fit(X_train,y_train)
```

```
[88]: XGBRegressor(alpha=10, base_score=0.5, booster='gbtree', colsample_bylevel=1,
                   colsample_bynode=1, colsample_bytree=0.4, eta=0.01, gamma=0,
                   gpu_id=-1, importance_type='gain', interaction_constraints='',
                   learning_rate=0.00999999978, max_delta_step=0, max_depth=5,
                   min_child_weight=1, missing=nan, monotone_constraints='()',
                   n_estimators=10000, n_jobs=4, num_parallel_tree=1, random_state=0,
                   reg_alpha=10, reg_lambda=1, scale_pos_weight=1, subsample=0.9,
                   tree_method='exact', validate_parameters=1, verbosity=None)
```

```
[89]: pred = model.predict(X_test)
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index =index)
Y_test
Pred=pd.Series(pred,index =index)
```

```

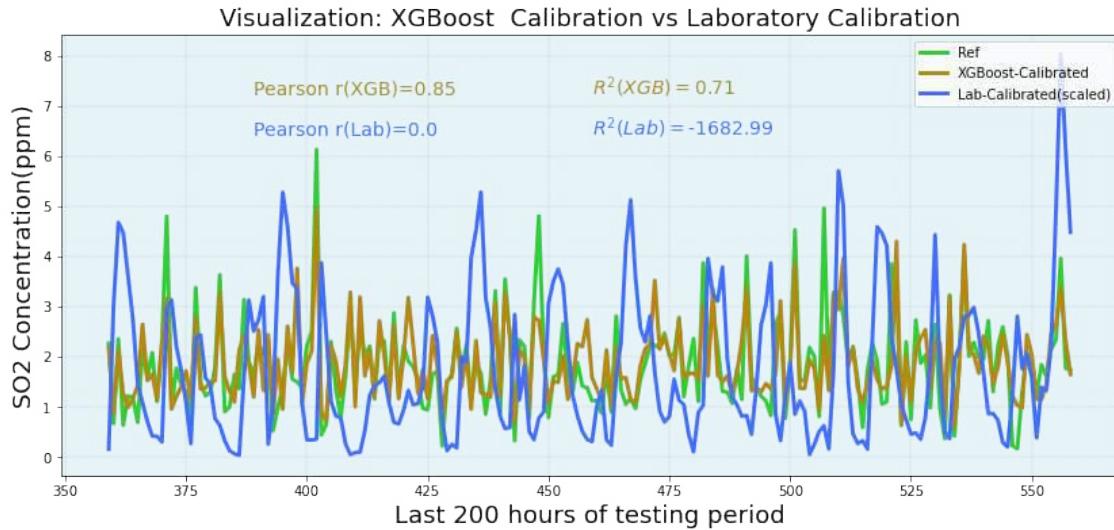
Lab1=pd.Series(lab1,index =index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_xgb_S02=sMAPE_lr
RMSE_xgb_S02=RMSE_lr/np.mean(np.array(y_test))
Pearson_xgb_S02=Pearson_lr
R2_xgb_S02=round(sm.r2_score(y_test, pred), 2)
RMSE_Xgb_S02=RMSE_lr

```

```

[90]: fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.plot(index[A:],y_test[A:], color='limegreen',linewidth=3)
plt.plot(index[A:],pred[A:], color='darkgoldenrod',linewidth=3)
plt.plot(index[A:],LAB1[A:], color="#426eff",linewidth=3)
plt.legend(['Ref', 'XGBoost-Calibrated', 'Lab-Calibrated(scaled)'], loc = 2,
           bbox_to_anchor = (0.8,1))
plt.ylabel('S02 Concentration(ppm)',fontsize=18)
plt.text(B-20, C,r'$R^2(XGB)=$'+str(R2_xgb_S02) , fontsize = 14,
         color='darkgoldenrod')
plt.text(B-20, D,r'$R^2(Lab)=$'+str(R2_lab_S02), fontsize = 14,
         color="#426eff")
plt.text(B-90, C, 'Pearson r(XGB)='+str(Pearson_lr), fontsize = 14,
         color='darkgoldenrod')
plt.text(B-90, D, 'Pearson r(Lab)='+str(Pearson_lab), fontsize = 14,
         color="#426eff")
plt.xlabel('Last 200 hours of testing period',fontsize=18)
plt.title('Visualization: XGBoost Calibration vs Laboratory
           Calibration',fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



```
[91]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred), 2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_test, pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))

MBE_XGB_SO2=MBE(pred,y_test)/np.std(y_test)
CRMSE_XGB_SO2=CRMSE(y_test,pred)/np.std(y_test)
pred_xgb=pred
```

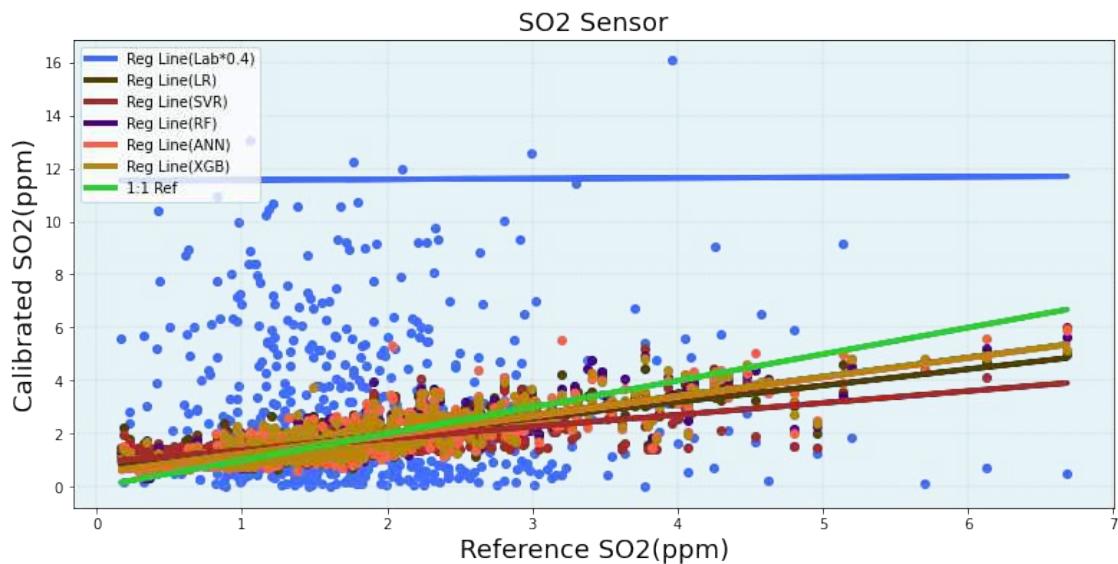
Regressor model performance:
 Mean absolute error(MAE) = 0.37
 Mean squared error(MSE) = 0.25
 Median absolute error = 0.29
 Explain variance score = 0.72
 R2 score = 0.71

```
[92]: import seaborn as sns
fig= plt.figure(figsize=(13,6))
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
m0, b0 = np.polyfit(np.array(y_test), 0.4*np.array(lab1), 1)
plt.plot(np.array(y_test), m0*np.array(y_test) + b0,color="#426eff", linewidth=4)
```

```

m1, b1 = np.polyfit(np.array(y_test), np.array(pred_lr), 1)
plt.plot(np.array(y_test), m1*np.array(y_test) + b1,color="#513e00", linewidth=4)
m2, b2 = np.polyfit(np.array(y_test), np.array(pred_svr), 1)
plt.plot(np.array(y_test), m2*np.array(y_test) + b2,color='brown', linewidth=4)
m3, b3 = np.polyfit(np.array(y_test), np.array(pred_rf), 1)
plt.plot(np.array(y_test), m3*np.array(y_test) + b3,color='indigo', linewidth=4)
m4, b4 = np.polyfit(np.array(y_test), np.array(pred_ann), 1)
plt.plot(np.array(y_test), m4*np.array(y_test) + b4,color='tomato', linewidth=4)
m5, b5 = np.polyfit(np.array(y_test), np.array(pred_xgb), 1)
plt.plot(np.array(y_test), m5*np.array(y_test) + b5,color='darkgoldenrod', linewidth=4)
plt.scatter(np.array(y_test),0.1*np.array(lab1),color="#426eff")
plt.scatter(np.array(y_test),np.array(pred_lr),color="#513e00")
plt.scatter(np.array(y_test),np.array(pred_svr),color='brown')
plt.scatter(np.array(y_test),np.array(pred_rf),color='indigo')
plt.scatter(np.array(y_test),np.array(pred_ann),color='tomato')
plt.scatter(np.array(y_test),np.array(pred_xgb),color='darkgoldenrod')
ax.plot(y_test,y_test, c ="limegreen", linewidth=4)
plt.xlabel('Reference SO2(ppm)', fontsize=18)
plt.ylabel('Calibrated SO2(ppm)', fontsize=18)
plt.legend(['Reg Line(Lab*0.4)', 'Reg Line(LR)', 'Reg Line(SVR)', 'Reg Line(RF)', 'Reg Line(ANN)', 'Reg Line(XGB)', '1:1 Ref'], loc = 2, bbox_to_anchor = (0,1))
plt.title('SO2 Sensor', fontsize=18 )
plt.grid(linestyle='-.', linewidth=0.3)

```



14 O3 CALIBRATION

```
[93]: import pandas as pd
import scipy.io
import numpy as np
mat= scipy.io.loadmat('O3.mat')
L_O3final=[]
for i in range(len(mat['O3final_1'])):
    L_O3final.append(list(mat['O3final_1'][i]))

O3_data=pd.DataFrame(L_O3final,columns=['Lab1',  
    'Lab2','Lab3','Lab4','Temp','RH','Ref','Time'])
Time=O3_data['Time'].to_list()
Time=np.array(Time)
Date=pd.to_datetime(Time-719529,unit='d').round('h')
O3_data['Date'] = Date.tolist()
O3_data=O3_data.set_index('Date')
O3_data.drop('Time',axis = 1, inplace = True)

mat = scipy.io.loadmat('O3_raw.mat')

L_O3_raw=[]
for i in range(len(mat['O3_raw'])):
    L_O3_raw.append(list(mat['O3_raw'][i]))

O3_raw=pd.DataFrame(L_O3_raw,columns=['WE', 'AE','Temp','RH','Time'])
O3_raw.head()
O3_data.insert(loc = 0,
               column = 'WE',
               value = O3_raw['WE'].to_list())

O3_data.insert(loc = 1,
               column = 'AE',
               value = O3_raw['AE'].to_list())

O3_data=O3_data.interpolate()
WE=np.array(O3_data['WE'].to_list())
AE=np.array(O3_data['AE'].to_list())
Signal=list(WE-AE)
O3_data.insert(loc = 2,
               column = 'Signal',
               value = Signal)
O3_data.head()
```

```
[93]:
```

Date	WE	AE	Signal	Lab1	\
2019-10-02 12:00:00	229.020000	232.625625	-3.605625	621.625704	
2019-10-02 13:00:00	284.284762	207.033333	77.251429	1835.510709	
2019-10-02 16:00:00	199.551940	131.729254	67.822687	2195.698656	
2019-10-02 17:00:00	234.016418	225.462090	8.554328	386.251890	
2019-10-03 16:00:00	234.671429	229.311429	5.360000	200.100352	

Date	Lab2	Lab3	Lab4	Temp	\
2019-10-02 12:00:00	67.962148	-12.822516	-987.063805	26.378438	
2019-10-02 13:00:00	293.683054	159.572059	-1331.518231	25.502791	
2019-10-02 16:00:00	615.567710	379.302767	-893.337914	30.827910	
2019-10-02 17:00:00	108.049071	34.461730	-696.227364	30.047164	
2019-10-03 16:00:00	122.550133	72.885162	-587.216200	29.441429	

Date	RH	Ref
2019-10-02 12:00:00	58.063437	52.449447
2019-10-02 13:00:00	59.868837	50.464425
2019-10-02 16:00:00	49.008060	37.972231
2019-10-02 17:00:00	51.259851	33.446343
2019-10-03 16:00:00	52.018571	33.071706

```
[94]:
```

```
03_data['Month']=03_data.index.month
03_data['Day_of_week']=03_data.index.dayofweek
03_data['Day']=03_data.index.day
03_data['Hour']=03_data.index.hour

03_data['Ref_CO']=CO_data['Ref'].to_list()
03_data['Ref_N02']=N02_data['Ref'].to_list()
03_data['Ref_S02']=S02_data['Ref'].to_list()
03_Data=03_data[['Signal','Ref_CO','Ref_N02','Ref_S02','Lab1','Temp','RH','Ref','Month','Day']]
03_Data=03_Data[(03_Data[03_Data.columns] >= 0).all(axis=1)]
#CO_data=CO_data.resample('D').mean()
03_Data=03_Data.dropna()
03_Data.shape
```

[94]: (733, 12)

14.1 Model 1: LR

```
[95]:
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error as mae
import sklearn.metrics as sm
```

```

import matplotlib.pyplot as plt
X=03_Data[['Signal','Ref_CO','Ref_N02','Ref_S02','Temp','RH','Month','Day_of_week','Day','Hour']]
y=03_Data['Ref']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
len(X_test)

```

[95]: 147

```

[96]: lr = LinearRegression()
model = lr.fit(X_train, y_train)
pred = model.predict(X_test)
lab1=03_Data['Lab1'].to_list()[len(y_train):]
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index =index)
Y_test
Pred=pd.Series(pred,index =index)
Lab1=pd.Series(lab1,index =index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_lr_03=sMAPE_lr
RMSE_lr_03=RMSE_lr/np.mean(np.array(y_test))
Pearson_lr_03=Pearson_lr
sMAPE_lab_03=sMAPE_lab
RMSE_lab_03=RMSE_lab/np.mean(np.array(lab1))
Pearson_lab_03=Pearson_lab
R2_lr_03=round(sm.r2_score(y_test, pred), 2)
R2_lab_03=round(sm.r2_score(y_test, lab1), 2)
RMSE_Lr_03=RMSE_lr
RMSE_Lab_03=RMSE_lab

A=len(y_test)
D=max(lab1)-0.2*max(lab1)
C=max(lab1)-0.1*max(lab1)
B=100

```

```

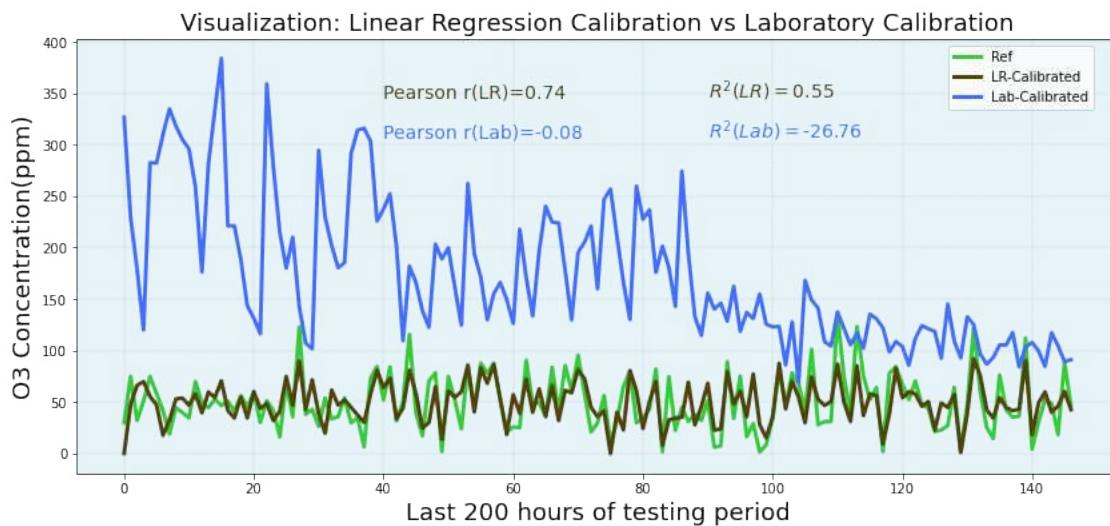
[97]: fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.plot(index,y_test, color='limegreen',linewidth=3)
plt.plot(index,pred, color='#513e00',linewidth=3)

```

```

plt.plot(index,lab1, color='#426eff', linewidth=3)
plt.legend(['Ref', 'LR-Calibrated', 'Lab-Calibrated'], loc = 2, bbox_to_anchor=(0.83,1))
plt.ylabel('O3 Concentration(ppm)', fontsize=18)
plt.text(B-10, C,r'$R^2(LR)=$'+str(R2_lr_03) , fontsize = 14, color="#513e00")
plt.text(B-10, D,r'$R^2(Lab)=$'+str(R2_lab_03) , fontsize = 14,color="#426eff")
plt.text(B-60, C, 'Pearson r(LR)='+str(Pearson_lr), fontsize = 14,color="#513e00")
plt.text(B-60, D, 'Pearson r(Lab)='+str(Pearson_lab), fontsize = 14,color="#426eff")
plt.xlabel('Last 200 hours of testing period',fontsize=18)
plt.title('Visualization: Linear Regression Calibration vs Laboratory Calibration', fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



```

[98]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred), 2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred), 2))
print("Explained variance score =", round(sm.explained_variance_score(y_test, pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))

```

```

MBE_LR_03=MBE(pred,y_test)/np.std(y_test)
CRMSE_LR_03=CRMSE(y_test,pred)/np.std(y_test)
MBE_LAB_03=MBE(lab1,y_test)/(3.6*np.std(y_test))
CRMSE_LAB_03=CRMSE(y_test,lab1)/(3.6*np.std(y_test))
pred_lr=pred

```

Regressor model performance:

Mean absolute error(MAE) = 14.87
 Mean squared error(MSE) = 344.54
 Median absolute error = 12.31
 Explain variance score = 0.55
 R2 score = 0.55

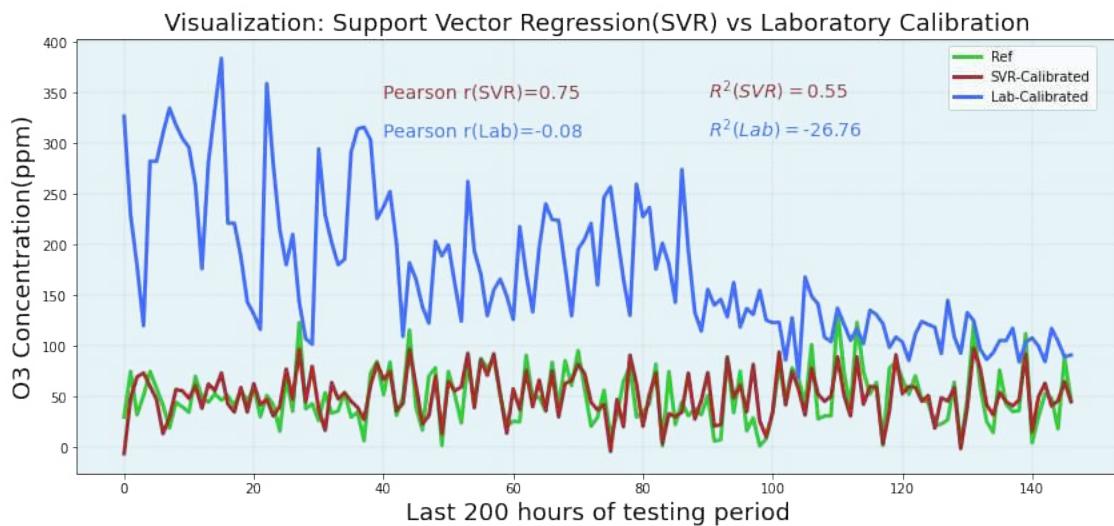
14.2 Model 2: SVR

```
[99]: from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
regressor = SVR(kernel = 'linear')
regressor.fit(X_train, y_train)
pred = regressor.predict(X_test)
```

```
[100]: lab1=O3_Data['Lab1'].to_list()[len(y_train):]
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index =index)
Y_test
Pred=pd.Series(pred,index =index)
Lab1=pd.Series(lab1,index =index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_svr_03=sMAPE_lr
RMSE_svr_03=RMSE_lr/np.mean(np.array(y_test))
Pearson_svr_03=Pearson_lr
R2_svr_03=round(sm.r2_score(y_test, pred), 2)
RMSE_Svr_03=RMSE_lr

A=len(y_test)
D=max(lab1)-0.2*max(lab1)
C=max(lab1)-0.1*max(lab1)
B=100
```

```
[101]: fig= plt.figure(figsize=(14,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.plot(index,y_test, color='limegreen', linewidth=3)
plt.plot(index,pred, color='brown', linewidth=3)
plt.plot(index,lab1, color='#426eff', linewidth=3)
plt.legend(['Ref', 'SVR-Calibrated', 'Lab-Calibrated'], loc = 2, bbox_to_anchor=(0.83,1))
plt.ylabel('O3 Concentration(ppm)', fontsize=18)
plt.text(B-10, C,r'$R^2$(SVR)='+str(R2_svr_03) , fontsize = 14, color='brown')
plt.text(B-10, D,r'$R^2$(Lab)='+str(R2_lab_03) , fontsize = 14, color='brown')
plt.text(B-60, C, 'Pearson r(SVR)='+str(Pearson_lr), fontsize = 14, color='brown')
plt.text(B-60, D, 'Pearson r(Lab)='+str(Pearson_lab), fontsize = 14, color='brown')
plt.xlabel('Last 200 hours of testing period', fontsize=18)
plt.title('Visualization: Support Vector Regression(SVR) vs Laboratory Calibration', fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()
```



```
[102]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred), 2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred), 2))
```

```

print("Median absolute error =", round(sm.median_absolute_error(y_test, pred), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_test, pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))

MBE_SVR_03=MBE(pred,y_test)/np.std(y_test)
CRMSE_SVR_03=CRMSE(y_test,pred)/np.std(y_test)
pred_svr=pred

```

Regressor model performance:

```

Mean absolute error(MAE) = 14.85
Mean squared error(MSE) = 343.49
Median absolute error = 11.88
Explain variance score = 0.55
R2 score = 0.55

```

14.3 Model 3 : Random Forest

```
[103]: from sklearn.ensemble import RandomForestRegressor

# create regressor object
regressor = RandomForestRegressor(n_estimators = 500, random_state = 0)

# fit the regressor with x and y data
regressor.fit(X_train, y_train)
```

```
[103]: RandomForestRegressor(n_estimators=500, random_state=0)
```

```
[104]: features_03=regressor.feature_importances_
pred = regressor.predict(X_test)
lab1=03_Data['Lab1'].to_list()[len(y_train):]
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index =index)
Y_test
Pred=pd.Series(pred,index =index)
Lab1=pd.Series(lab1,index =index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_rf_03=sMAPE_lr
RMSE_rf_03=RMSE_lr/np.mean(np.array(y_test))
```

```

Pearson_rf_03=Pearson_lr
R2_rf_03=round(sm.r2_score(y_test, pred), 2)
RMSE_Rf_03=RMSE_lr

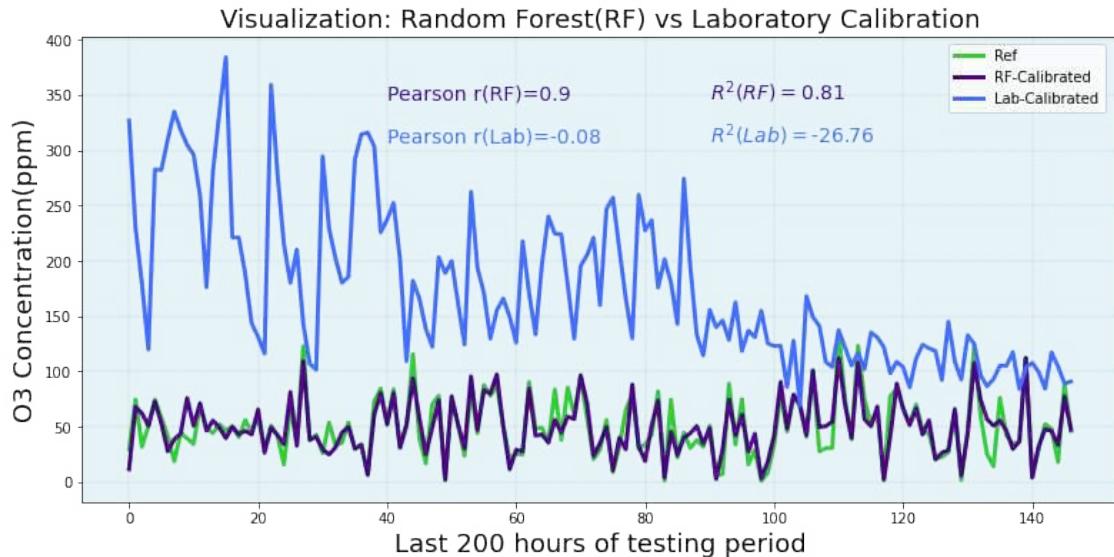
A=len(y_test)
D=max(lab1)-0.2*max(lab1)
C=max(lab1)-0.1*max(lab1)
B=100

```

```

[105]: fig= plt.figure(figsize=(13,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
ax.plot(index,y_test, color='limegreen', linewidth=3)
ax.plot(index,pred, color='indigo', linewidth=3)
ax.plot(index,lab1, color='#426eff', linewidth=3)
plt.legend(['Ref', 'RF-Calibrated', 'Lab-Calibrated'], loc = 2, bbox_to_anchor=(0.83,1))
plt.ylabel('03 Concentration(ppm)', fontsize=18)
plt.text(B-10, C,r'$R^2$ (RF)=' +str(R2_rf_03) , fontsize = 14, color='indigo')
plt.text(B-10, D,r'$R^2$ (Lab)=' +str(R2_lab_03) , fontsize = 14,color='#426eff')
plt.text(B-60, C, 'Pearson r(RF)=' +str(Pearson_lr), fontsize = 14,color='indigo')
plt.text(B-60, D, 'Pearson r(Lab)=' +str(Pearson_lab), fontsize = 14,color='#426eff')
plt.xlabel('Last 200 hours of testing period', fontsize=18)
plt.title('Visualization: Random Forest(RF) vs Laboratory Calibration', fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



```
[106]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred), 2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred), 2))
print("Explained variance score =", round(sm.explained_variance_score(y_test, pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))

MBE_RF_03=MBE(pred,y_test)/np.std(y_test)
CRMSE_RF_03=CRMSE(y_test,pred)/np.std(y_test)
pred_rf=pred
```

Regressor model performance:
 Mean absolute error(MAE) = 8.4
 Mean squared error(MSE) = 148.34
 Median absolute error = 4.51
 Explained variance score = 0.81
 R2 score = 0.81

14.4 Model 4: ANN

```
[107]: from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
from sklearn.preprocessing import StandardScaler
model = Sequential()
model.add(Dense(6, input_shape = (10,),kernel_initializer='normal', activation='linear'))
model.add(Dense(128,kernel_initializer='normal', activation= 'relu'))
model.add(Dense(128, kernel_initializer='normal',activation= 'relu'))
model.add(Dense(100, kernel_initializer='normal',activation= 'relu'))
model.add(Dense(1,kernel_initializer='normal',activation='linear',))
sgd = optimizers.Adam(learning_rate=0.01)

model.compile(optimizer = sgd, loss = 'mean_squared_error', metrics= ['mse','mae'])
model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
dense_14 (Dense)	(None, 6)	66
dense_15 (Dense)	(None, 128)	896
dense_16 (Dense)	(None, 128)	16512
dense_17 (Dense)	(None, 100)	12900
dense_18 (Dense)	(None, 1)	101

Total params: 30,475

Trainable params: 30,475

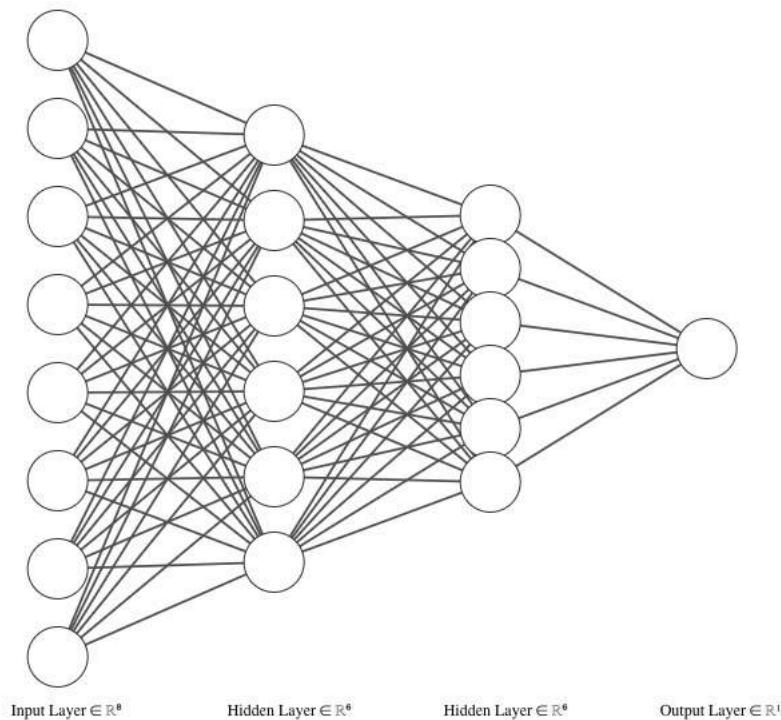
Non-trainable params: 0

```
[108]: scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled=scaler.transform(X_train)
X_test_scaled=scaler.transform(X_test)
model.fit(X_train_scaled, y_train, batch_size= 100, epochs=200, verbose= 0)
```

```
[108]: <tensorflow.python.keras.callbacks.History at 0x1415a4880>
```

```
[109]: from IPython.display import Image  
Image('NN.png')
```

[109]:



```
[110]: train_pred = model.predict(X_train_scaled)  
test_pred = model.predict(X_test_scaled)  
pred=[]  
for i in range(len(test_pred)):  
    pred.append(sum(list(test_pred[i])))  
len(y_test)
```

[110]: 147

```
[111]: lab1=03_Data['Lab1'].to_list()[len(y_train):]  
index=[i for i in range(len(y_test))]  
Y_test=y_test.to_list()  
Y_test=pd.Series(Y_test,index =index)  
Y_test  
Pred=pd.Series(pred,index =index)  
Lab1=pd.Series(lab1,index =index)  
sMAPE_lr=round(smape_loss(Y_test,Pred),2)  
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)  
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)  
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
```

```

Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_ann_03=sMAPE_lr
RMSE_ann_03=RMSE_lr/np.mean(np.array(y_test))
Pearson_ann_03=Pearson_lr
R2_ann_03=round(sm.r2_score(y_test, pred), 2)
RMSE_Ann_03=RMSE_lr

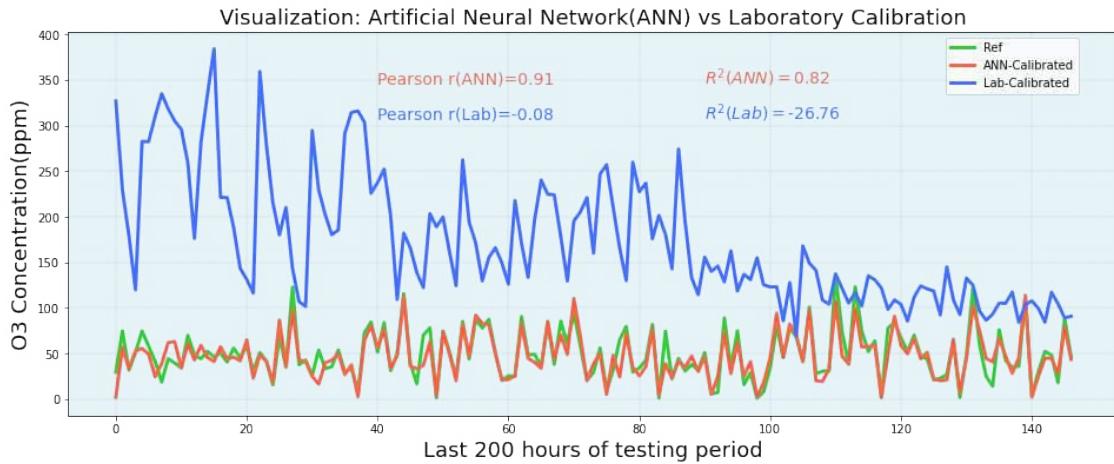
A=len(y_test)
D=max(lab1)-0.2*max(lab1)
C=max(lab1)-0.1*max(lab1)
B=100

```

```

[112]: fig= plt.figure(figsize=(16,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
ax.plot(index,y_test, color='limegreen', linewidth=3)
ax.plot(index,pred, color='tomato', linewidth=3)
ax.plot(index,lab1, color="#426eff", linewidth=3)
plt.legend(['Ref', 'ANN-Calibrated', 'Lab-Calibrated'], loc = 2, bbox_to_anchor=[0.83,1])
plt.ylabel('O3 Concentration(ppm)', fontsize=18)
plt.text(B-10, C,r'$R^2(ANN)=$'+str(R2_ann_03) , fontsize = 14, color='tomato')
plt.text(B-10, D, r'$R^2(Lab)=$'+str(R2_lab_03), fontsize = 14, color="#426eff")
plt.text(B-60, C, 'Pearson r(ANN)='+str(Pearson_lr), fontsize = 14, color='tomato')
plt.text(B-60, D, 'Pearson r(Lab)='+str(Pearson_lab), fontsize = 14, color="#426eff")
plt.xlabel('Last 200 hours of testing period',fontsize=18)
plt.title('Visualization: Artificial Neural Network(ANN) vs Laboratory Calibration',fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



```
[113]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred), 2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred), 2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred), 2))
print("Explain variance score =", round(sm.explained_variance_score(y_test, pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))

MBE_AN3=MBE(pred,y_test)/np.std(y_test)
CRMSE_AN3=CRMSE(y_test,pred)/np.std(y_test)
pred_ANN=pred
```

Regressor model performance:
 Mean absolute error(MAE) = 8.35
 Mean squared error(MSE) = 137.96
 Median absolute error = 5.35
 Explain variance score = 0.83
 R2 score = 0.82

15 Model 5: XGBoost

```
[114]: from xgboost import XGBRegressor
from numpy import absolute
from pandas import read_csv
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
```

```
# create an xgboost regression model
#n_estimators=10000, max_depth=5, eta=0.01, subsample=0.9,colsample_bytree=0.
→4,alpha=10
model = XGBRegressor(n_estimators=10000, max_depth=5, eta=0.01, subsample=0.9,
                      colsample_bytree=0.4,alpha=10)

model.fit(X_train,y_train)
```

[114]: XGBRegressor(alpha=10, base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.4, eta=0.01, gamma=0, gpu_id=-1, importance_type='gain', interaction_constraints='', learning_rate=0.00999999978, max_delta_step=0, max_depth=5, min_child_weight=1, missing=nan, monotone_constraints='()', n_estimators=10000, n_jobs=4, num_parallel_tree=1, random_state=0, reg_alpha=10, reg_lambda=1, scale_pos_weight=1, subsample=0.9, tree_method='exact', validate_parameters=1, verbosity=None)

```
[115]: pred = model.predict(X_test)
index=[i for i in range(len(y_test))]
Y_test=y_test.to_list()
Y_test=pd.Series(Y_test,index =index)
Y_test
Pred=pd.Series(pred,index =index)
Lab1=pd.Series(lab1,index =index)
sMAPE_lr=round(smape_loss(Y_test,Pred),2)
sMAPE_lab=round (smape_loss(Y_test,Lab1),2)
RMSE_lr=round(np.sqrt(sm.mean_squared_error(y_test, pred)),1)
RMSE_lab=round(np.sqrt(sm.mean_squared_error(y_test, lab1)),1)
Pearson_lr=round(np.corrcoef(y_test, pred)[0, 1],2)
Pearson_lab=round(np.corrcoef(y_test, lab1)[0, 1],2)
sMAPE_xgb_03=sMAPE_lr
RMSE_xgb_03=RMSE_lr/np.mean(np.array(y_test))
Pearson_xgb_03=Pearson_lr
R2_xgb_03=round(sm.r2_score(y_test, pred), 2)
RMSE_Xgb_03=RMSE_lr

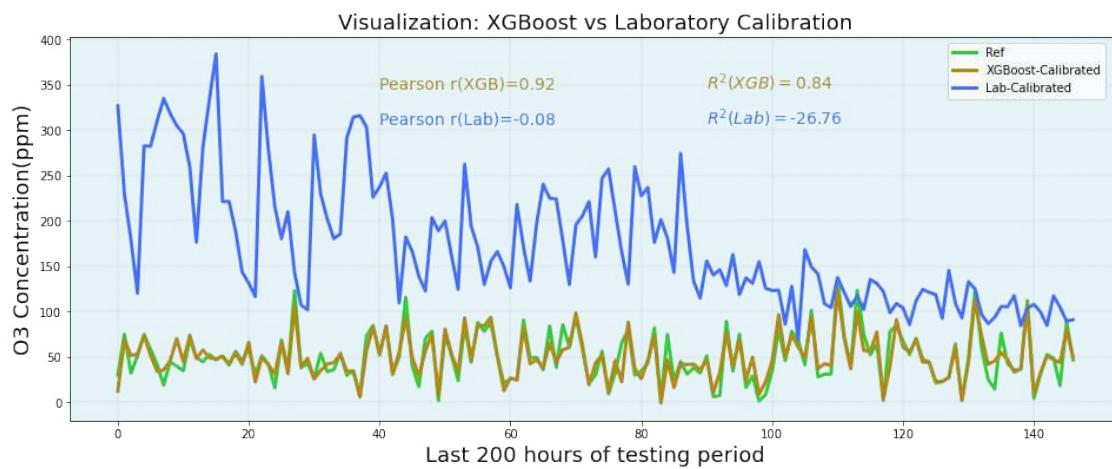
A=len(y_test)
D=max(lab1)-0.2*max(lab1)
C=max(lab1)-0.1*max(lab1)
B=100
```

```
[116]: fig= plt.figure(figsize=(16,6))
index=[i for i in range(len(y_test))]
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
```

```

ax.plot(index,y_test, color='limegreen',linewidth=3)
ax.plot(index,pred, color='darkgoldenrod',linewidth=3)
ax.plot(index,lab1, color='#426eff',linewidth=3)
plt.legend(['Ref', 'XGBoost-Calibrated', 'Lab-Calibrated'], loc = 2,
           bbox_to_anchor = (0.83,1))
plt.ylabel('O3 Concentration(ppm)',fontsize=18)
plt.text(B-10, C,r'$R^2(XGB)=$'+str(R2_xgb_03) , fontsize = 14,
         color='darkgoldenrod')
plt.text(B-10, D,r'$R^2(Lab)=$'+str(R2_lab_03) , fontsize = 14,
         color='#426eff')
plt.text(B-60, C, 'Pearson r(XGB)='+str(Pearson_lr), fontsize = 14,
         color='darkgoldenrod')
plt.text(B-60, D, 'Pearson r(Lab)='+str(Pearson_lab), fontsize = 14,
         color='#426eff')
plt.xlabel('Last 200 hours of testing period',fontsize=18)
plt.title('Visualization: XGBoost vs Laboratory Calibration',fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



```

[117]: print("Regressor model performance:")
print("Mean absolute error(MAE) =", round(sm.mean_absolute_error(y_test, pred),
                                           2))
print("Mean squared error(MSE) =", round(sm.mean_squared_error(y_test, pred),
                                           2))
print("Median absolute error =", round(sm.median_absolute_error(y_test, pred),
                                           2))
print("Explained variance score =", round(sm.explained_variance_score(y_test,
                                                                       pred), 2))
print("R2 score =", round(sm.r2_score(y_test, pred), 2))

```

```

MBE_XGB_03=MBE(pred,y_test)/np.std(y_test)
CRMSE_XGB_03=CRMSE(y_test,pred)/np.std(y_test)
pred_xgb=pred

```

Regressor model performance:

Mean absolute error(MAE) = 7.8
 Mean squared error(MSE) = 123.46
 Median absolute error = 5.4
 Explain variance score = 0.84
 R2 score = 0.84

```

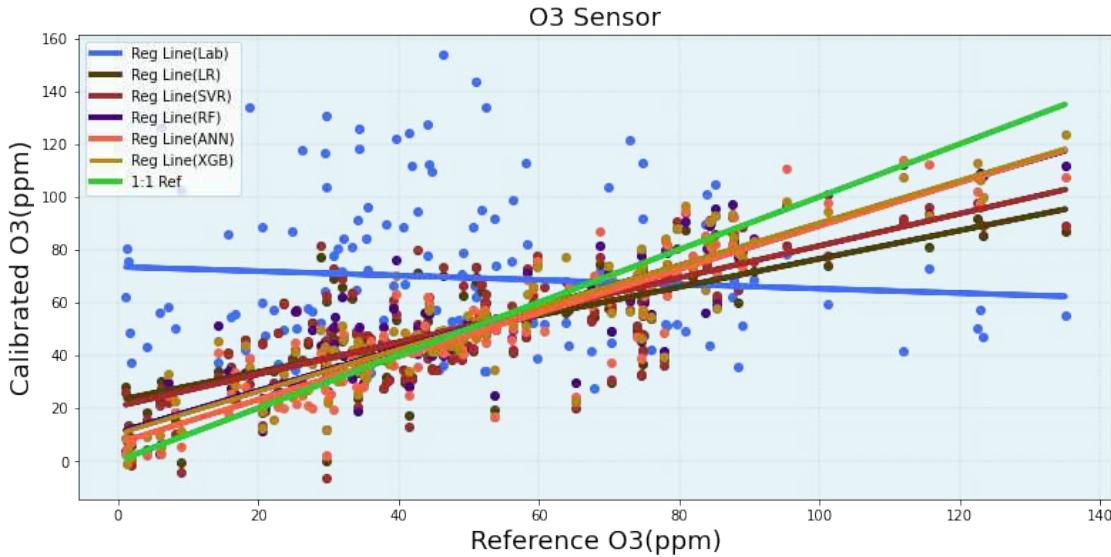
[118]: fig= plt.figure(figsize=(13,6))

ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)

m0, b0 = np.polyfit(np.array(y_test),0.4*np.array(lab1), 1)
plt.plot(np.array(y_test), m0*np.array(y_test) + b0,color="#426eff",linewidth=4)
m1, b1 = np.polyfit(np.array(y_test), np.array(pred_lr), 1)
plt.plot(np.array(y_test), m1*np.array(y_test) + b1,color="#513e00",linewidth=4)
m2, b2 = np.polyfit(np.array(y_test), np.array(pred_svr), 1)
plt.plot(np.array(y_test), m2*np.array(y_test) + b2,color='brown',linewidth=4)
m3, b3 = np.polyfit(np.array(y_test), np.array(pred_rf), 1)
plt.plot(np.array(y_test), m3*np.array(y_test) + b3,color='indigo',linewidth=4)
m4, b4 = np.polyfit(np.array(y_test), np.array(pred_ann), 1)
plt.plot(np.array(y_test), m4*np.array(y_test) + b4,color='tomato',linewidth=4)
m5, b5 = np.polyfit(np.array(y_test), np.array(pred_xgb), 1)
plt.plot(np.array(y_test), m5*np.array(y_test) + b5,color='darkgoldenrod',linewidth=3)

plt.scatter(np.array(y_test),0.4*np.array(lab1),color="#426eff")
plt.scatter(np.array(y_test),np.array(pred_lr),color="#513e00")
plt.scatter(np.array(y_test),np.array(pred_svr),color='brown')
plt.scatter(np.array(y_test),np.array(pred_rf),color='indigo')
plt.scatter(np.array(y_test),np.array(pred_ann),color='tomato')
plt.scatter(np.array(y_test),np.array(pred_xgb),color='darkgoldenrod')
ax.plot(y_test,y_test, c ="limegreen",linewidth=4)
plt.xlabel('Reference O3(ppm)',fontsize=18)
plt.ylabel('Calibrated O3(ppm)',fontsize=18)
plt.legend(['Reg Line(Lab)', 'Reg Line(LR)', 'Reg Line(SVR)', 'Reg Line(RF)', 'Reg Line(ANN)', 'Reg Line(XGB)', '1:1 Ref'], loc = 2, bbox_to_anchor = (0,1))
plt.title('O3 Sensor',fontsize=18 )
plt.grid(linestyle='-.', linewidth=0.3)

```



16 Data Analytics

```
[119]: import chart_studio
import plotly.express as px
from IPython.display import Image
import plotly.graph_objects as go
import numpy as np

LAB_PR=[Pearson_lab_CO,Pearson_lab_N02,Pearson_lab_S02,Pearson_lab_03]
LR_PR=[Pearson_lr_CO,Pearson_lr_N02,Pearson_lr_S02,Pearson_lr_03]
SVR_PR=[Pearson_svr_CO,Pearson_svr_N02,Pearson_svr_S02,Pearson_svr_03]
RF_PR=[Pearson_rf_CO,Pearson_rf_N02,Pearson_rf_S02,Pearson_rf_03]
ANN_PR=[Pearson_ann_CO,Pearson_ann_N02,Pearson_ann_S02,Pearson_ann_03]
XGB_PR=[Pearson_xgb_CO,Pearson_xgb_N02,Pearson_xgb_S02,Pearson_xgb_03]

LAB_SM=[sMAPE_lab_CO,sMAPE_lab_N02,sMAPE_lab_S02,sMAPE_lab_03]
LR_SM=[sMAPE_lr_CO,sMAPE_lr_N02,sMAPE_lr_S02,sMAPE_lr_03]
SVR_SM=[sMAPE_svr_CO,sMAPE_svr_N02,sMAPE_svr_S02,sMAPE_svr_03]
RF_SM=[sMAPE_rf_CO,sMAPE_rf_N02,sMAPE_rf_S02,sMAPE_rf_03]
ANN_SM=[sMAPE_ann_CO,sMAPE_ann_N02,sMAPE_ann_S02,sMAPE_ann_03]
XGB_SM=[sMAPE_xgb_CO,sMAPE_xgb_N02,sMAPE_xgb_S02,sMAPE_xgb_03]

LAB_RM=[RMSE_lab_CO,RMSE_lab_N02,RMSE_lab_S02,RMSE_lab_03]
LR_RM=[RMSE_lr_CO,RMSE_lr_N02,RMSE_lr_S02,RMSE_lr_03]
SVR_RM=[RMSE_svr_CO,RMSE_svr_N02,RMSE_svr_S02,RMSE_svr_03]
```

```

RF_RM=[RMSE_rf_C0,RMSE_rf_N02,RMSE_rf_S02,RMSE_rf_O3]
ANN_RM=[RMSE_ann_C0,RMSE_ann_N02,RMSE_ann_S02,RMSE_ann_O3]
XGB_RM=[RMSE_xgb_C0,RMSE_xgb_N02,RMSE_xgb_S02,RMSE_xgb_O3]

PR=LAB_PR+LR_PR+SVR_PR+RF_PR+ANN_PR+XGB_PR
SM=LAB_SM+LR_SM+SVR_SM+RF_SM+ANN_SM+XGB_SM
RM=LAB_RM+LR_RM+SVR_RM+RF_RM+ANN_RM+XGB_RM
x1=['LAB' for i in range(4)]
x2=['LR' for i in range(4)]
x3=['SVR' for i in range(4)]
x4=['RF' for i in range(4)]
x5=['ANN' for i in range(4)]
x6=['XGBoost' for i in range(4)]

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=PR,
    x=x,
    name='Pearson r',
    marker_color='#426eff'
))

fig.add_trace(go.Box(
    y=SM,
    x=x,
    name='sMAPE',
    marker_color='orange'
))

fig.add_trace(go.Box(
    y=RM,
    x=x,
    name='NRMSE',
    marker_color='teal'
))

fig.update_layout(autosize=False,
    width=1100,
)

```

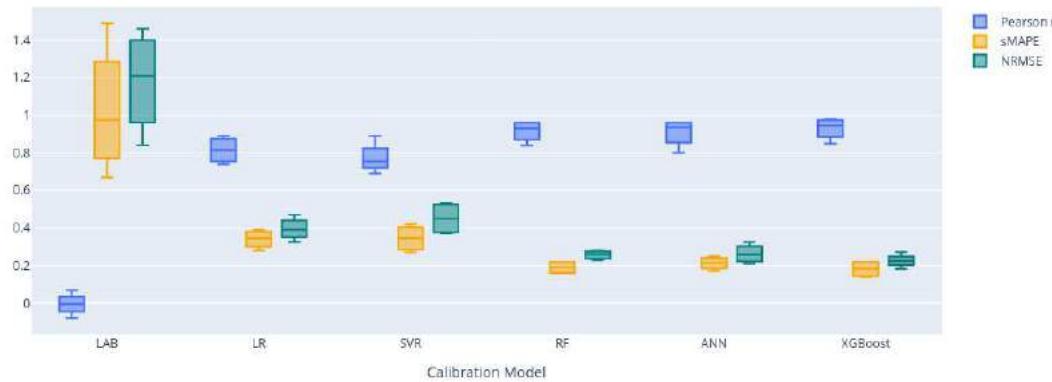
```

height=500,
# group together boxes of the different
# traces for each value of x
boxmode='group'
)

fig.update_xaxes(title_text="Calibration Model")
chart_studio.plotly.sign_in('vinylango', 'gybbJWWfRSUoTcRRSa6J')
chart_studio.plotly.image.save_as(fig, filename='models_boxplot.png')
Image('models_boxplot.png')

```

[119]:



[120]: #Violin plot which also show the density of the distribution

```

import plotly.express as px
Metric1=['Pearson r' for i in range(len(PR))]
Metric2=['RMSE' for i in range(len(RM))]
Metric3=['sMAPE' for i in range(len(RM))]
Metric=Metric1+Metric2+Metric3
Model=x+x+x
Values=PR+RM+SM
lst=[[Model[i],Values[i],Metric[i]] for i in range(len(Model))]
df = pd.DataFrame(lst, columns=['Calibration Model', 'Performance', 'Metric'])

#fig = px.violin( df,y="Performance", x="Calibration Model", color='Metric', box=True,points="all",
#                hover_data=df.columns)
fig = px.violin( df,y="Performance", x="Calibration Model", color='Metric', box=True,points="all",
                hover_data=df.columns)

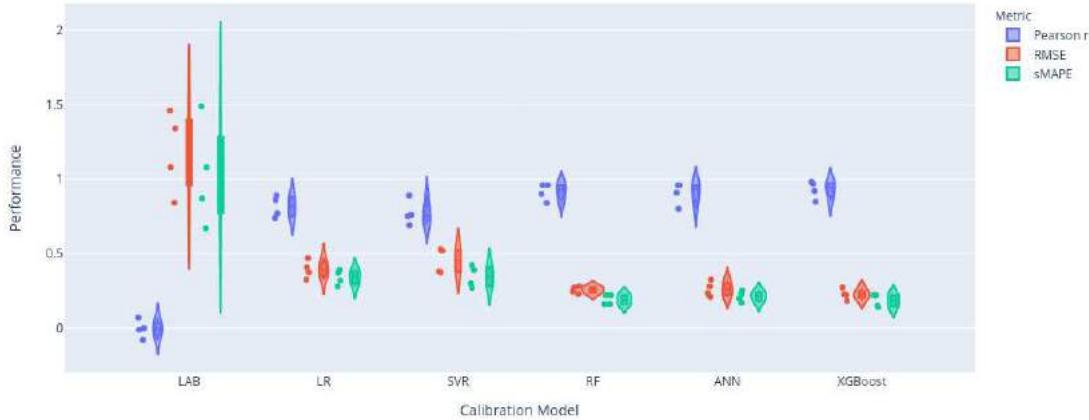
```

```

fig.update_layout(autosize=False,
                  width=1100,
                  height=500)
#fig.show()
chart_studio.plotly.sign_in('vinylango', 'gybbJWWfRSUoTcRRSa6J')
chart_studio.plotly.image.save_as(fig, filename='models_violinplots.png')
Image('models_violinplots.png')

```

[120] :



16.1 Target Diagrams

```

[121]: theta = np.linspace( 0 , 2 * np.pi , 150 )

radius = 1

a = radius * np.cos( theta )
b = radius * np.sin( theta )

CRMSE_LAB=[CRMSE_LAB_03,CRMSE_LAB_CO,CRMSE_LAB_N02,CRMSE_LAB_S02]
CRMSE_LR=[CRMSE_LR_03,CRMSE_LR_CO,CRMSE_LR_N02,CRMSE_LR_S02]
CRMSE_SVR=[CRMSE_SVR_03,CRMSE_SVR_CO,CRMSE_SVR_N02,CRMSE_SVR_S02]
CRMSE_RF=[CRMSE_RF_03,CRMSE_RF_CO,CRMSE_RF_N02,CRMSE_RF_S02]
CRMSE_ANN=[CRMSE_ANN_03,CRMSE_ANN_CO,CRMSE_ANN_N02,CRMSE_ANN_S02]
CRMSE_XGB=[CRMSE_XGB_03,CRMSE_XGB_CO,CRMSE_XGB_N02,CRMSE_XGB_S02]

MBE_LAB=[MBE_LAB_03,MBE_LAB_CO,MBE_LAB_N02,MBE_LAB_S02]
MBE_LR=[MBE_LR_03,MBE_LR_CO,MBE_LR_N02,MBE_LR_S02]
MBE_SVR=[MBE_SVR_03,MBE_SVR_CO,MBE_SVR_N02,MBE_SVR_S02]

```

```

MBE_RF=[MBE_RF_03,MBE_RF_CO,MBE_RF_N02,MBE_RF_S02]
MBE ANN=[MBE ANN_03,MBE ANN_CO,MBE ANN_N02,MBE ANN_S02]
MBE_XGB=[MBE_XGB_03,MBE_XGB_CO,MBE_XGB_N02,MBE_XGB_S02]

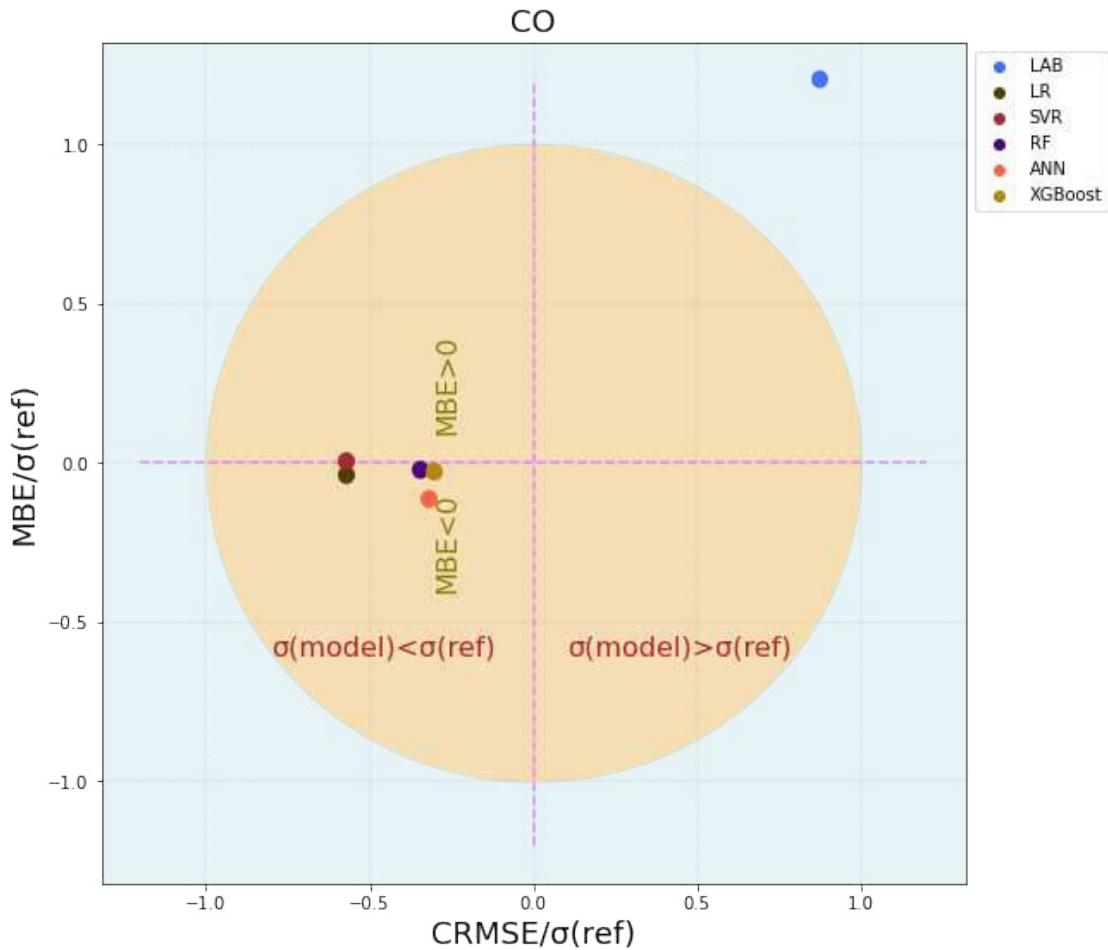
```

```

[122]: fig= plt.figure(figsize=(9,9))
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.text(CRMSE_LAB[0],MBE_LAB[0] , '•', rotation=90, va='center',fontsize = 36, color="#426eff")
plt.text(CRMSE_LR[0],MBE_LR[0] , '•', rotation=90, va='center',fontsize = 36, color="#513e00")
plt.text(CRMSE_SVR[0],MBE_SVR[0] , '•', rotation=90, va='center',fontsize = 36, color='brown')
plt.text(CRMSE_RF[0],MBE_RF[0] , '•', rotation=90, va='center',fontsize = 36, color='indigo')
plt.text(CRMSE_ANN[0],MBE_ANN[0] , '•', rotation=90, va='center',fontsize = 36, color='tomato')
plt.text(CRMSE_XGB[0],MBE_XGB[0] , '•', rotation=90, va='center',fontsize = 36, color='darkgoldenrod')
plt.scatter(CRMSE_LAB[2]/1.1,MBE_LAB[2],color="#426eff")
plt.scatter(CRMSE_LR,MBE_LR,color="#513e00")
plt.scatter(CRMSE_SVR[0],MBE_SVR[0],color='brown')
plt.scatter(CRMSE_RF,MBE_RF,color='indigo')
plt.scatter(CRMSE_ANN,MBE_ANN,color='tomato')
plt.scatter(CRMSE_XGB,MBE_XGB,color='darkgoldenrod')
plt.legend(['LAB','LR','SVR','RF','ANN','XGBoost'],loc = 2, bbox_to_anchor =(1,1))

plt.Circle((0, 0), 1, color='wheat')
plt.vlines([0], -1.2, 1.2, linestyles='dashed',color='violet')
plt.hlines([0], -1.2, 1.2, linestyles='dashed', color='violet')
plt.text(-0.8, -0.6, '(model)<(ref)', fontsize = 16, color='firebrick')
plt.text(0.1, -0.6, '(model)>(ref)', fontsize = 16, color='firebrick')
plt.text(-0.3, -0.25, 'MBE<0', rotation=90, va='center',fontsize = 16, color='olive')
plt.text(-0.3, 0.25, 'MBE>0', rotation=90, va='center',fontsize = 16, color='olive')
plt.fill_between(a, b, color='wheat')
plt.xlabel('CRMSE/ (ref)',fontsize=18)
plt.ylabel('MBE/ (ref)',fontsize=18)
plt.title('CO',fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```

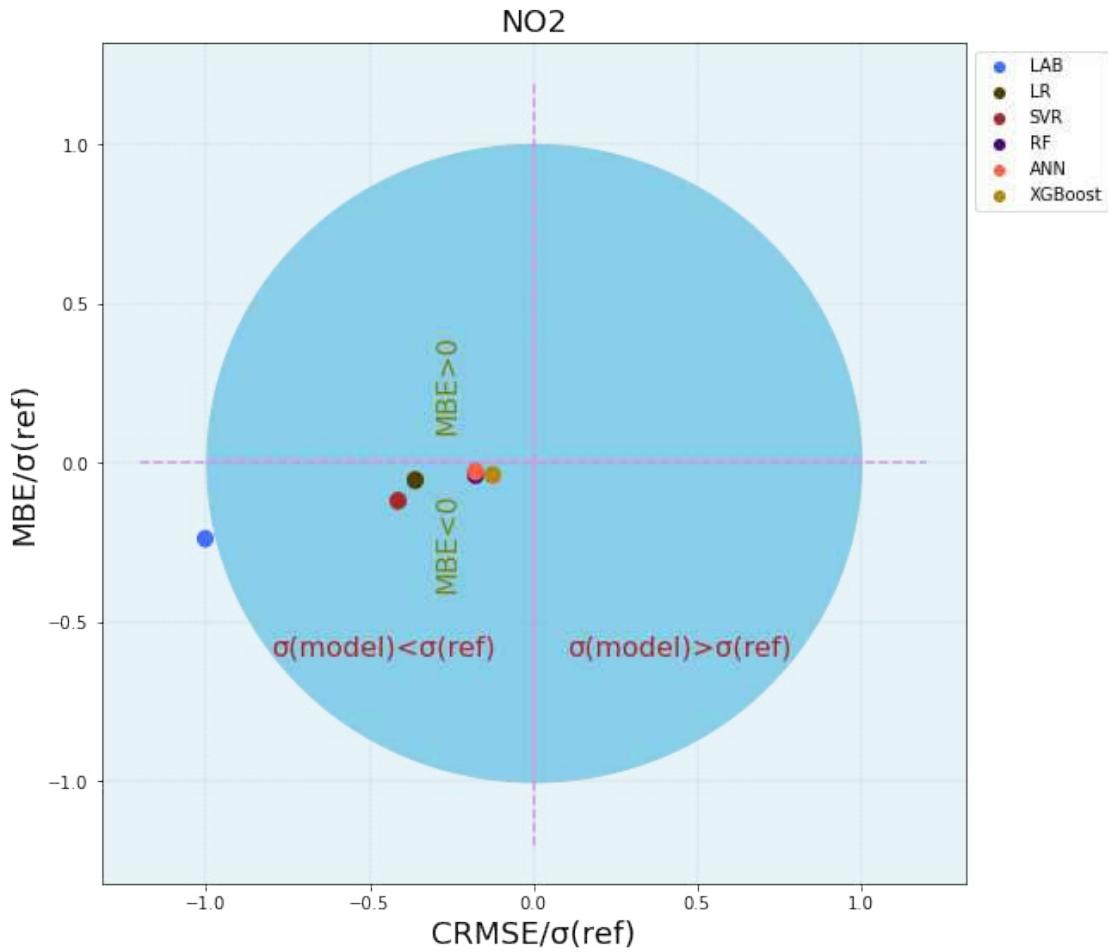


```
[123]: fig= plt.figure(figsize=(9,9))
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.text(CRMSE_LAB[1],MBE_LAB[1] , '•', rotation=90, va='center', fontsize = 36,
         color="#426eff")
plt.text(CRMSE_LR[1],MBE_LR[1] , '•', rotation=90, va='center', fontsize = 36,
         color="#513e00")
plt.text(CRMSE_SVR[1],MBE_SVR[1] , '•', rotation=90, va='center', fontsize = 36,
         color='brown')
plt.text(CRMSE_RF[1],MBE_RF[1] , '•', rotation=90, va='center', fontsize = 36,
         color='indigo')
plt.text(CRMSE_ANN[1],MBE_ANN[1] , '•', rotation=90, va='center', fontsize = 36,
         color='tomato')
plt.text(CRMSE_XGB[1],MBE_XGB[1] , '•', rotation=90, va='center', fontsize = 36,
         color='darkgoldenrod')
plt.scatter(CRMSE_LAB[2]/1.1,MBE_LAB[2],color="#426eff")
```

```

plt.scatter(CRMSE_LR, MBE_LR, color="#513e00")
plt.scatter(CRMSE_SVR[0], MBE_SVR[0], color='brown')
plt.scatter(CRMSE_RF, MBE_RF, color='indigo')
plt.scatter(CRMSE_ANN, MBE_ANN, color='tomato')
plt.scatter(CRMSE_XGB, MBE_XGB, color='darkgoldenrod')
plt.legend(['LAB', 'LR', 'SVR', 'RF', 'ANN', 'XGBoost'], loc = 2, bbox_to_anchor =(1,1))
plt.Circle((0, 0), 1, color='wheat')
plt.vlines([0], -1.2, 1.2, linestyles='dashed',color='violet')
plt.hlines([0], -1.2, 1.2, linestyles='dashed', color='violet')
plt.text(-0.8, -0.6, ' (model)< (ref)', fontsize = 16, color='firebrick')
plt.text(0.1, -0.6, ' (model)> (ref)', fontsize = 16, color='firebrick')
plt.text(-0.3, -0.25, 'MBE<0', rotation=90, va='center',fontsize = 16,color='olive')
plt.text(-0.3, 0.25, 'MBE>0', rotation=90, va='center',fontsize = 16,color='olive')
plt.fill_between(a, b, color='skyblue')
plt.xlabel('CRMSE/ (ref)', fontsize=18)
plt.ylabel('MBE/ (ref)', fontsize=18)
plt.title('NO2', fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```

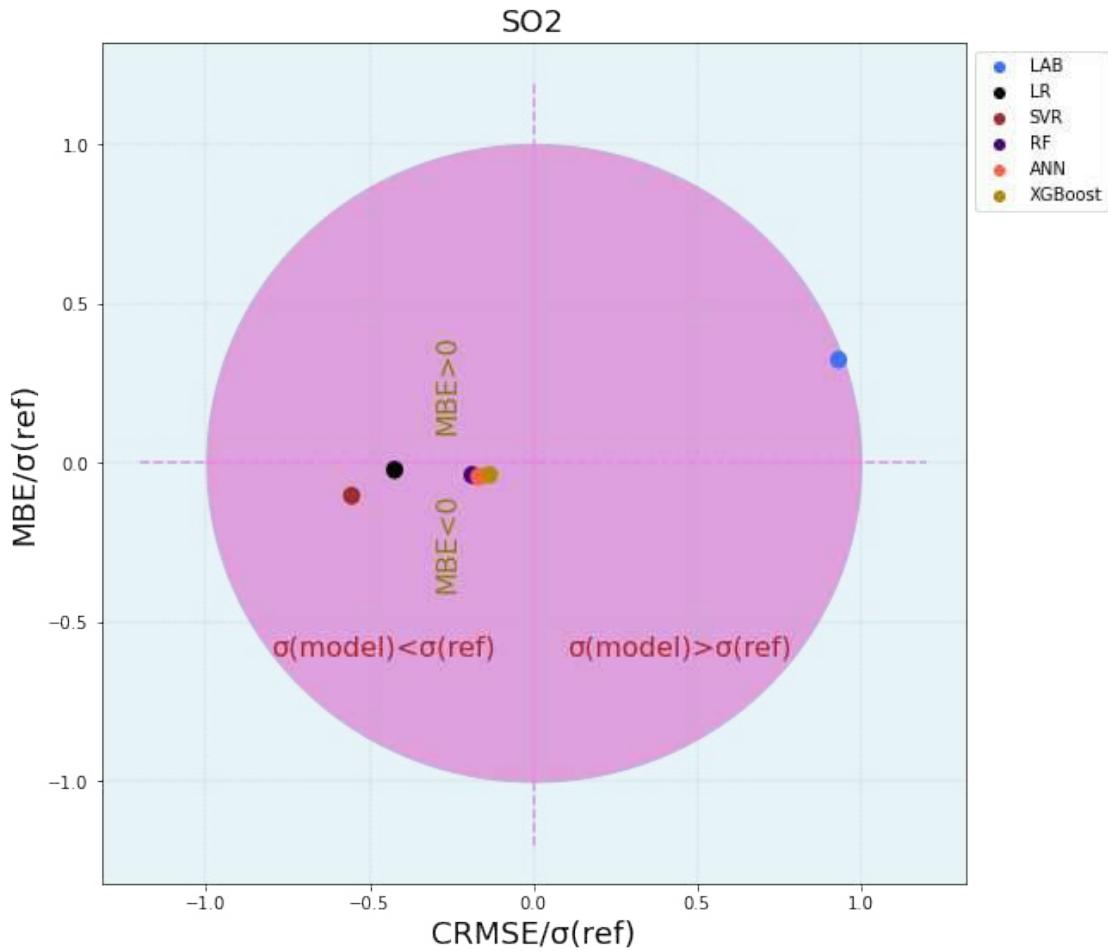


```
[124]: fig= plt.figure(figsize=(9,9))
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.text(CRMSE_LAB[2],MBE_LAB[2] , '•', rotation=90, va='center', fontsize = 36,
         color="#426eff")
plt.text(CRMSE_LR[2],MBE_LR[2] , '•', rotation=90, va='center', fontsize = 36,
         color='black')
plt.text(CRMSE_SVR[2],MBE_SVR[2] , '•', rotation=90, va='center', fontsize = 36,
         color='brown')
plt.text(CRMSE_RF[2],MBE_RF[2] , '•', rotation=90, va='center', fontsize = 36,
         color='indigo')
plt.text(CRMSE_ANN[2],MBE_ANN[2] , '•', rotation=90, va='center', fontsize = 36,
         color='tomato')
plt.text(CRMSE_XGB[2],MBE_XGB[2] , '•', rotation=90, va='center', fontsize = 36,
         color='darkgoldenrod')
plt.scatter(CRMSE_LAB[2]/1.1,MBE_LAB[2],color="#426eff")
```

```

plt.scatter(CRMSE_LR, MBE_LR, color='black')
plt.scatter(CRMSE_SVR[0], MBE_SVR[0], color='brown')
plt.scatter(CRMSE_RF, MBE_RF, color='indigo')
plt.scatter(CRMSE_ANN, MBE_ANN, color='tomato')
plt.scatter(CRMSE_XGB, MBE_XGB, color='darkgoldenrod')
plt.legend(['LAB', 'LR', 'SVR', 'RF', 'ANN', 'XGBoost'], loc = 2, bbox_to_anchor =(1,1))
plt.Circle((0, 0), 1, color='wheat')
plt.vlines([0], -1.2, 1.2, linestyles='dashed', color='violet')
plt.hlines([0], -1.2, 1.2, linestyles='dashed', color='violet')
plt.text(-0.8, -0.6, '(model)<(ref)', fontsize = 16, color='firebrick')
plt.text(0.1, -0.6, '(model)>(ref)', fontsize = 16, color='firebrick')
plt.text(-0.3, -0.25, 'MBE<0', rotation=90, va='center', fontsize = 16, color='olive')
plt.text(-0.3, 0.25, 'MBE>0', rotation=90, va='center', fontsize = 16, color='olive')
plt.fill_between(a, b, color='plum')
plt.xlabel('CRMSE/ (ref)', fontsize=18)
plt.ylabel('MBE/ (ref)', fontsize=18)
plt.title('SO2', fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```

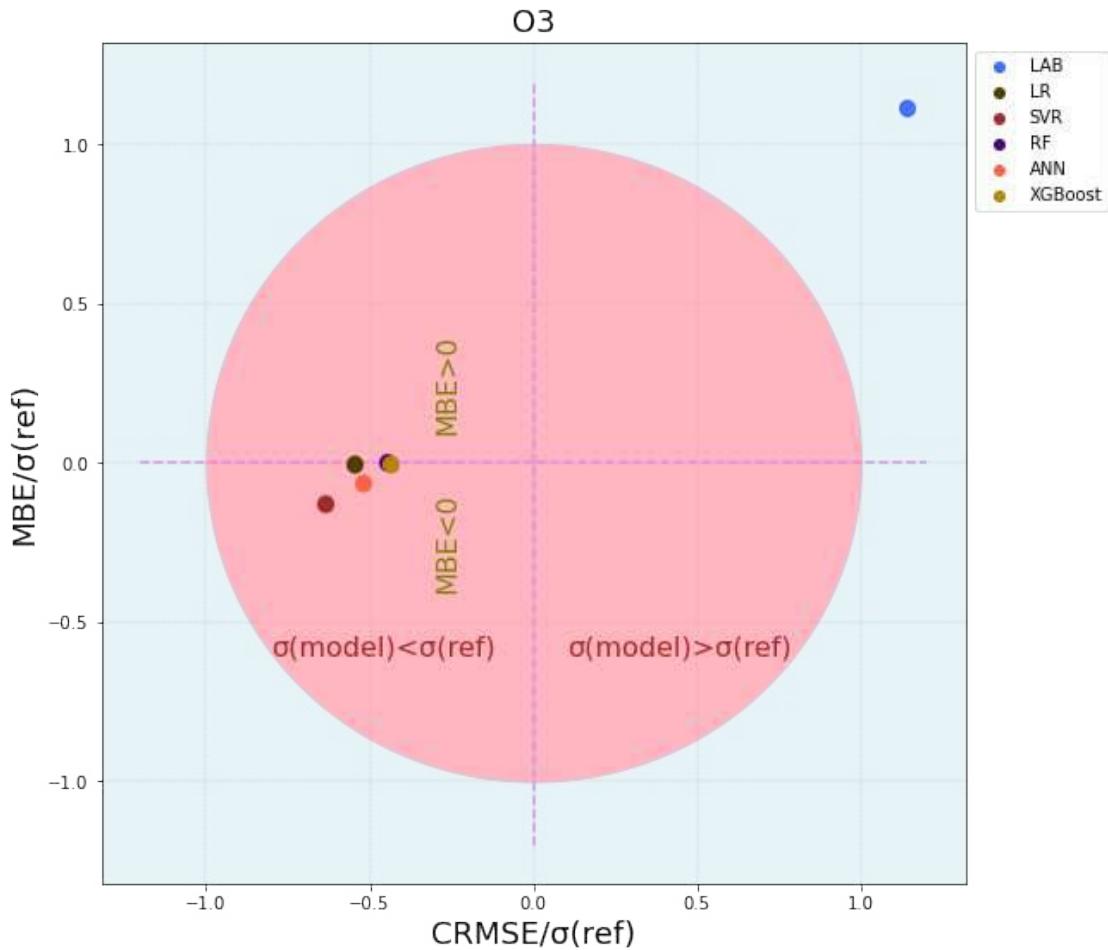


```
[125]: fig= plt.figure(figsize=(9,9))
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
plt.text(CRMSE_LAB[3],MBE_LAB[3] , '•', rotation=90, va='center', fontsize = 36,
         color="#426eff")
plt.text(CRMSE_LR[3],MBE_LR[3] , '•', rotation=90, va='center', fontsize = 36,
         color="#513e00")
plt.text(CRMSE_SVR[3],MBE_SVR[3] , '•', rotation=90, va='center', fontsize = 36,
         color='brown')
plt.text(CRMSE_RF[3],MBE_RF[3] , '•', rotation=90, va='center', fontsize = 36,
         color='indigo')
plt.text(CRMSE_ANN[3],MBE_ANN[3] , '•', rotation=90, va='center', fontsize = 36,
         color='tomato')
plt.text(CRMSE_XGB[3],MBE_XGB[3] , '•', rotation=90, va='center', fontsize = 36,
         color='darkgoldenrod')
plt.scatter(CRMSE_LAB[2]/1.1,MBE_LAB[2],color="#426eff")
```

```

plt.scatter(CRMSE_LR, MBE_LR, color="#513e00")
plt.scatter(CRMSE_SVR[0], MBE_SVR[0], color='brown')
plt.scatter(CRMSE_RF, MBE_RF, color='indigo')
plt.scatter(CRMSE_ANN, MBE_ANN, color='tomato')
plt.scatter(CRMSE_XGB, MBE_XGB, color='darkgoldenrod')
plt.legend(['LAB', 'LR', 'SVR', 'RF', 'ANN', 'XGBoost'], loc = 2, bbox_to_anchor =(1,1))
plt.Circle((0, 0), 1, color='wheat')
plt.vlines([0], -1.2, 1.2, linestyles='dashed',color='violet')
plt.hlines([0], -1.2, 1.2, linestyles='dashed', color='violet')
plt.text(-0.8, -0.6, ' (model)< (ref)', fontsize = 16, color='firebrick')
plt.text(0.1, -0.6, ' (model)> (ref)', fontsize = 16, color='firebrick')
plt.text(-0.3, -0.25, 'MBE<0', rotation=90, va='center',fontsize = 16,color='olive')
plt.text(-0.3, 0.25, 'MBE>0', rotation=90, va='center',fontsize = 16,color='olive')
plt.fill_between(a, b, color='lightpink')
plt.xlabel('CRMSE/ (ref)', fontsize=18)
plt.ylabel('MBE/ (ref)', fontsize=18)
plt.title('03', fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```

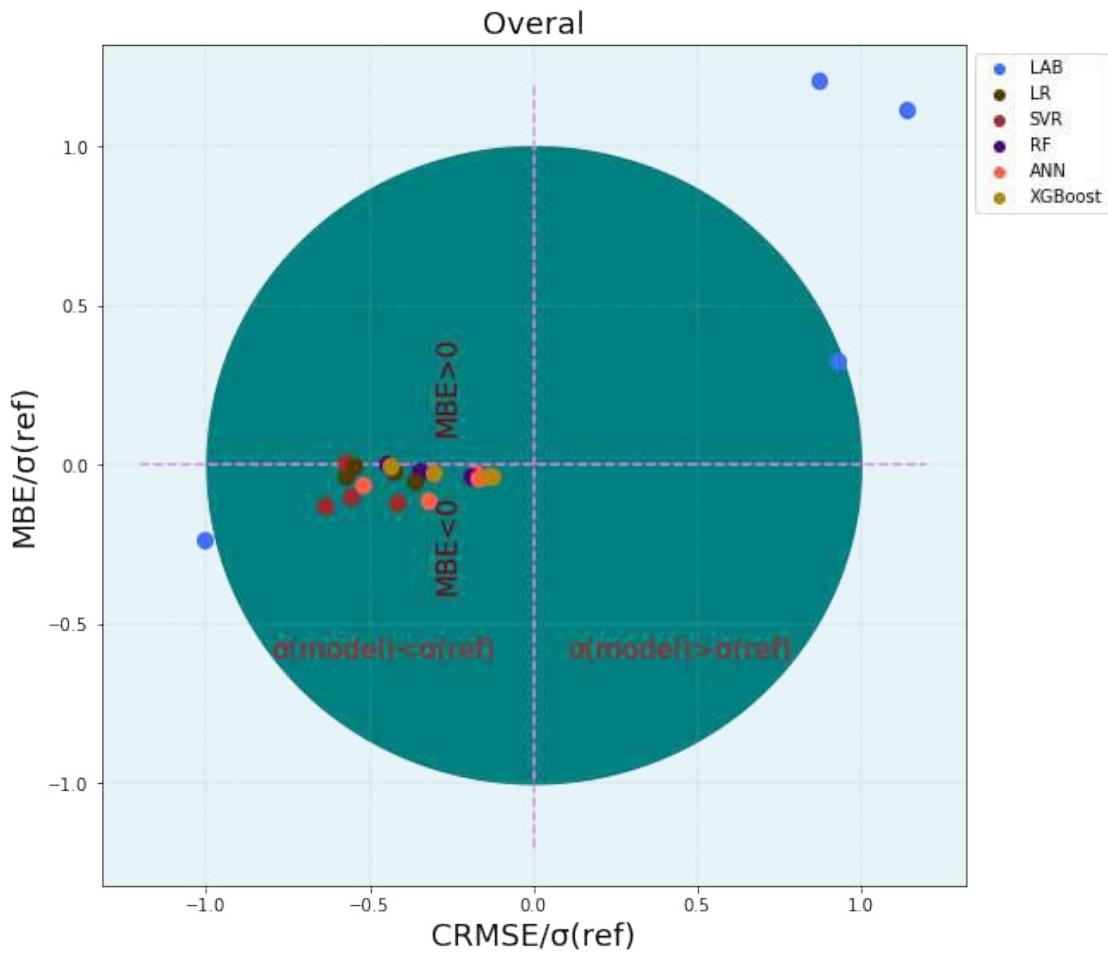


```
[126]: fig= plt.figure(figsize=(9,9))
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
for i in range(4):
    plt.text(CRMSE_LAB[i],MBE_LAB[i] , '•', rotation=90, va='center', fontsize = 36, color='#426eff')
    plt.text(CRMSE_LR[i],MBE_LR[i] , '•', rotation=90, va='center', fontsize = 36, color="#513e00")
    plt.text(CRMSE_SVR[i],MBE_SVR[i] , '•', rotation=90, va='center', fontsize = 36, color='brown')
    plt.text(CRMSE_RF[i],MBE_RF[i] , '•', rotation=90, va='center', fontsize = 36, color='indigo')
    plt.text(CRMSE_ANN[i],MBE_ANN[i] , '•', rotation=90, va='center', fontsize = 36, color='tomato')
    plt.text(CRMSE_XGB[i],MBE_XGB[i] , '•', rotation=90, va='center', fontsize = 36, color='darkgoldenrod')
```

```

plt.scatter(CRMSE_LAB[2]-0.2,MBE_LAB[2],color='#426eff')
plt.scatter(CRMSE_LR,MBE_LR,color='#513e00')
plt.scatter(CRMSE_SVR,MBE_SVR,color='brown')
plt.scatter(CRMSE_RF,MBE_RF,color='indigo')
plt.scatter(CRMSE_ANN,MBE_ANN,color='tomato')
plt.scatter(CRMSE_XGB,MBE_XGB,color='darkgoldenrod')
plt.legend(['LAB','LR','SVR','RF','ANN','XGBoost'],loc = 2, bbox_to_anchor = (1,1))
plt.vlines([0], -1.2, 1.2, linestyles='dashed',color='violet')
plt.hlines([0], -1.2, 1.2, linestyles='dashed', color='violet')
plt.text(-0.8, -0.6, ' (model)< (ref)', fontsize = 16, color='firebrick')
plt.text(0.1, -0.6, ' (model)> (ref)', fontsize = 16, color='firebrick')
plt.text(-0.3, -0.25, 'MBE<0', rotation=90, va='center',fontsize = 16,color='maroon')
plt.text(-0.3, 0.25, 'MBE>0', rotation=90, va='center',fontsize = 16,color='maroon')
plt.fill_between(a, b, color='teal')
plt.xlabel('CRMSE/ (ref)',fontsize=18)
plt.ylabel('MBE/ (ref)',fontsize=18)
plt.title('Overal',fontsize=18)
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



16.2 Feature Importance

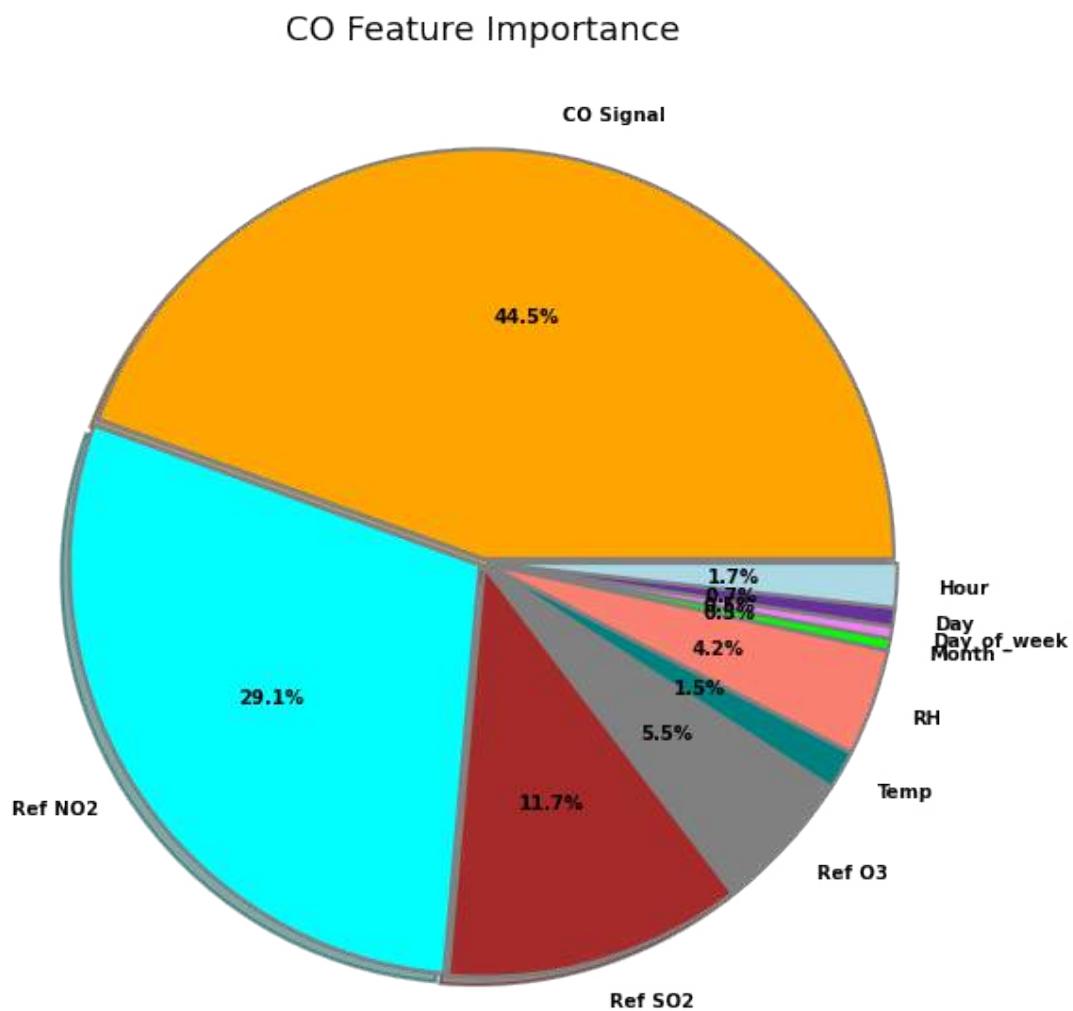
```
[127]: fig= plt.figure(figsize=(10,10))
# T-shirt sales for sizes Small, Medium, and Large
features = features_C0
# wedge labels
labels = 'CO_Signal', 'Ref_N02', 'Ref_S02', 'Ref_03', 'Temp',
         →'RH', 'Month', 'Day_of_week', 'Day', 'Hour'
# wedge colors
colors = [ "orange", "cyan", "brown",
           "grey", "teal", "salmon", "lime", "violet", "rebeccapurple", "lightblue"]
# explode the wedge for Small
explode = (0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01)
# line properties
wedge_props = {"linewidth": 2, "edgecolor": "grey"}
# text properties
```

```

text_props = {"weight": "bold"}

# plot a pie chart
plt.pie(features,
         labels=labels,
         autopct="%1.1f%%",
         colors=colors,
         shadow=True,
         explode=explode,
         wedgeprops=wedge_props,
         textprops=text_props)
plt.title("CO Feature Importance", fontsize=18)
plt.show()

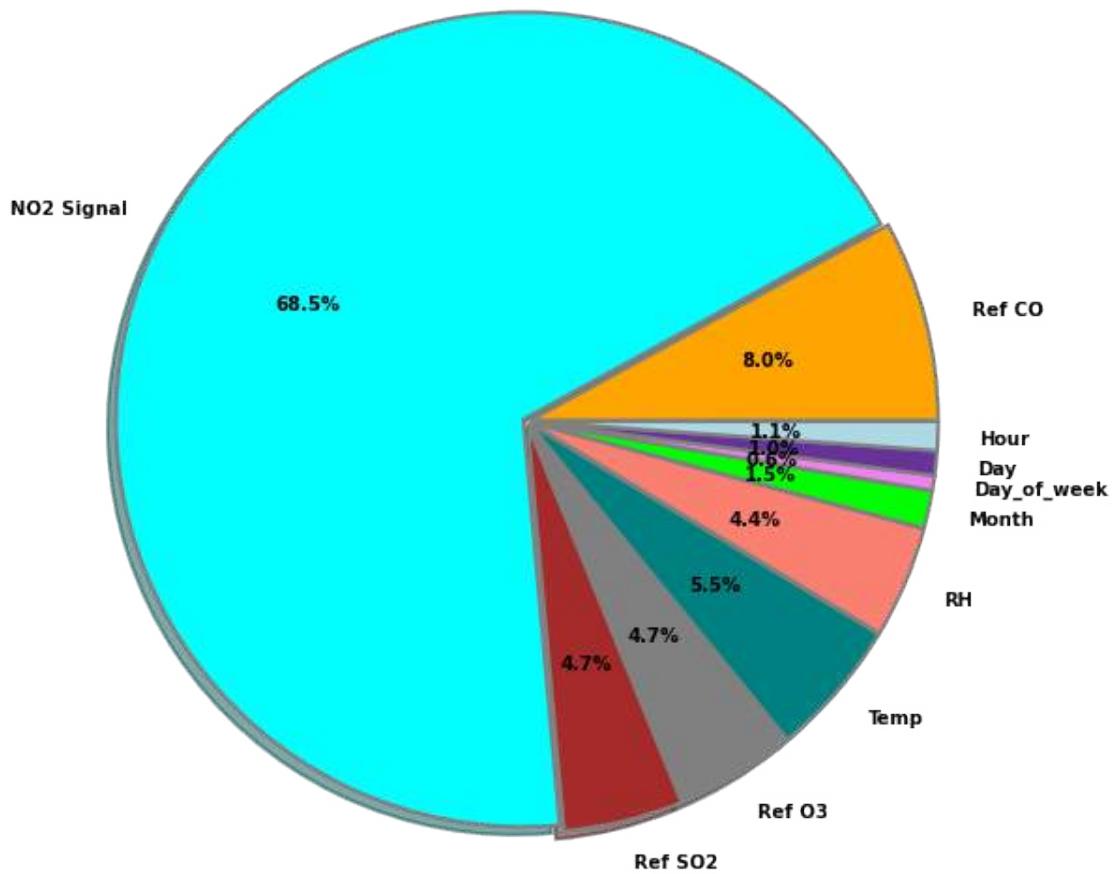
```



```
[128]: fig= plt.figure(figsize=(10,10))
# T-shirt sales for sizes Small, Medium, and Large
features =features_N02
# wedge labels
labels = ' Ref CO', 'N02 Signal', 'Ref S02', 'Ref O3', 'Temp',「
→'RH','Month','Day_of_week','Day','Hour'
# wedge colors
colors = [ "orange", "cyan", "brown",
           "grey","teal","salmon",'lime','violet','rebeccapurple','lightblue']
# explode the wedge for Small
explode = (0.01, 0.01, 0.01, 0.01,0.01,0.01, 0.01, 0.01,0.01,0.01)
# line properties
wedge_props = {"linewidth": 2, "edgecolor": "grey"}
# text properties
text_props = {"weight": "bold"}

# plot a pie chart
plt.pie(features,
         labels=labels,
         autopct="%1.1f%%",
         colors=colors,
         shadow=True,
         explode=explode,
         wedgeprops=wedge_props,
         textprops=text_props)
plt.title("N02 Feature Importance", fontsize=18)
plt.show()
```

NO2 Feature Importance



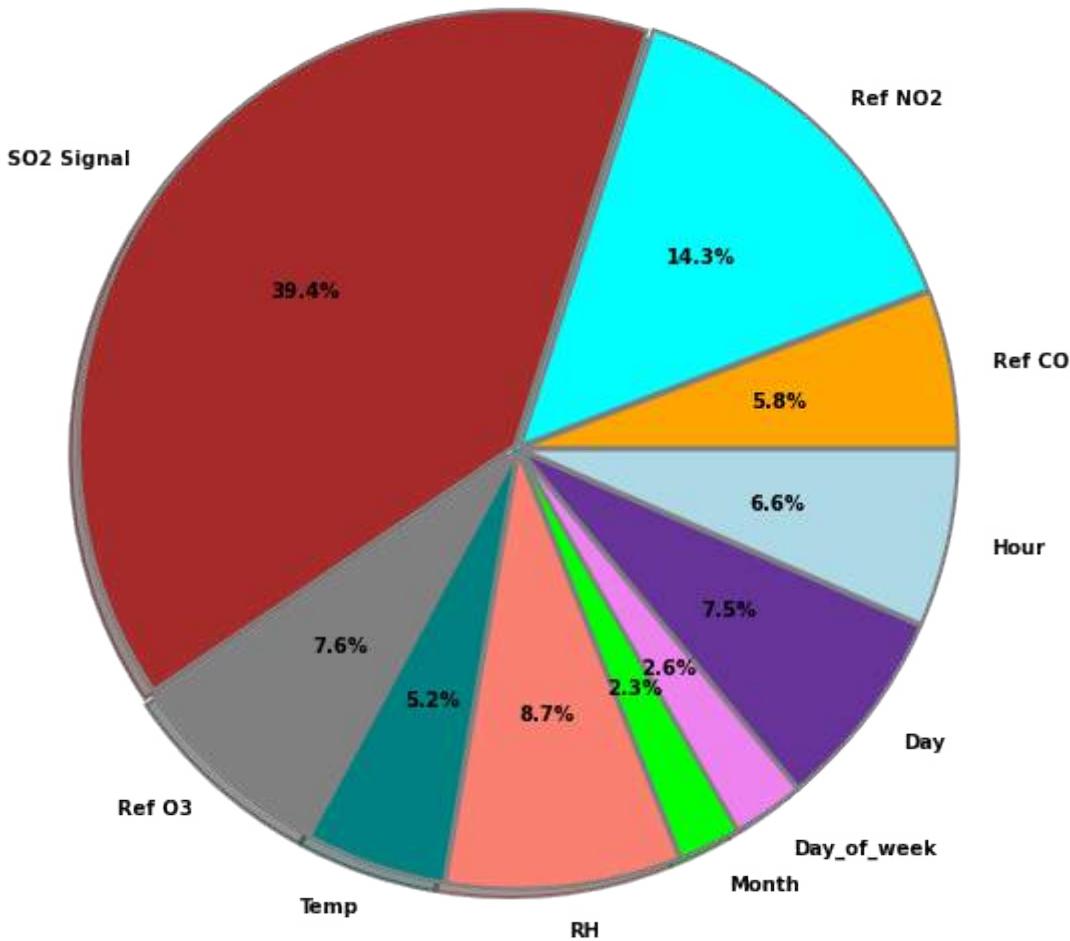
```
[129]: fig= plt.figure(figsize=(10,10))
# T-shirt sales for sizes Small, Medium, and Large
features =features_SO2
# wedge labels
labels = 'Ref CO', 'Ref NO2', 'SO2 Signal', 'Ref O3', 'Temp', □
    ↪'RH', 'Month', 'Day_of_week', 'Day', 'Hour'
# wedge colors
colors = [ "orange", "cyan", "brown",
           "grey", "teal", "salmon", "lime", "violet", "rebeccapurple", "lightblue"]
# explode the wedge for Small
explode = (0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01)
# line properties
wedge_props = {"linewidth": 2, "edgecolor": "grey"}
# text properties
```

```
text_props = {"weight": "bold"}  
  
# plot a pie chart  
plt.pie(features,  
         labels=labels,  
         autopct="%1.1f%%",  
         colors=colors,  
         shadow=True,  
         explode=explode,  
         wedgeprops=wedge_props,  
         textprops=text_props)  
plt.title("SO2 Feature Importance", fontsize=18)  
plt.show()
```

```
<ipython-input-129-47f9506edcaf>:17: MatplotlibDeprecationWarning:
```

```
normalize=None does not normalize if the sum is less than 1 but this behavior is  
deprecated since 3.3 until two minor releases later. After the deprecation  
period the default value will be normalize=True. To prevent normalization pass  
normalize=False
```

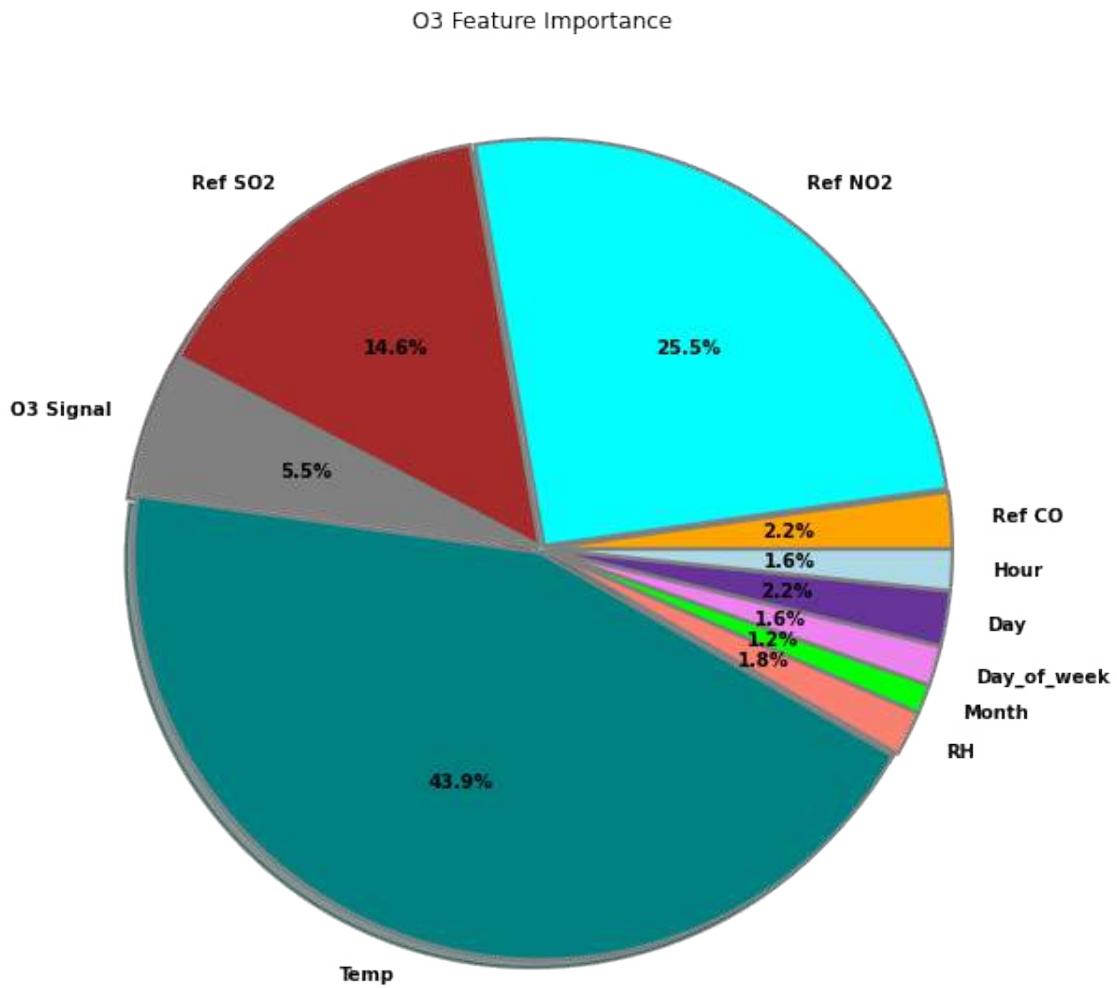
SO2 Feature Importance



```
[130]: fig= plt.figure(figsize=(10,10))
# T-shirt sales for sizes Small, Medium, and Large
features =features_03
# wedge labels
labels = 'Ref CO', 'Ref N02', 'Ref S02', '03 Signal', 'Temp', □
→'RH','Month','Day_of_week','Day','Hour'
# wedge colors
colors = [ "orange", "cyan", "brown",
          "grey","teal","salmon",'lime','violet','rebeccapurple','lightblue']
# explode the wedge for Small
explode = (0.01, 0.01, 0.01, 0.01,0.01,0.01, 0.01, 0.01,0.01,0.01)
# line properties
```

```
wedge_props = {"linewidth": 2, "edgecolor": "grey"}
# text properties
text_props = {"weight": "bold"}

# plot a pie chart
plt.pie(features,
         labels=labels,
         autopct="%1.1f%%",
         colors=colors,
         shadow=True,
         explode=explode,
         wedgeprops=wedge_props,
         textprops=text_props)
plt.title("O3 Feature Importance")
plt.show()
```



```
[131]: # libraries
import numpy as np
import matplotlib.pyplot as plt
fig= plt.figure(figsize=(10,5))
# set width of bars
ax = fig.add_subplot(111)
ax.patch.set_facecolor('lightblue')
ax.patch.set_alpha(0.3)
barWidth = 0.15

# set heights of bars
Signal1 = [features_C0[0],features_N02[0],features_S02[0],features_O3[0]]
Signal2 = [features_C0[1],features_N02[1],features_S02[1],features_O3[1]]
Signal3 = [features_C0[2],features_N02[2],features_S02[2],features_O3[2]]
Signal4 = [features_C0[3],features_N02[3],features_S02[3],features_O3[3]]
Temp = [features_C0[4],features_N02[4],features_S02[4],features_O3[4]]
RH = [features_C0[5],features_N02[5],features_S02[5],features_O3[5]]

# Set position of bar on X axis
r1 = np.arange(len(Signal1))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]
r4 = [x + barWidth for x in r3]
r5 = [x + barWidth for x in r4]
r6 = [x + barWidth for x in r5]

# Make the plot
plt.bar(r1, Signal1, color='magenta', width=barWidth, edgecolor='white', ▾
         ↳label='C0')
plt.bar(r2, Signal2, color='teal', width=barWidth, edgecolor='white', ▾
         ↳label='N02')
plt.bar(r3, Signal3, color='salmon', width=barWidth, edgecolor='white', ▾
         ↳label='S02')
plt.bar(r4, Signal4, color='rebeccapurple', width=barWidth, edgecolor='white', ▾
         ↳label='O3')
plt.bar(r5, Temp, color='olive', width=barWidth, edgecolor='white', ▾
         ↳label='Temperature')
plt.bar(r6, RH, color='darkgoldenrod', width=barWidth, edgecolor='white', ▾
         ↳label='RH')

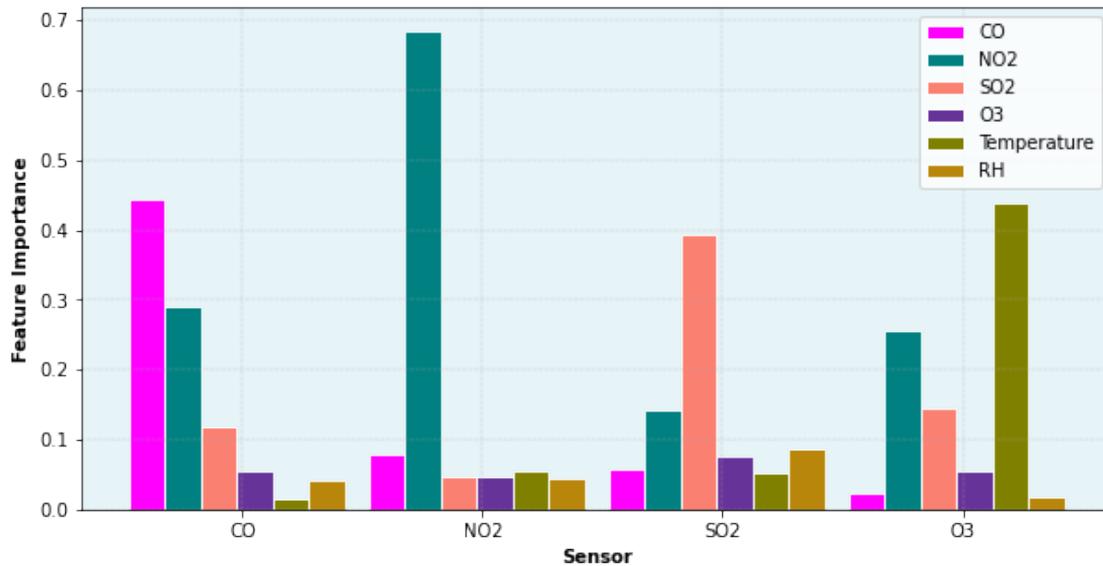
# Add xticks on the middle of the group bars
plt.xlabel('Sensor', fontweight='bold')
```

```

plt.ylabel('Feature Importance', fontweight='bold')
plt.xticks([r + barWidth+0.25 for r in range(len(Signal1))], ['CO', 'NO2', 'SO2', 'O3'])

# Create legend & Show graphic
plt.legend()
plt.grid(linestyle='-.', linewidth=0.3)
plt.show()

```



```

[132]: import plotly.graph_objects as go
import pandas as pd
model=['LAB','LR','SVR','RF','ANN','XGBoost']
Pearson=[Pearson_lab_CO,Pearson_lr_CO,Pearson_svr_CO,Pearson_rf_CO,Pearson_ann_CO,Pearson_xgb_CO]
R2=[R2_lab_CO,R2_lr_CO,R2_svr_CO,R2_rf_CO,R2_ann_CO,R2_xgb_CO]
RMSE=[RMSE_Lab_CO,RMSE_Lr_CO,RMSE_Svr_CO,RMSE_Rf_CO,RMSE_Ann_CO,RMSE_Xgb_CO]
sMAPE=[sMAPE_lab_CO,sMAPE_lr_CO,sMAPE_svr_CO,sMAPE_rf_CO,sMAPE_ann_CO,sMAPE_xgb_CO]

fig = go.Figure(data=[go.Table(
    header=dict(values=['Model', 'Pearson r', 'R^2', 'sMAPE', 'RMSE'],
                fill_color='paleturquoise',
                align='left'),
    cells=dict(values=[model,Pearson,R2,sMAPE, RMSE ],
               fill_color='lavender',
               align='left'))])
fig.update_layout(
    title={'text': "CO calibration : Model Performance",

```

```

'y':0.86,
'x':0.5,
'xanchor': 'center',
'yanchor': 'top'},
width=700,
height=600,
)
#fig.show()
chart_studio.plotly.sign_in('vinylango', 'gybbJWWfRSUoTcRRSa6J')
chart_studio.plotly.image.save_as(fig, filename='models_performance_C0.png')
Image('models_performance_C0.png')

```

[132] :

CO calibration : Model Performance

Model	Pearson r	R^2	sMAPE	RMSE
LAB	-0.01	-0.25	0.67	807.6
LR	0.89	0.79	0.37	331
SVR	0.89	0.74	0.27	371.9
RF	0.96	0.92	0.16	198
ANN	0.96	0.93	0.2	196.9
XGBoost	0.98	0.95	0.15	158.7

[133] : import plotly.graph_objects as go

```

import pandas as pd
model=['LAB','LR','SVR','RF','ANN','XGBoost']
Pearson=[Pearson_lab_N02,Pearson_lr_N02,Pearson_svr_N02,Pearson_rf_N02,Pearson_ann_N02,Pearson_xgb_N02]
R2=[R2_lab_N02,R2_lr_N02,R2_svr_N02,R2_rf_N02,R2_ann_N02,R2_xgb_N02]

```

```

RMSE=[RMSE_Lab_N02,RMSE_Lr_N02,RMSE_Svr_N02,RMSE_Rf_N02,RMSE_Ann_N02,RMSE_Xgb_N02]
sMAPE=[sMAPE_lab_N02,sMAPE_lr_N02,sMAPE_svr_N02,sMAPE_rf_N02,sMAPE_ann_N02,sMAPE_xgb_N02]

fig = go.Figure(data=[go.Table(
    header=dict(values=['Model', 'Pearson r', 'R^2', 'sMAPE', 'RMSE'],
                fill_color='paleturquoise',
                align='left'),
    cells=dict(values=[model,Pearson,R2,sMAPE, RMSE ],
               fill_color='lavender',
               align='left'))
])
fig.update_layout(
title={'text': "N02 calibration : Model Performance",
'y':0.86,
'x':0.5,
'xanchor': 'center',
'yanchor': 'top'},
width=700,
height=600,
)
chart_studio.plotly.sign_in('vinylango', 'gybbJWWfRSUoTcRRSa6J')
chart_studio.plotly.image.save_as(fig, filename='models_performance_N02.png')
Image('models_performance_N02.png')

```

[133]:

NO2 calibration : Model Performance

Model	Pearson r	R^2	sMAPE	RMSE
LAB	0.07	-4.6	0.87	43
LR	0.86	0.73	0.32	9.5
SVR	0.76	0.57	0.42	11.9
RF	0.96	0.92	0.16	5.2
ANN	0.96	0.93	0.17	4.8
XGBoost	0.97	0.95	0.14	4.2

```
[134]: import plotly.graph_objects as go
import pandas as pd
model=['LAB','LR','SVR','RF','ANN','XGBoost']
Pearson=[Pearson_lab_S02,Pearson_lr_S02,Pearson_svr_S02,Pearson_rf_S02,Pearson_ann_S02,Pearson_xgb_S02]
R2=[R2_lab_S02,R2_lr_S02,R2_svr_S02,R2_rf_S02,R2_ann_S02,R2_xgb_S02,]
RMSE=[RMSE_Lab_S02,RMSE_Lr_S02,RMSE_Svr_S02,RMSE_Rf_S02,RMSE_Ann_S02,RMSE_Xgb_S02]
sMAPE=[sMAPE_lab_S02,sMAPE_lr_S02,sMAPE_svr_S02,sMAPE_rf_S02,sMAPE_ann_S02,sMAPE_xgb_S02]

fig = go.Figure(data=[go.Table(
    header=dict(values=['Model','Pearson r','R^2','sMAPE','RMSE'],
                fill_color='paleturquoise',
                align='left'),
    cells=dict(values=[model,Pearson,R2,sMAPE, RMSE ],
               fill_color='lavender',
               align='left'))])
fig.update_layout(
```

```

title={'text': "SO2 calibration : Model Performance",
       'y':0.86,
       'x':0.5,
       'xanchor': 'center',
       'yanchor': 'top'},
width=700,
height=600,
)
chart_studio.plotly.sign_in('vinylango', 'gybbJVWfRSUoTcRRSa6J')
chart_studio.plotly.image.save_as(fig, filename='models_performance_SO2.png')
Image('models_performance_SO2.png')

```

[134]:

SO2 calibration : Model Performance				
Model	Pearson r	R^2	sMAPE	RMSE
LAB	0	-1682.99	1.49	38.8
LR	0.77	0.59	0.28	0.6
SVR	0.69	0.46	0.3	0.7
RF	0.84	0.7	0.22	0.5
ANN	0.8	0.62	0.25	0.6
XGBoost	0.85	0.71	0.22	0.5

[135]: import plotly.graph_objects as go

```

import pandas as pd
model=['LAB','LR','SVR','RF','ANN','XGBoost']
Pearson=[str(Pearson_lab_03),Pearson_lr_03,Pearson_svr_03,Pearson_rf_03,Pearson_ann_03,Pearson_xgb_03]

```

```

R2=[R2_lab_03,R2_lr_03,R2_svr_03,R2_rf_03,R2_ann_03,R2_xgb_03]
RMSE=[RMSE_Lab_03,RMSE_Lr_03,RMSE_Svr_03,RMSE_Rf_03,RMSE_Ann_03,RMSE_Xgb_03]
sMAPE=[sMAPE_lab_03,sMAPE_lr_03,sMAPE_svr_03,sMAPE_rf_03,sMAPE_ann_03,sMAPE_xgb_03]

fig = go.Figure(data=[go.Table(
    header=dict(values=['Model', 'Pearson r', 'R^2', 'sMAPE', 'RMSE'],
                fill_color='paleturquoise',
                align='left'),
    cells=dict(values=[model,Pearson,R2,sMAPE, RMSE ],
               fill_color='lavender',
               align='left'))
])
fig.update_layout(
    title={'text': "03 calibration : Model Performance",
           'y':0.86,
           'x':0.5,
           'xanchor': 'center',
           'yanchor': 'top'},
    width=700,
    height=600,
)
chart_studio.plotly.sign_in('vinylango', 'gybbJWWfRSUoTcRRSa6J')
chart_studio.plotly.image.save_as(fig, filename='models_performance_03.png')
Image('models_performance_03.png')

```

[135]:

O3 calibration : Model Performance

Model	Pearson r	R^2	sMAPE	RMSE
LAB	-0.08	-26.76	1.08	145.9
LR	0.74	0.55	0.39	18.6
SVR	0.75	0.55	0.39	18.5
RF	0.9	0.81	0.22	12.2
ANN	0.91	0.82	0.23	11.7
XGBoost	0.92	0.84	0.22	11.1

```
[136]: Pearson_lab=np.  
        →array([Pearson_lab_C0,Pearson_lab_N02,Pearson_lab_S02,Pearson_lab_03])  
Pearson_lr=np.array([Pearson_lr_C0,Pearson_lr_N02,Pearson_lr_S02,Pearson_lr_03])  
Pearson_svr=np.  
        →array([Pearson_svr_C0,Pearson_svr_N02,Pearson_svr_S02,Pearson_svr_03])  
Pearson_rf=np.array([Pearson_rf_C0,Pearson_rf_N02,Pearson_rf_S02,Pearson_rf_03])  
Pearson_ann=np.  
        →array([Pearson_ann_C0,Pearson_ann_N02,Pearson_ann_S02,Pearson_ann_03])  
Pearson_xgb=np.  
        →array([Pearson_xgb_C0,Pearson_xgb_N02,Pearson_xgb_S02,Pearson_xgb_03])  
R2_lab=np.array([R2_lab_C0,R2_lab_N02,R2_lab_S02,R2_lab_03])  
R2_lr=np.array([R2_lr_C0,R2_lr_N02,R2_lr_S02,R2_lr_03])  
R2_svr=np.array([R2_svr_C0,R2_svr_N02,R2_svr_S02,R2_svr_03])  
R2_rf=np.array([R2_rf_C0,R2_rf_N02,R2_rf_S02,R2_rf_03])  
R2_ann=np.array([R2_ann_C0,R2_ann_N02,R2_ann_S02,R2_ann_03])  
R2_xgb=np.array([R2_xgb_C0,R2_xgb_N02,R2_xgb_S02,R2_xgb_03])  
RMSE_lab=np.array([RMSE_Lab_C0,RMSE_Lab_N02,RMSE_Lab_S02,RMSE_Lab_03])
```

```

RMSE_lr=np.array([RMSE_Lr_C0,RMSE_Lr_N02,RMSE_Lr_S02,RMSE_Lr_O3])
RMSE_svr=np.array([RMSE_Svr_C0,RMSE_Svr_N02,RMSE_Svr_S02,RMSE_Svr_O3])
RMSE_rf=np.array([RMSE_Rf_C0,RMSE_Rf_N02,RMSE_Rf_S02,RMSE_Rf_O3])
RMSE_ann=np.array([RMSE_Ann_C0,RMSE_Ann_N02,RMSE_Ann_S02,RMSE_Ann_O3])
RMSE_xgb=np.array([RMSE_Xgb_C0,RMSE_Xgb_N02,RMSE_Xgb_S02,RMSE_Xgb_O3])
sMAPE_lab=np.array([sMAPE_lab_C0,sMAPE_lab_N02,sMAPE_lab_S02,sMAPE_lab_O3])
sMAPE_lr=np.array([sMAPE_lr_C0,sMAPE_lr_N02,sMAPE_lr_S02,sMAPE_lr_O3])
sMAPE_svr=np.array([sMAPE_svr_C0,sMAPE_svr_N02,sMAPE_svr_S02,sMAPE_svr_O3])
sMAPE_rf=np.array([sMAPE_rf_C0,sMAPE_rf_N02,sMAPE_rf_S02,sMAPE_rf_O3])
sMAPE_ann=np.array([sMAPE_ann_C0,sMAPE_ann_N02,sMAPE_ann_S02,sMAPE_ann_O3])
sMAPE_xgb=np.array([sMAPE_xgb_C0,sMAPE_xgb_N02,sMAPE_xgb_S02,sMAPE_xgb_O3])
Pearson_lab_mean=round(np.mean(Pearson_lab),2)
Pearson_lab_std=round(np.std(Pearson_lab),2)
Pearson_lr_mean=round(np.mean(Pearson_lr),2)
Pearson_lr_std=round(np.std(Pearson_lr),2)
Pearson_svr_mean=round(np.mean(Pearson_svr),2)
Pearson_svr_std=round(np.std(Pearson_svr),2)
Pearson_rf_mean=round(np.mean(Pearson_rf),2)
Pearson_rf_std=round(np.std(Pearson_rf),2)
Pearson_ann_mean=round(np.mean(Pearson_ann),2)
Pearson_ann_std=round(np.std(Pearson_ann),2)
Pearson_xgb_mean=round(np.mean(Pearson_xgb),2)
Pearson_xgb_std=round(np.std(Pearson_xgb),2)
R2_lab_mean=round(np.mean(R2_lab),2)
R2_lab_std=round(np.std(R2_lab),2)
R2_lr_mean=round(np.mean(R2_lr),2)
R2_lr_std=round(np.std(R2_lr),2)
R2_svr_mean=round(np.mean(R2_svr),2)
R2_svr_std=round(np.std(R2_svr),2)
R2_rf_mean=round(np.mean(R2_rf),2)
R2_rf_std=round(np.std(R2_rf),2)
R2_ann_mean=round(np.mean(R2_ann),2)
R2_ann_std=round(np.std(R2_ann),2)
R2_xgb_mean=round(np.mean(R2_xgb),2)
R2_xgb_std=round(np.std(R2_xgb),2)
RMSE_lab_mean=round(np.mean(RMSE_lab),2)
RMSE_lab_std=round(np.std(RMSE_lab),2)
RMSE_lr_mean=round(np.mean(RMSE_lr),2)
RMSE_lr_std=round(np.std(RMSE_lr),2)
RMSE_svr_mean=round(np.mean(RMSE_svr),2)
RMSE_svr_std=round(np.std(RMSE_svr),2)
RMSE_rf_mean=round(np.mean(RMSE_rf),2)
RMSE_rf_std=round(np.std(RMSE_rf),2)
RMSE_ann_mean=round(np.mean(RMSE_ann),2)
RMSE_ann_std=round(np.std(RMSE_ann),2)
RMSE_xgb_mean=round(np.mean(RMSE_xgb),2)
RMSE_xgb_std=round(np.std(RMSE_xgb),2)

```

```
sMAPE_lab_mean=round(np.mean(sMAPE_lab),2)
sMAPE_lab_std=round(np.std(sMAPE_lab),2)
sMAPE_lr_mean=round(np.mean(sMAPE_lr),2)
sMAPE_lr_std=round(np.std(sMAPE_lr),2)
sMAPE_svr_mean=round(np.mean(sMAPE_svr),2)
sMAPE_svr_std=round(np.std(sMAPE_svr),2)
sMAPE_rf_mean=round(np.mean(sMAPE_rf),2)
sMAPE_rf_std=round(np.std(sMAPE_rf),2)
sMAPE_ann_mean=round(np.mean(sMAPE_ann),2)
sMAPE_ann_std=round(np.std(sMAPE_ann),2)
sMAPE_xgb_mean=round(np.mean(sMAPE_xgb),2)
sMAPE_xgb_std=round(np.std(sMAPE_xgb),2)
```

```
[137]: import plotly.graph_objects as go
import pandas as pd
model=['LAB','LR','SVR','RF','ANN','XGBoost']
Pearson=[str(Pearson_lab_mean)+ '±' +str(Pearson_lab_std),str(Pearson_lr_mean)+
         →'±' +str(Pearson_lr_std),
         str(Pearson_svr_mean)+ '±' +str(Pearson_svr_std),str(Pearson_rf_mean)+
         →'±' +str(Pearson_rf_std),
         str(Pearson_ann_mean)+ '±' +
         →+str(Pearson_ann_std),str(Pearson_xgb_mean)+ '±' +str(Pearson_xgb_std)]
```



```
R2=[str(R2_lab_mean)+ '±' +str(R2_lab_std),str(R2_lr_mean)+ '±' +str(R2_lr_std),
    str(R2_svr_mean)+ '±' +str(R2_svr_std),str(R2_rf_mean)+ '±' +str(R2_rf_std),
    str(R2_ann_mean)+ '±' +str(R2_ann_std),str(R2_xgb_mean)+ '±' +
    →+str(R2_xgb_std)]
```



```
RMSE=[str(RMSE_lab_mean)+ '±' +str(RMSE_lab_std),str(RMSE_lr_mean)+ '±' +
      →+str(RMSE_lr_std),
      str(RMSE_svr_mean)+ '±' +str(RMSE_svr_std),str(RMSE_rf_mean)+ '±' +
      →+str(RMSE_rf_std),
      str(RMSE_ann_mean)+ '±' +str(RMSE_ann_std),str(RMSE_xgb_mean)+ '±' +
      →+str(RMSE_xgb_std)]
```



```
sMAPE=[str(sMAPE_lab_mean)+ '±' +str(sMAPE_lab_std),str(sMAPE_lr_mean)+ '±' +
       →+str(sMAPE_lr_std),
       str(sMAPE_svr_mean)+ '±' +str(sMAPE_svr_std),str(sMAPE_rf_mean)+ '±' +
       →+str(sMAPE_rf_std),
       str(sMAPE_ann_mean)+ '±' +str(sMAPE_ann_std),str(sMAPE_xgb_mean)+ '±' +
       →+str(sMAPE_xgb_std)]
```

```

fig = go.Figure(data=[go.Table(
    header=dict(values=['Model', 'Pearson r ± ', 'R^2 ± ', 'sMAPE ± ', 'RMSE ±'],
                fill_color='paleturquoise',
                align='left'),
    cells=dict(values=[model,Pearson,R2,sMAPE,RMSE ],
               fill_color='lavender',
               align='left'))
])
fig.update_layout(
    title={'text': "03 calibration : Average Model Performance",
           'y':0.93,
           'x':0.5,
           'xanchor': 'center',
           'yanchor': 'top'},
    width=1000,
    height=1100,
)
chart_studio.plotly.sign_in('vinylango', 'gybbJVWfRSUoTcRRSa6J')
chart_studio.plotly.image.save_as(fig, filename='models_performance_03.png')
Image('models_performance_03.png')

```

[137]:

O3 calibration : Average Model Performance

Model	Pearson r ± σ	R² ± σ	sMAPE ± σ	RMSE ± σ
LAB	-0.0±0.05	-428.65±724.26	1.03±0.3	258.82±319.73
LR	0.82±0.06	0.66±0.1	0.34±0.04	89.93±139.33
SVR	0.77±0.07	0.58±0.1	0.34±0.06	100.75±156.68
RF	0.91±0.05	0.84±0.09	0.19±0.03	53.97±83.26
ANN	0.91±0.07	0.82±0.13	0.21±0.03	53.5±82.89
XGBoost	0.93±0.05	0.86±0.1	0.18±0.04	43.62±66.55

[]:

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':
```

```

        S0_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 S0_2[i]=='nan':
        S0_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(S0_2)):
    if S0_2[i]== 'BDL':
        S0_2[i]='nan'
for i in range(len(O_3)):
    if O_3[i]== 'BDL':
        O_3[i]='nan'
for i in range(len(S0_2)):
    if (S0_2[i]== 'nan'):
        S0_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],S0_2[i],O_3[i], CO[i]] for i in
         range(len(date_NT))]

data_NT= pd.DataFrame(data_NT, columns = ['Date','NO_2','S0_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["S0_2"] = pd.to_numeric(data_NT["S0_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['S0_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.set_index('Date')
data_NT_0=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]:

	NO_2	SO_2	O_3	CO	PM_2.5	PM_10
Date						
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	
mean	30.552916	2.724416	56.590080	503.460266	17.622353	
std	12.430579	1.524104	23.430660	236.155304	6.828044	
min	4.437500	0.111667	11.179167	161.383179	3.000000	
25%	20.973541	1.520833	35.293701	337.561981	13.000000	
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	

	PM_10
count	1067.000000
mean	43.386223
std	28.948460
min	10.600000
25%	30.000000

```

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+C0', 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       C0
```

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)]
C0=[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(C0)):
    if (C0[i]=='BDL'):
        C0[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.set_index('Date')
data_NR_0=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]:

	NO_2	SO_2	O_3	CO	PM_2.5	PM_10
Date						
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	
mean	21.873568	1.978501	68.560783	308.482452	14.992857	
std	11.902061	1.115598	23.170956	160.722107	6.087932	
min	4.245833	0.000000	15.056521	17.467501	1.700000	
25%	12.418188	1.220833	49.006149	198.868439	11.100000	
50%	18.887501	1.803333	69.943756	263.924561	14.200000	
75%	29.322292	2.504167	87.273094	377.548126	17.775000	
max	65.085716	7.175000	129.175003	1169.722534	78.100000	
						PM_10
count	1032.000000					
mean	35.353973					
std	28.894236					
min	9.600000					
25%	24.800000					
50%	29.900000					
75%	36.900000					
max	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.set_index('Date')
data_Z_0=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]:

	NO_2	SO_2	O_3	CO	PM_2.5	PM_10
Date						
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	

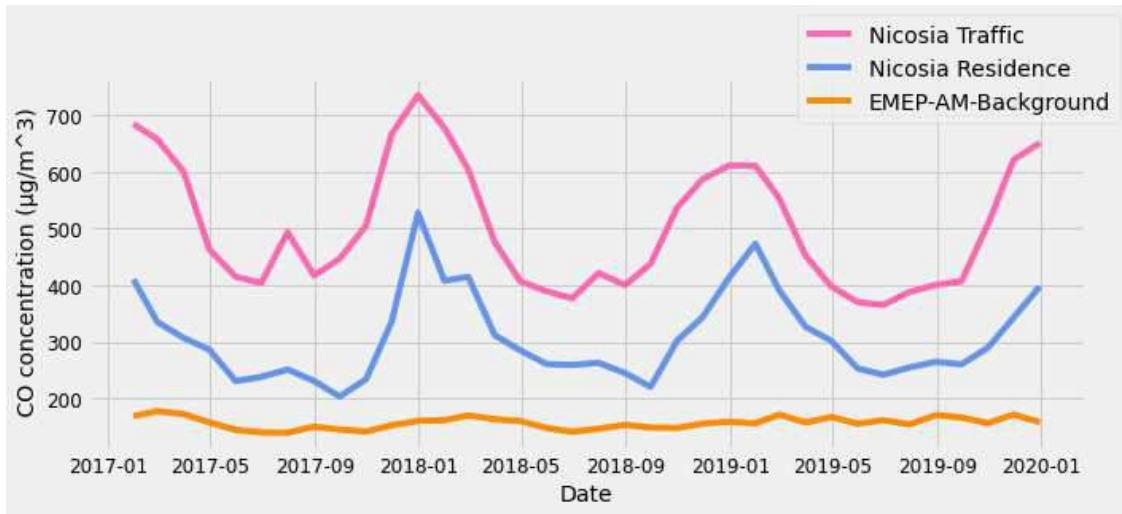
min	2.715000	0.000000	60.569565	82.362503	0.500000
25%	6.381042	0.563333	85.797916	138.268333	6.875000
50%	7.892917	0.966667	95.721115	153.547058	9.900000
75%	9.839693	1.435000	106.540264	170.518753	13.400000
max	26.083334	9.247827	144.033340	346.175018	71.000000

	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

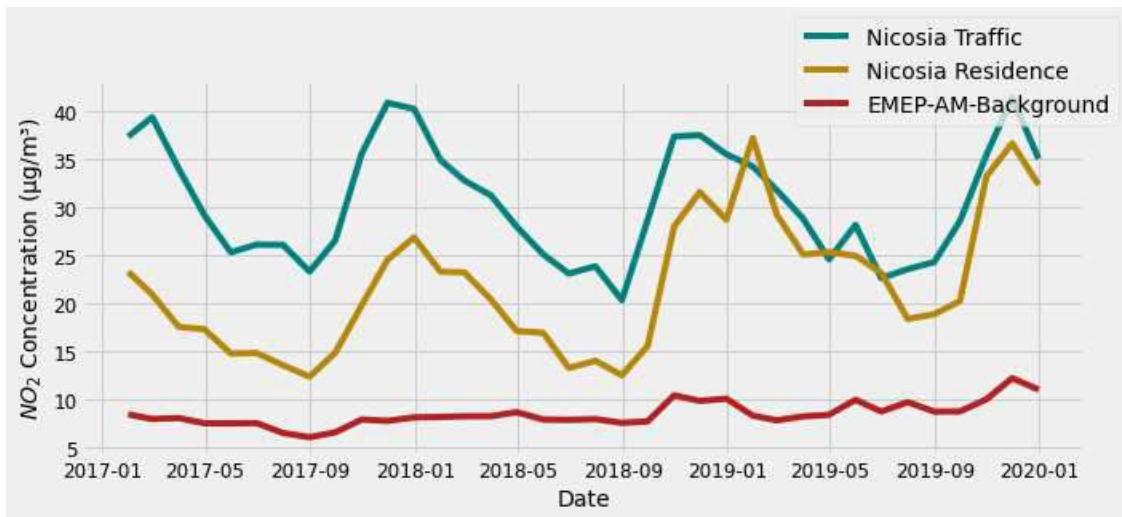
```
[9]: import matplotlib.pyplot as plt
import seaborn as sns
data_NT_by_month=data_NT.resample('M').mean()
data_NR_by_month=data_NR.resample('M').mean()
data_AM_by_month=data_AM.resample('M').mean()
fig= plt.figure(figsize=(10,4))
#colors = 
    →['lightpink', 'pink', 'fuchsia', 'mistyrose', 'hotpink', '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)')

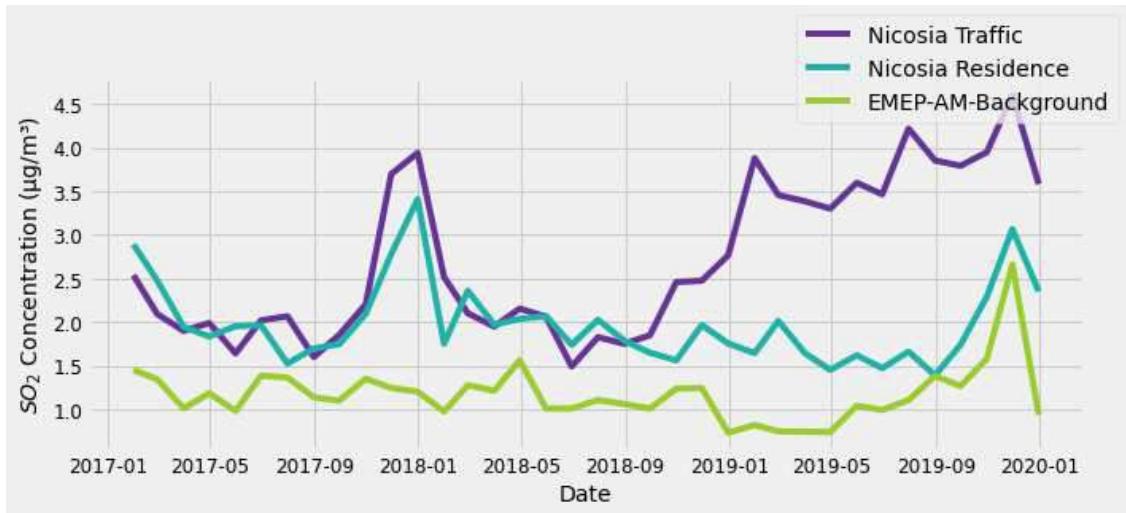


```
[10]: fig= plt.figure(figsize=(10,4))
#colors = 
    →['lightpink', 'pink', 'fuchsia', 'mistyrose', 'hotpink', 'deeppink', 'magenta']
sns.lineplot(x=data_NT_by_month.
    →index,y='NO_2',data=data_NT_by_month,color='teal')
sns.lineplot(x=data_NR_by_month.
    →index,y='NO_2',data=data_NR_by_month,color='darkgoldenrod')
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 ($\mu g/m^3$)')
```

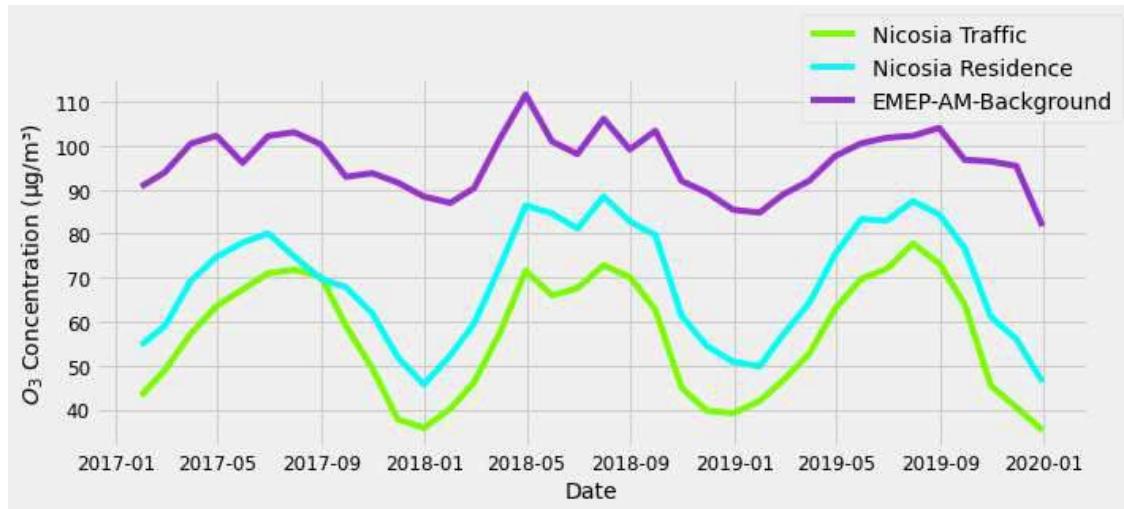
[10]: Text(0, 0.5, '\$NO_{2}\$ Concentration (\$\mu g/m^3\$)')



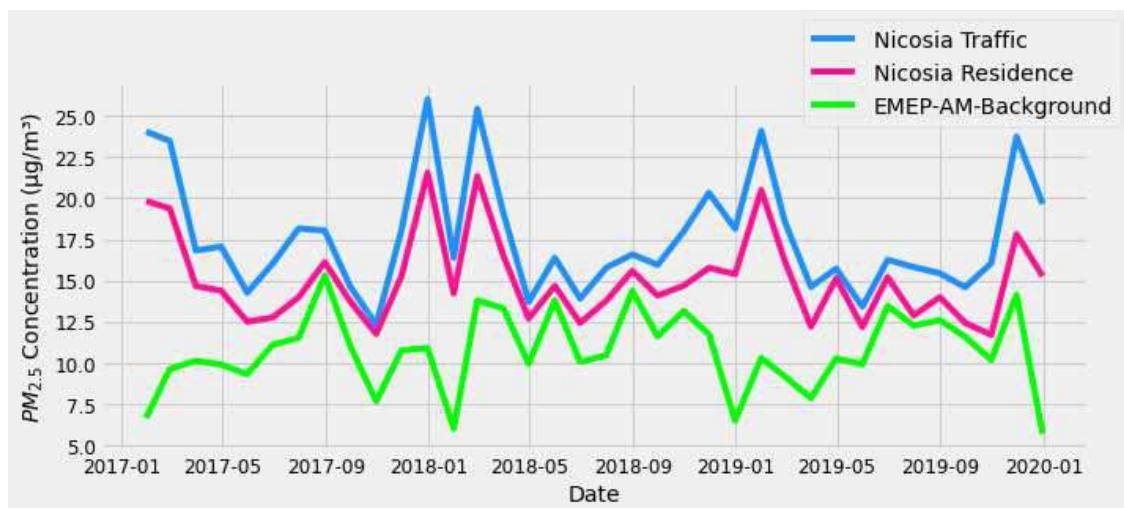
```
[11]: fig= plt.figure(figsize=(10,4))
#colors = 
    ['lightpink', 'pink', 'fuchsia', 'mistyrose', 'hotpink', 'deeppink', 'magenta']
sns.lineplot(x=data_NT_by_month.
    index,y='SO_2',data=data_NT_by_month,color='rebeccapurple')
sns.lineplot(x=data_NR_by_month.
    index,y='SO_2',data=data_NR_by_month,color='lightseagreen')
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 ( $\mu\text{g}/\text{m}^3$ )')
plt.show()
```



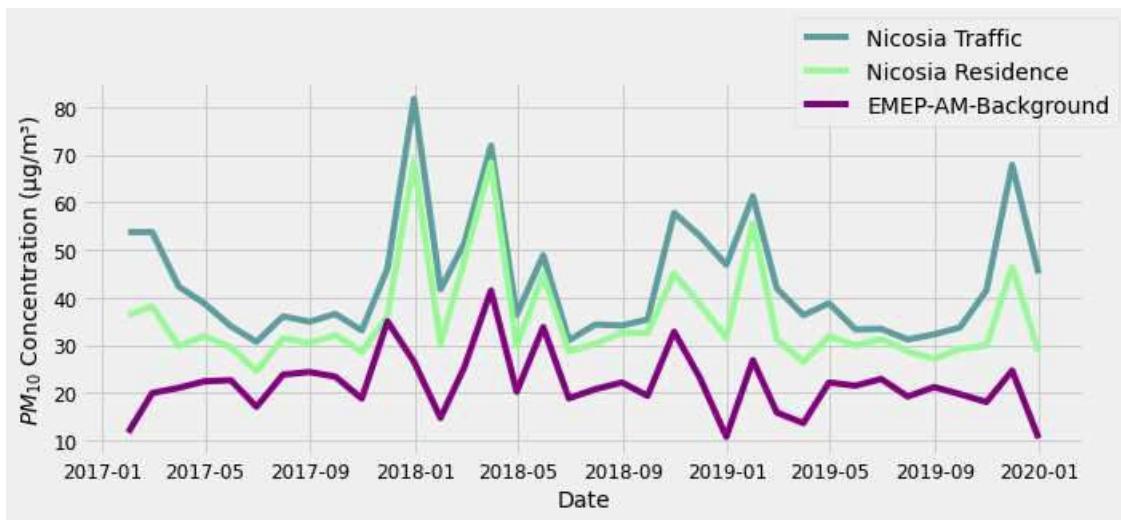
```
[12]: fig= plt.figure(figsize=(10,4))
#colors = 
    ['lightpink', 'pink', 'fuchsia', 'mistyrose', 'hotpink', 'deeppink', 'magenta']
sns.lineplot(x=data_NT_by_month.
    index,y='O_3',data=data_NT_by_month,color='lawngreen')
sns.lineplot(x=data_NR_by_month.
    index,y='O_3',data=data_NR_by_month,color='cyan')
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 ( $\mu\text{g}/\text{m}^3$ )')
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', 'deppink', '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 (pg/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_0['Hour']=data_NT_0.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_0['Hour']=data_NR_0.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_0['Hour']=data_Z_0.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_0, data_NR_0, data_Z_0]
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()

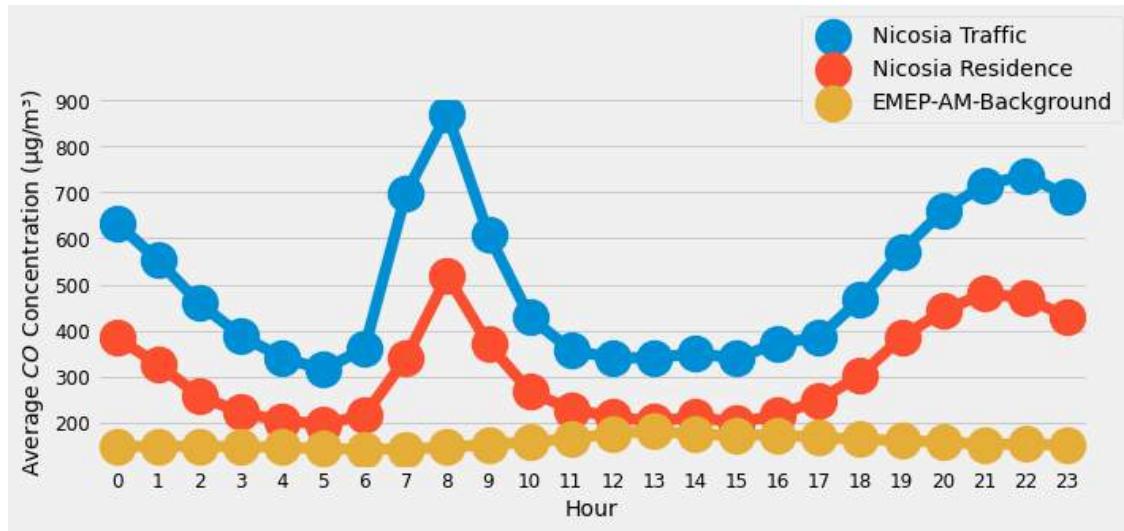
```

1.4 Hourly Visualization

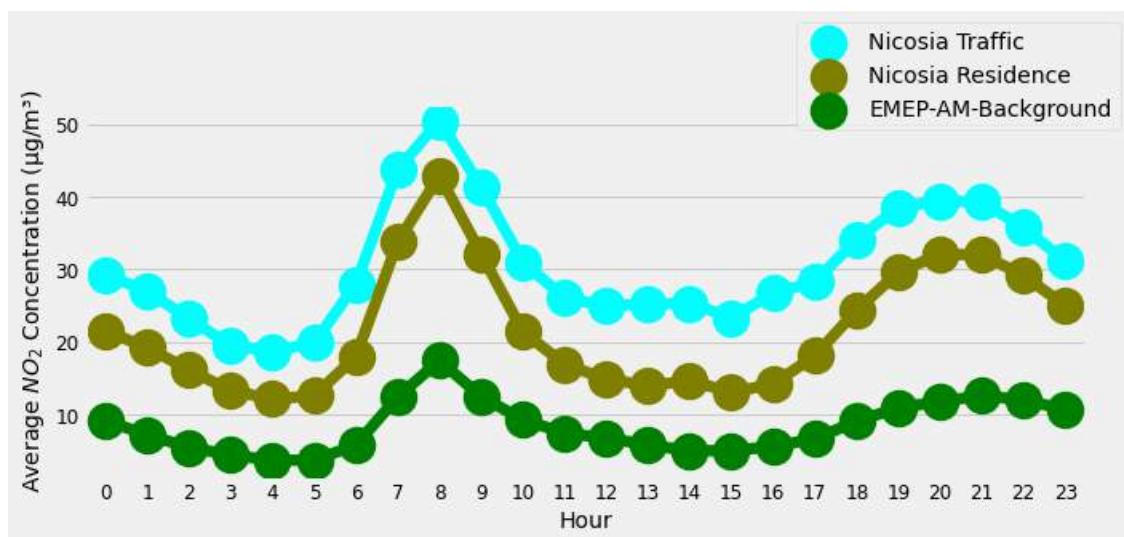
```

[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'$CO_{\mu g/m^3}$' + ' Concentration ($\mu g/m^3$)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()

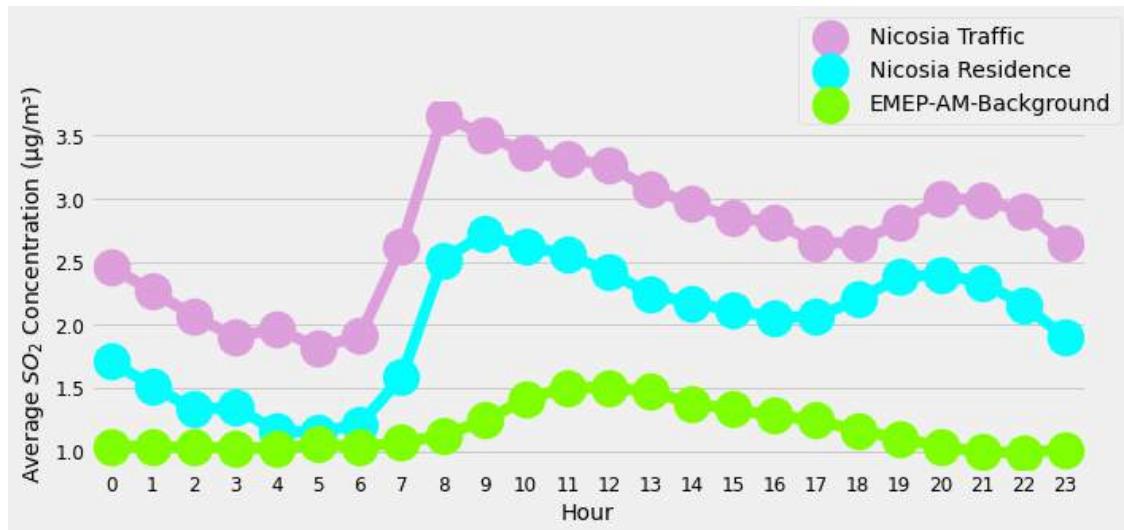
```



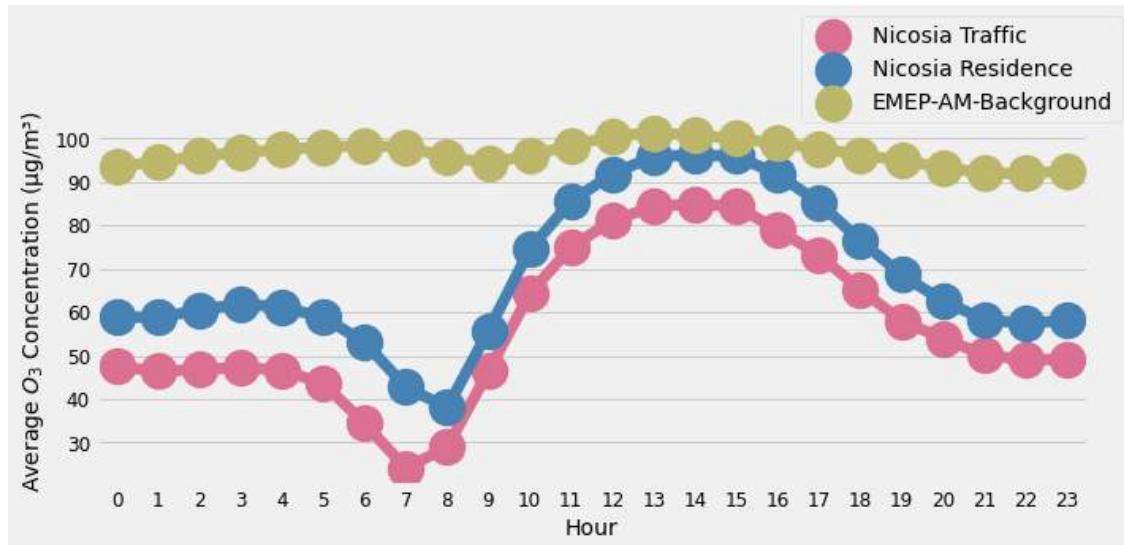
```
[21]: fig= plt.figure(figsize=(10,4))
sns.pointplot(x="Hour", y="NO_2", hue="Station",
               data=original, palette=['cyan','olive','green'],
               ci=None)
plt.legend(['Nicosia Traffic', 'Nicosia Residence', 'EMEP-AM-Background'], loc=2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$NO_{-2}$' + ' Concentration ( $\mu\text{g}/\text{m}^3$ )')
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 (\mu g/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()
```

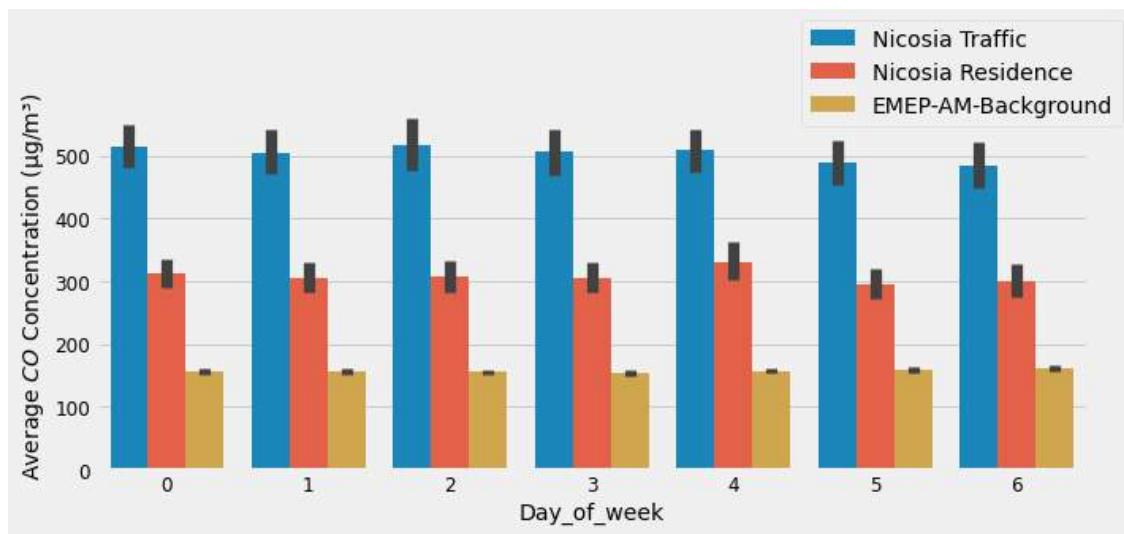


```
[23]: fig= plt.figure(figsize=(10,4))
sns.pointplot(x="Hour", y="O_3", hue="Station",
               data=original, palette=['palevioletred','steelblue','darkkhaki'],
               ci=None)
#plt.legend(['Nicosia Traffic', 'Nicosia Residence', 'EMEP-AM-Background'], loc=2, bbox_to_anchor = (0.7,1.2))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

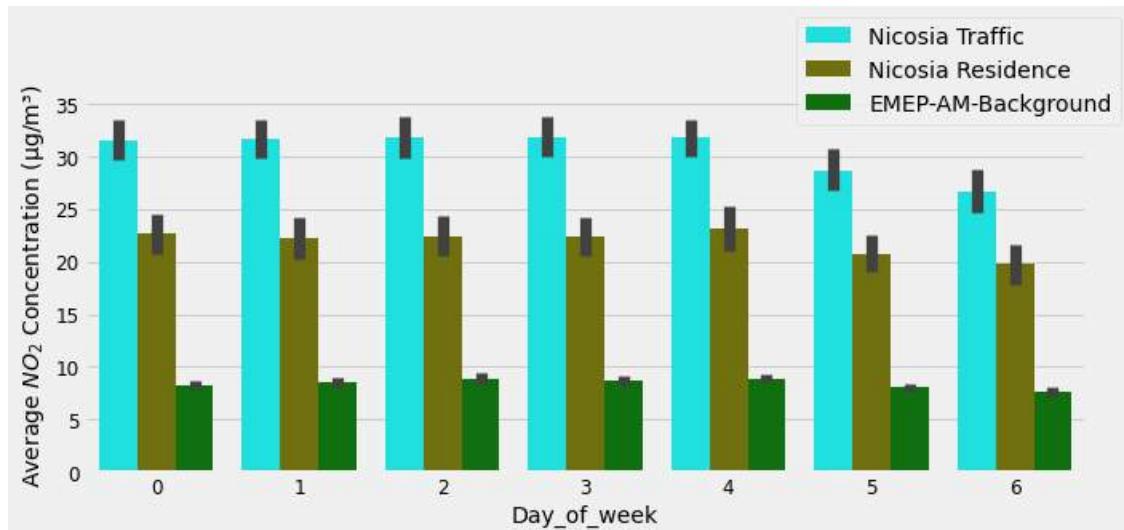


1.5 Day of Week visualization

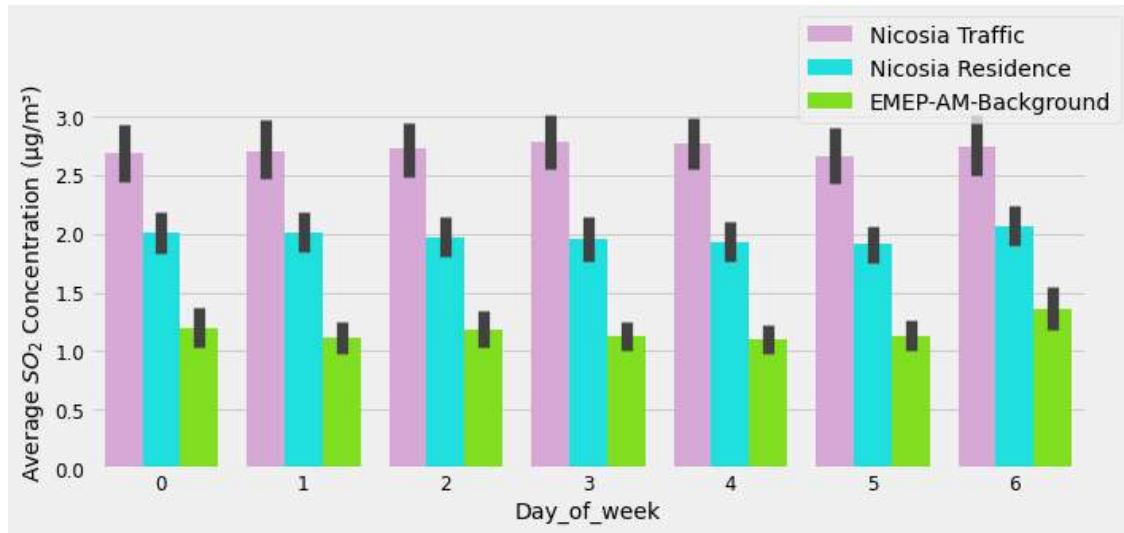
```
[24]: fig= plt.figure(figsize=(10,4))
sns.barplot(x="Day_of_week", y="CO", hue="Station", data=result)
# plt.legend(['Nicosia Traffic', 'Nicosia Residence', 'EMEP-AM-Background'], loc=2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$CO_{-}{}$' + ' Concentration (µg/m³)')
plt.legend(loc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



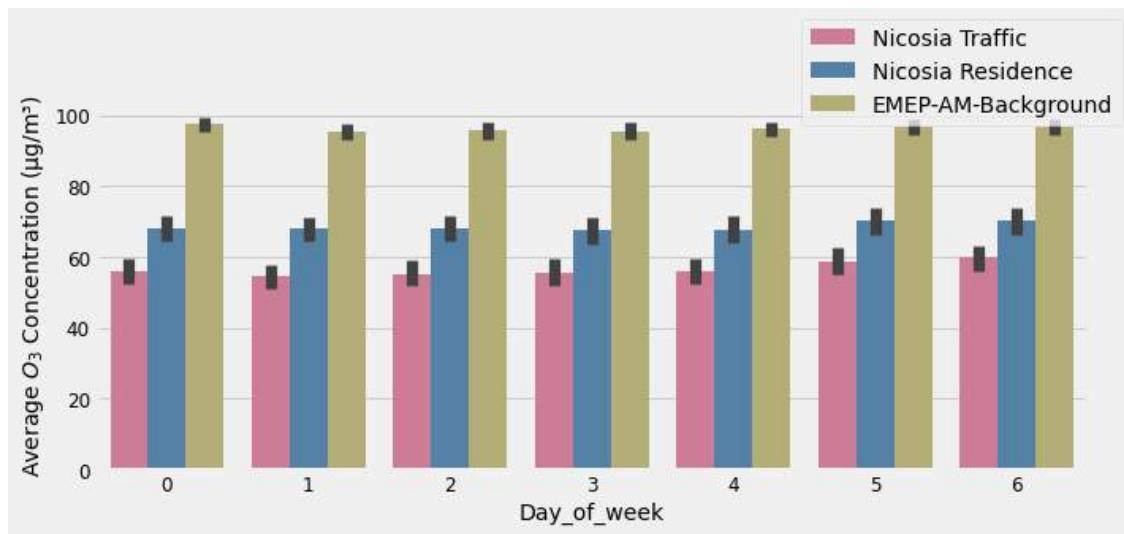
```
[25]: fig= plt.figure(figsize=(10,4))
sns.barplot(x="Day_of_week", y="NO_2", hue="Station", data=result,
             palette=['cyan','olive','green'])
# plt.legend(['Nicosia Traffic', 'Nicosia Residence', 'EMEP-AM-Background'], loc=2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$NO_{2}' + ' Concentration (\mu g/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 (\mu g/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'$\mathbf{O_3}$' + ' Concentration ($\mu\text{g}/\text{m}^3$)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()
```

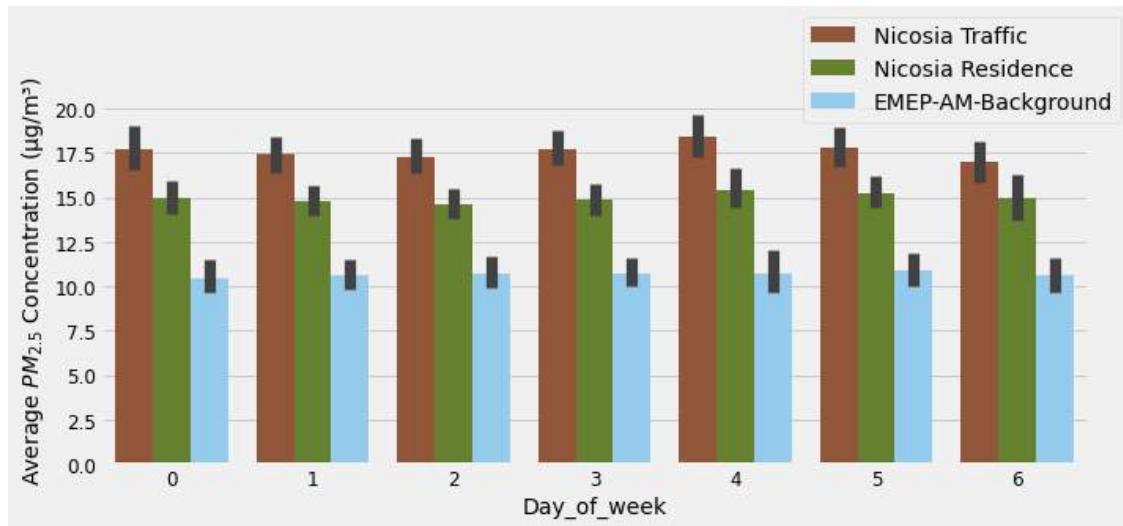


```
[28]: fig= plt.figure(figsize=(10,4))
```

```

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 (\mu g/m³)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()

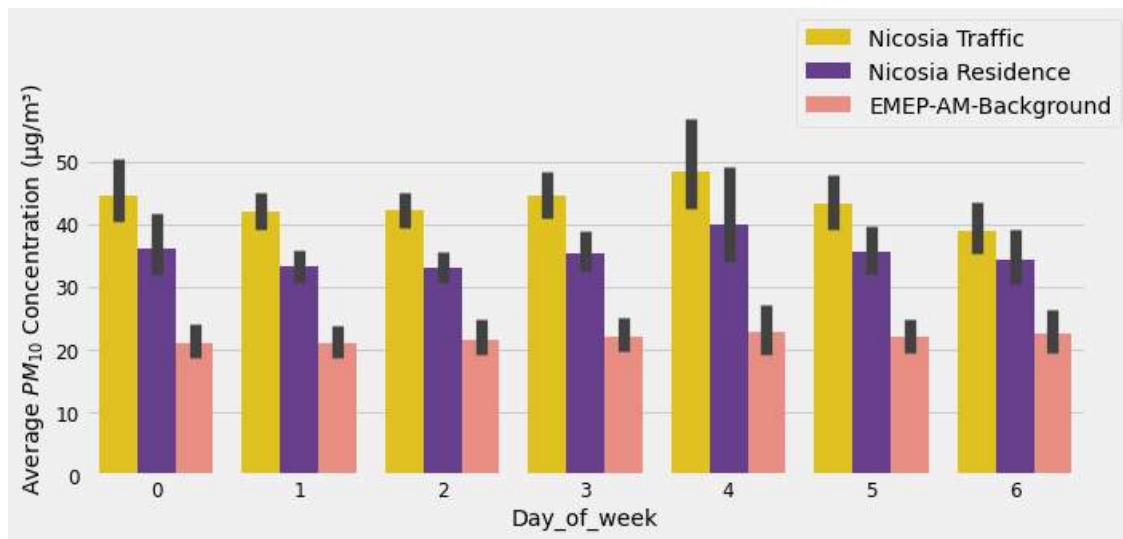
```



```

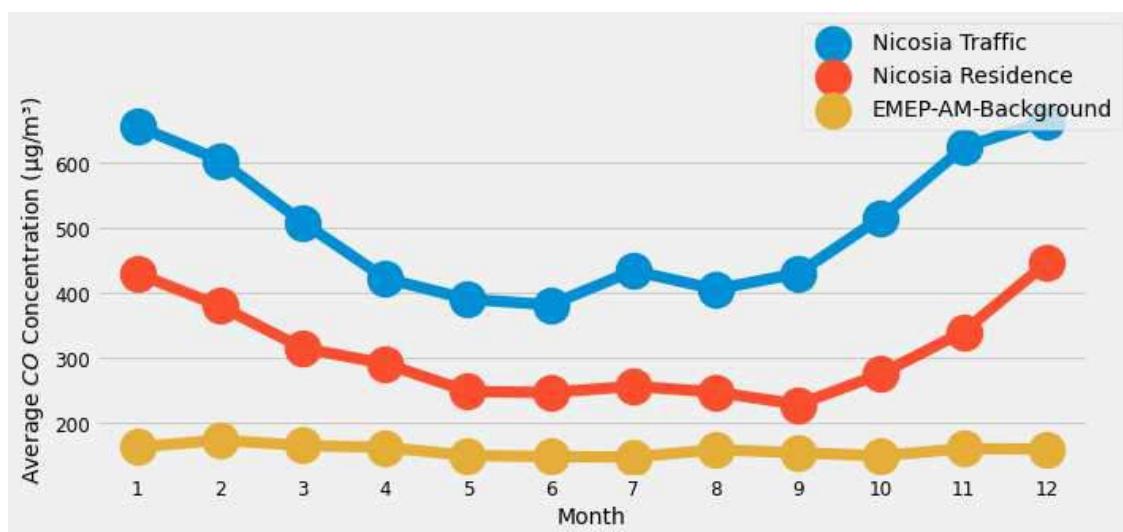
[29]: fig= plt.figure(figsize=(10,4))
sns.barplot(x="Day_of_week", y="PM_10", hue="Station", data=result,
            palette=['gold', 'rebeccapurple', 'salmon'])
plt.legend(['Nicosia Traffic', 'Nicosia Residence', 'EMEP-AM-Background'], loc=2,
           bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average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

```

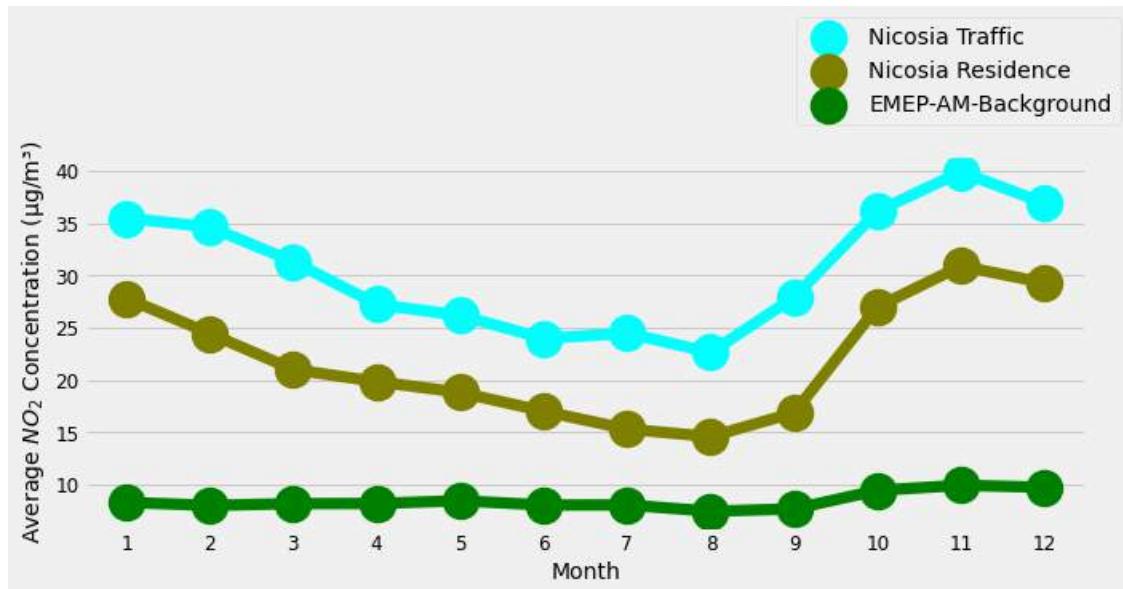


1.6 Monthly Visualization

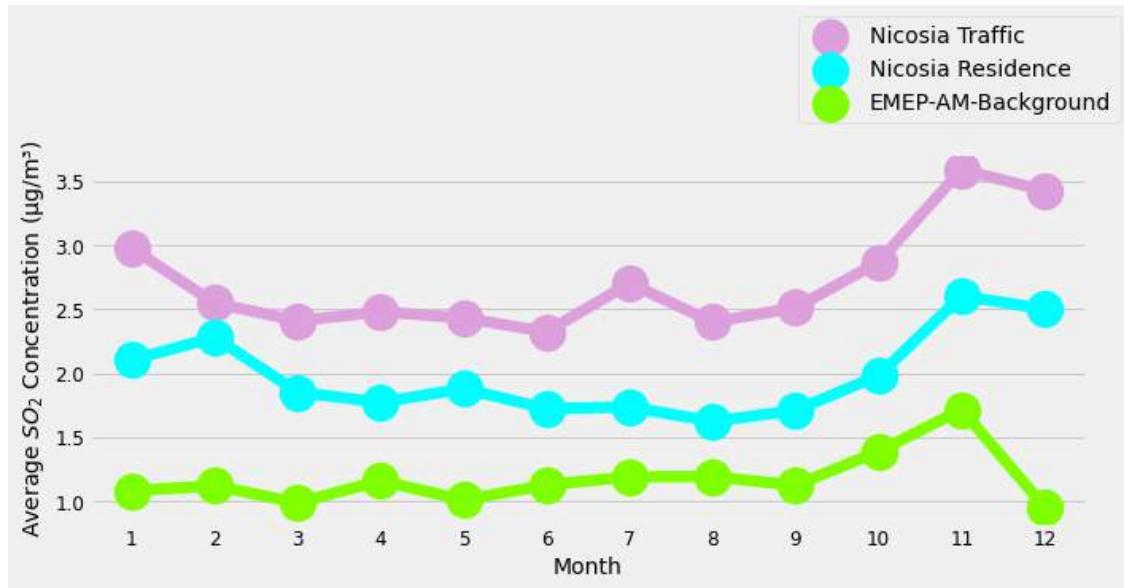
```
[30]: fig= plt.figure(figsize=(10,4))
sns.pointplot(x="Month", y="CO", hue="Station", data=result,
               ci=None)
plt.legend(['Nicosia Traffic', 'Nicosia Residence', 'EMEP-AM-Background'], loc=2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$CO_{}$' + ' Concentration (\mu g/m³)')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



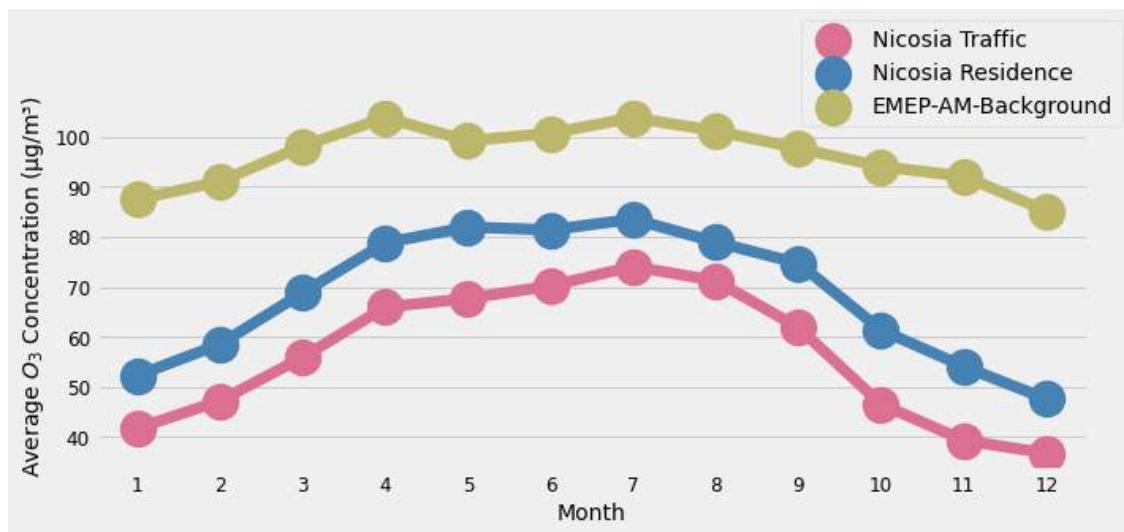
```
[31]: fig= plt.figure(figsize=(10,4))
sns.pointplot(x="Month", y="NO_2", hue="Station",
               data=result, palette=['cyan','olive','green'],
               ci=None)
#plt.legend(['Nicosia Traffic', 'Nicosia Residence', 'EMEP-AM-Background'], loc=2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$NO_{2}$' + ' Concentration ($\mu\text{g}/\text{m}^3$)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.4))
plt.show()
```



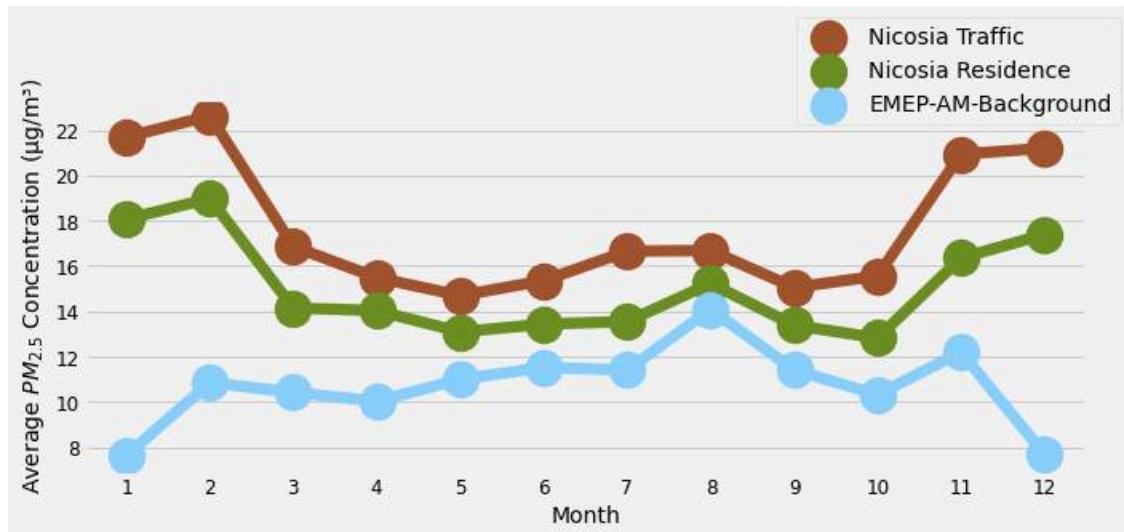
```
[32]: fig= plt.figure(figsize=(10,4))
sns.pointplot(x="Month", y="SO_2", hue="Station",
               data=result, palette=['plum','aqua','chartreuse'],
               ci=None)
#plt.legend(['Nicosia Traffic', 'Nicosia Residence', 'EMEP-AM-Background'], loc=2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$SO_{2}$' + ' Concentration ($\mu\text{g}/\text{m}^3$)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.4))
plt.show()
```



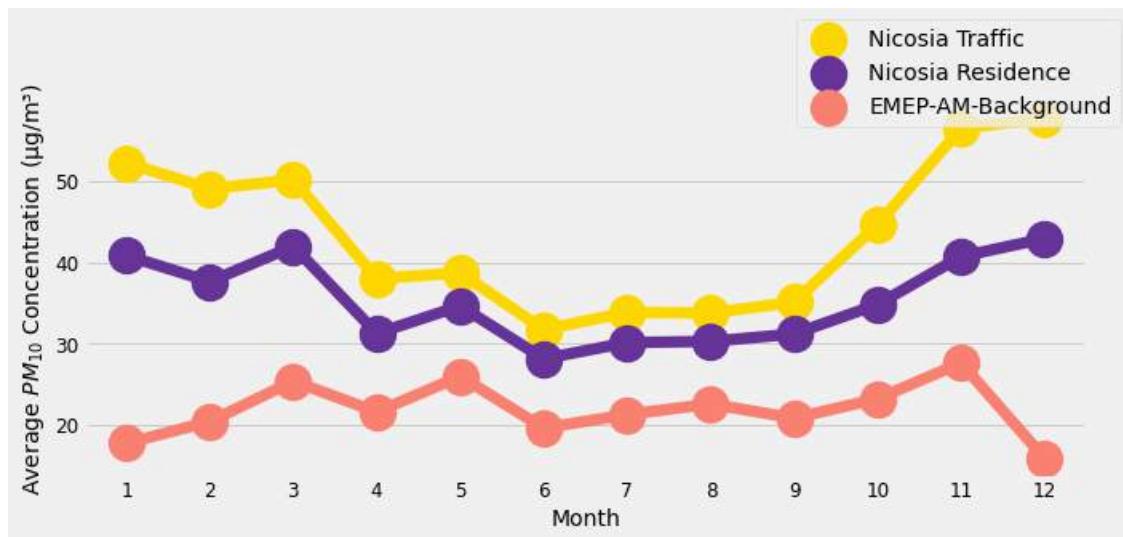
```
[33]: fig= plt.figure(figsize=(10,4))
sns.pointplot(x="Month", y="O_3", hue="Station",
               data=result, palette=['palevioletred','steelblue','darkkhaki'],
               ci=None)
plt.legend(['Nicosia Traffic', 'Nicosia Residence', 'EMEP-AM-Background'], loc=2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$\text{O}_3$' + ' Concentration ( $\mu\text{g}/\text{m}^3$ )')
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 ($\mu g/m^3$)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()
```

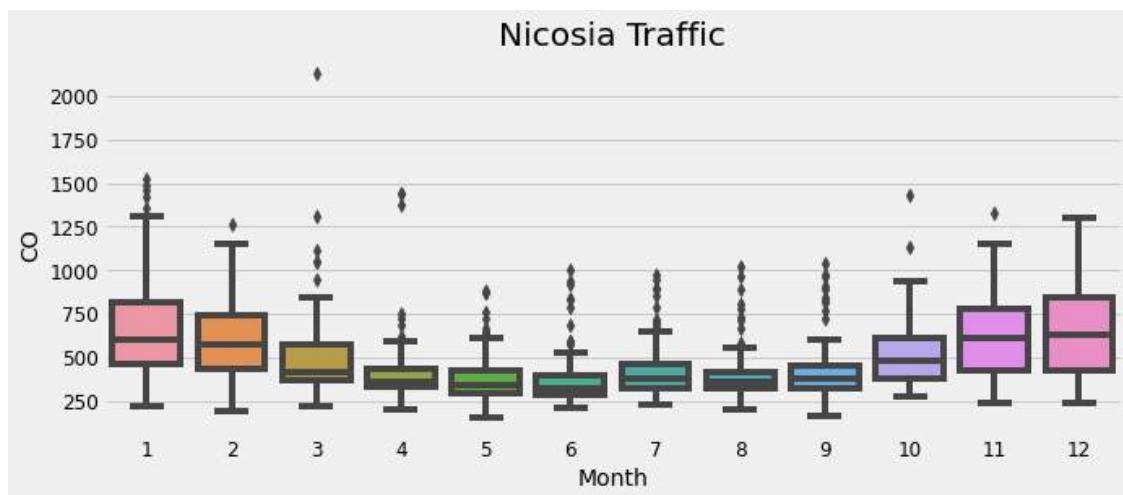


```
[35]: fig= plt.figure(figsize=(10,4))
sns.pointplot(x="Month", y="PM_10", hue="Station",
               data=result, palette=['gold','rebeccapurple','salmon'],
               ci=None)
#plt.legend(['Nicosia Traffic', 'Nicosia Residence', 'EMEP-AM-Background'], loc=2, bbox_to_anchor = (0.7,1.2))
plt.ylabel( 'Average ' + r'$PM_{10}$' + ' Concentration ($\mu g/m^3$)')
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.show()
```

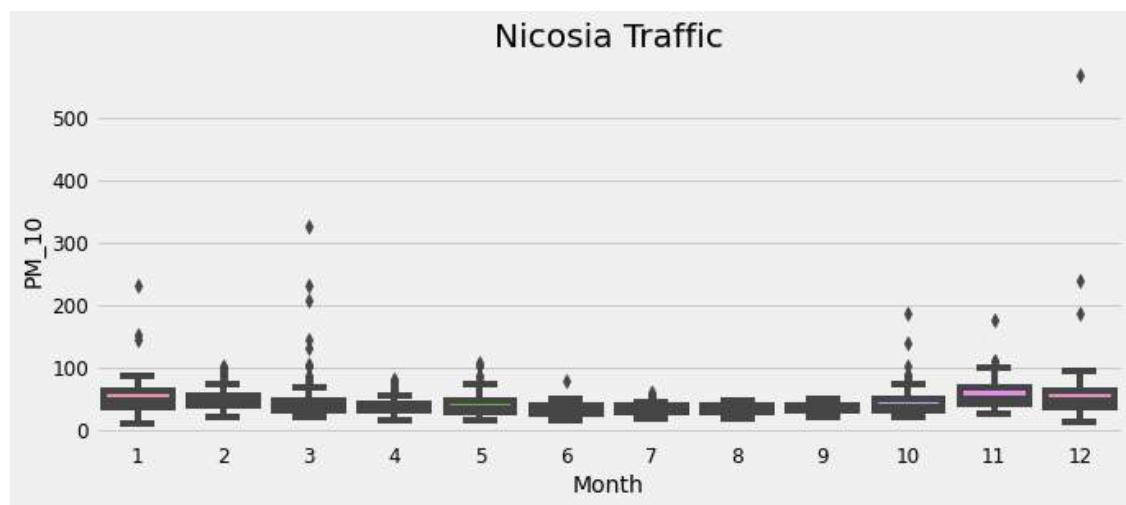
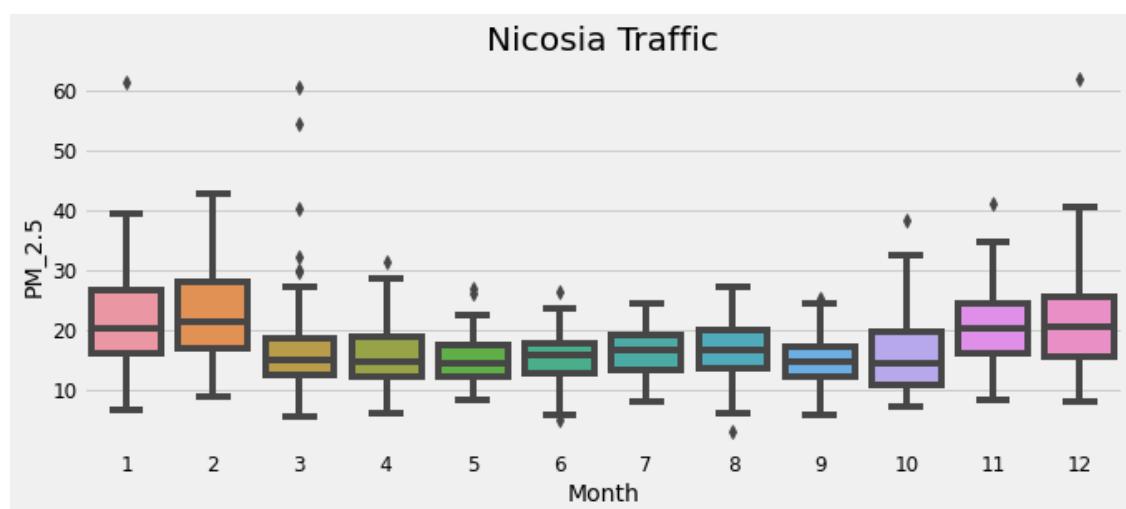


```
[36]: palette=[[<code>'cyan', 'olive', 'green'</code>], [<code>'plum', 'aqua', 'chartreuse'</code>], [<code>'palevioletred', 'steelblue', 'darkblue'</code>], [<code>'sienna', 'olivedrab', 'lightskyblue'</code>], [<code>'gold', 'rebeccapurple', 'salmon'</code>], [<code>'plum', 'olivedrab', 'darkblue'</code>]]
```

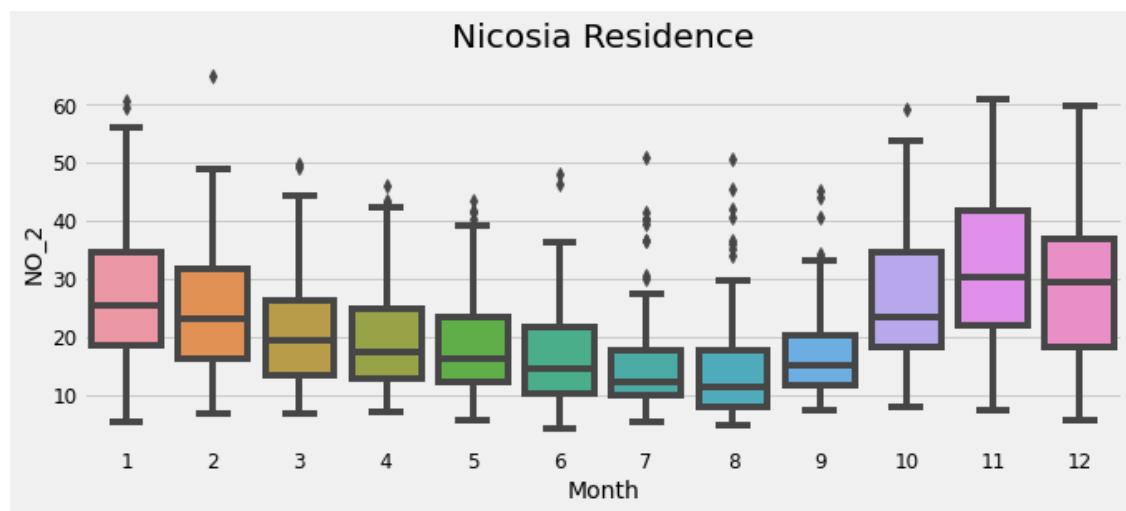
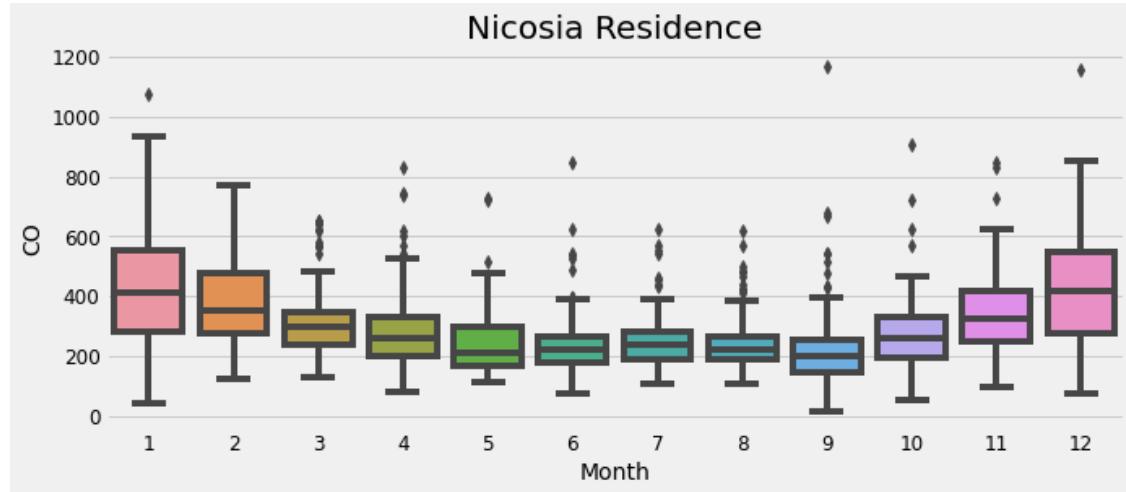
```
[37]: column=['CO', 'NO_2', 'SO_2', 'O_3', 'PM_2.5', 'PM_10']
for i in range(len(column)):
    fig= plt.figure(figsize=(10,4))
    sns.boxplot(data=data_NT, x='Month', y=column[i])
    plt.title('Nicosia Traffic')
    plt.show()
```

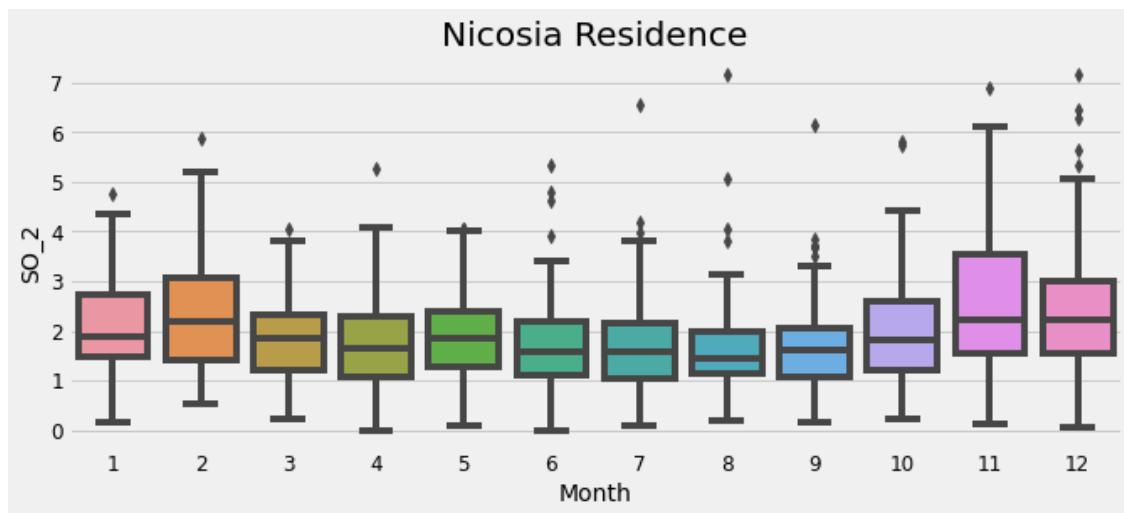


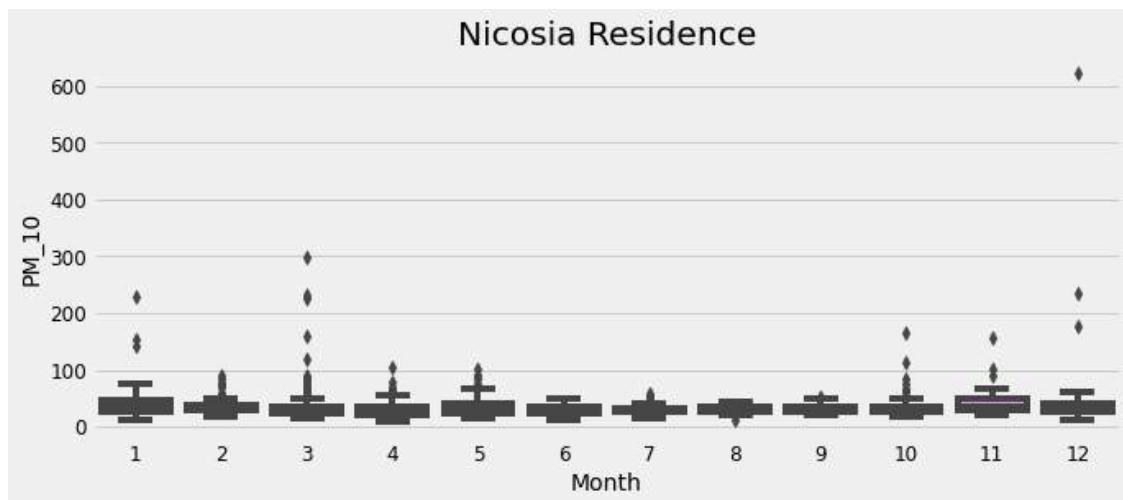
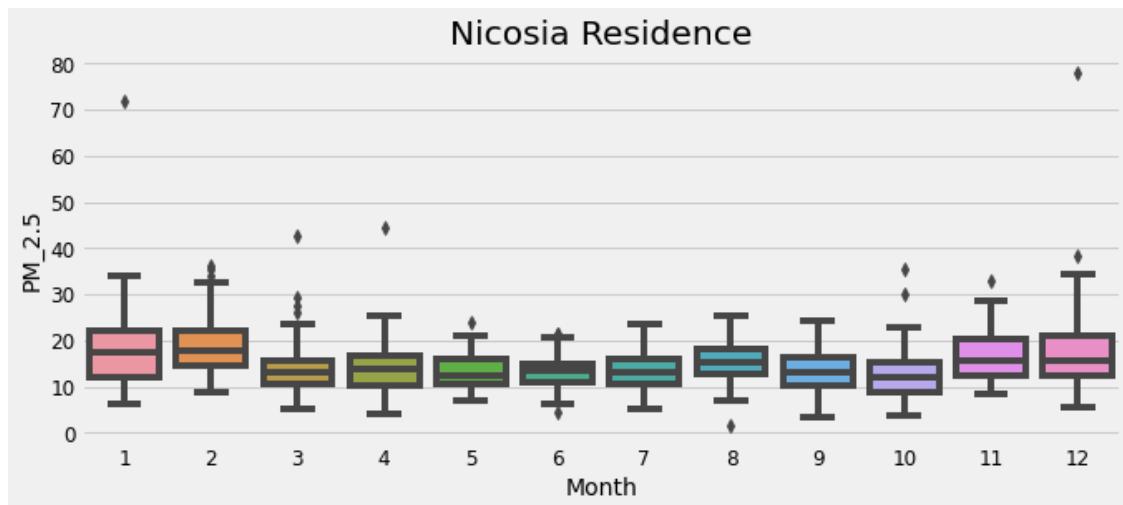




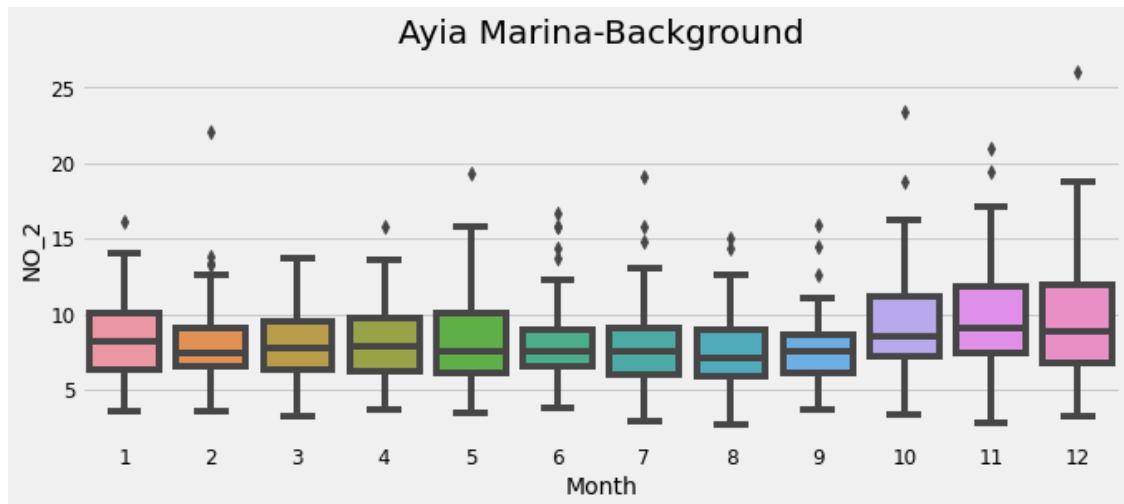
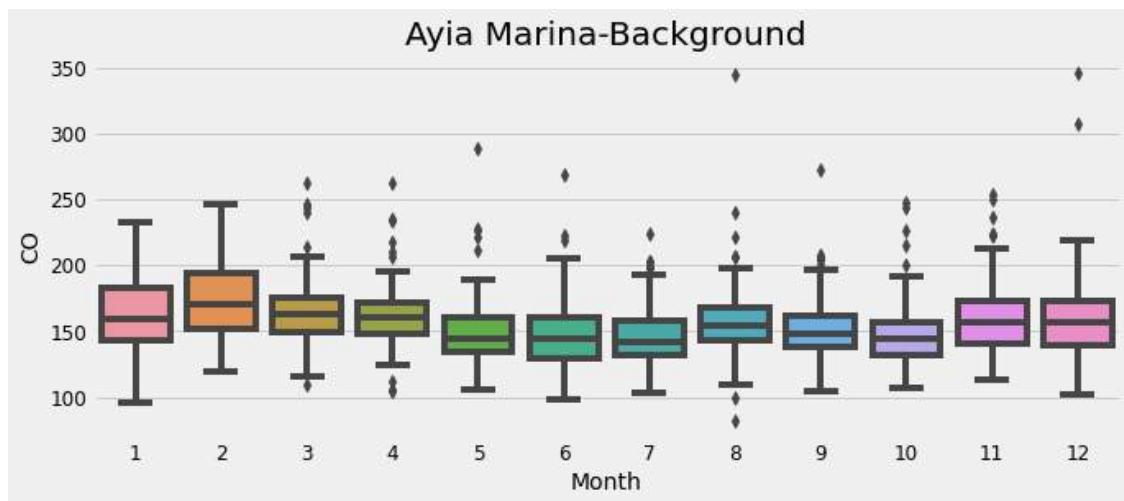
```
[38]: column=['CO','NO_2','SO_2','O_3','PM_2.5','PM_10']
for i in range(len(column)):
    fig= plt.figure(figsize=(10,4))
    sns.boxplot(data=data_NR, x='Month', y=column[i])
    plt.title('Nicosia Residence')
    plt.show()
```

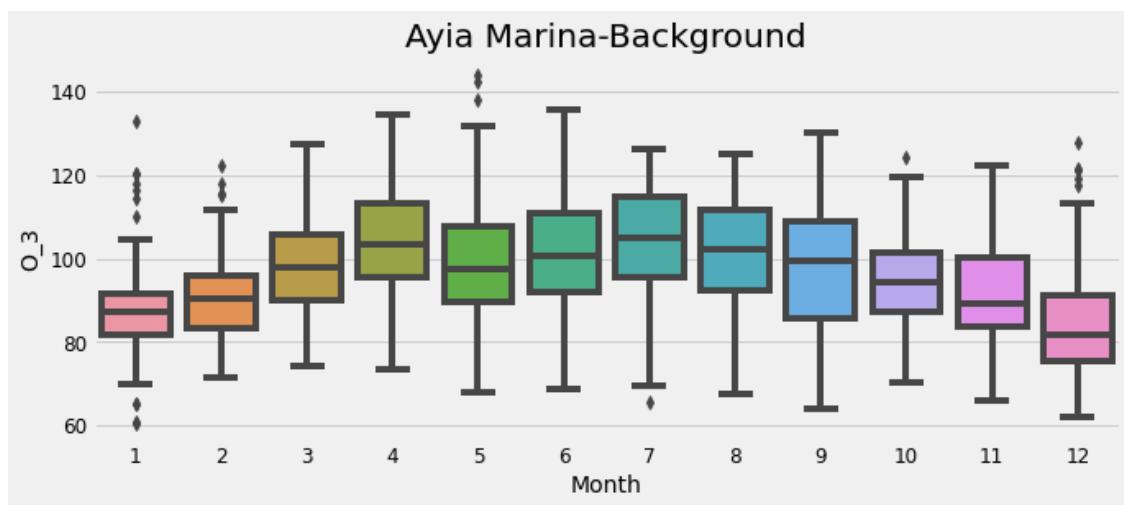
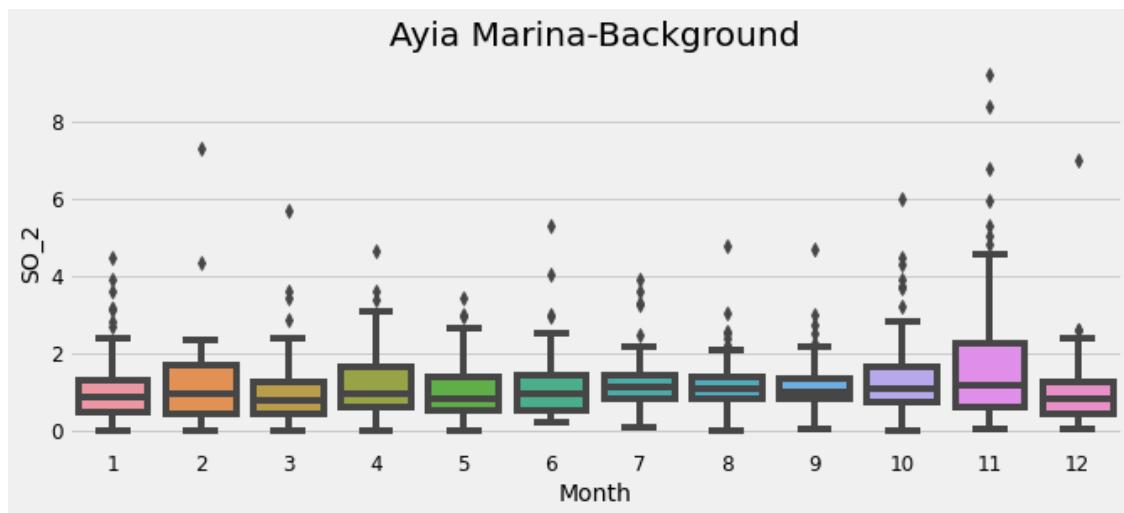


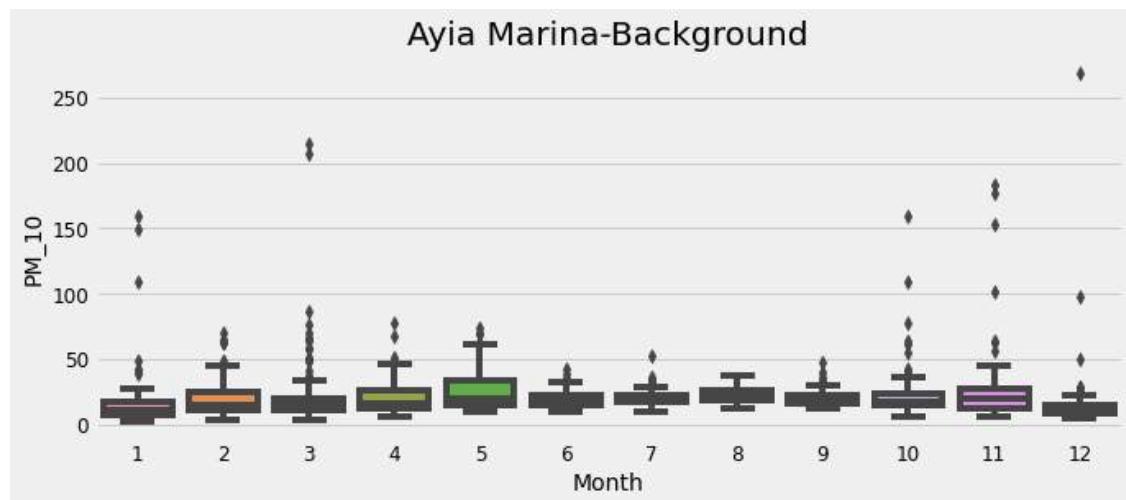
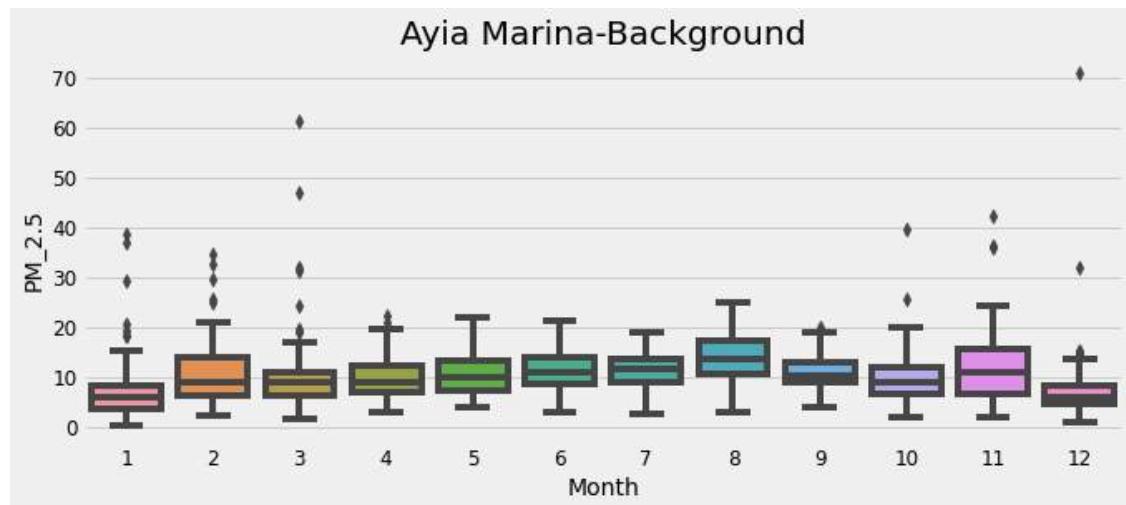




```
[39]: column=['CO','NO_2','SO_2','O_3','PM_2.5','PM_10']
for i in range(len(column)):
    fig= plt.figure(figsize=(10,4))
    sns.boxplot(data=data_AM, x='Month', y=column[i])
    plt.title('Ayia Marina-Background')
    plt.show()
```

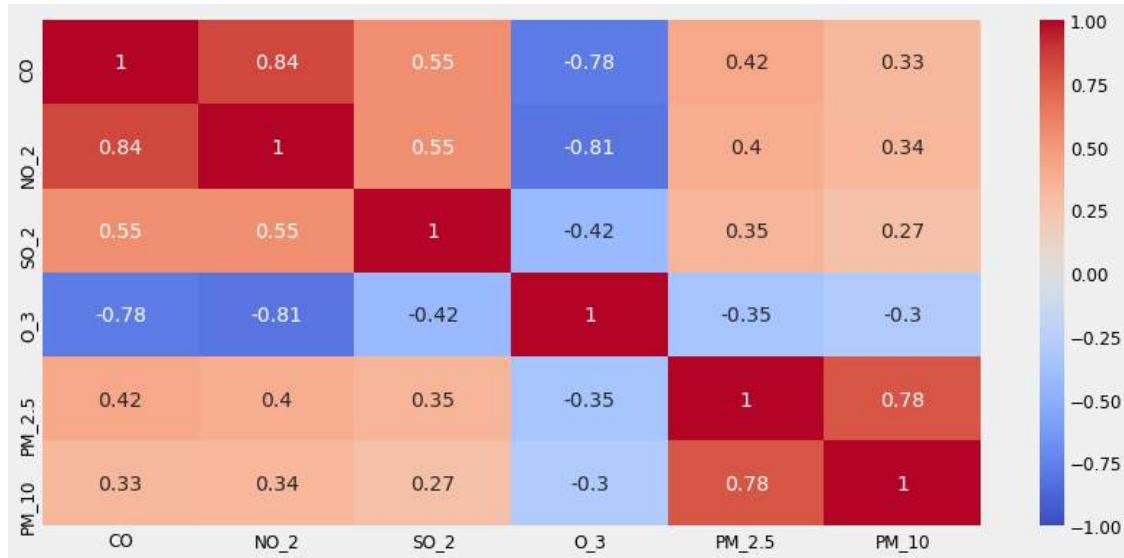






```
[40]: fig= plt.figure(figsize=(13,6))
sns.heatmap(result[['CO','NO_2','SO_2','O_3','PM_2.5','PM_10']].corr(), annot = True,
            vmin=-1, vmax=1,
            center= 0, cmap= 'coolwarm')
```

```
[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]:

	NO_2	SO_2	O_3	CO	PM_2.5	PM_10	Year	\
Date								
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	

	Month	Day_of_week	Day	Hour	AQI	Pollution Level
Date						
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]:

	NO_2	SO_2	O_3	CO	PM_2.5	PM_10	Year	\
Date								
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	

	Month	Day_of_week	Day	Hour	AQI	Pollution Level
Date						
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]:

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

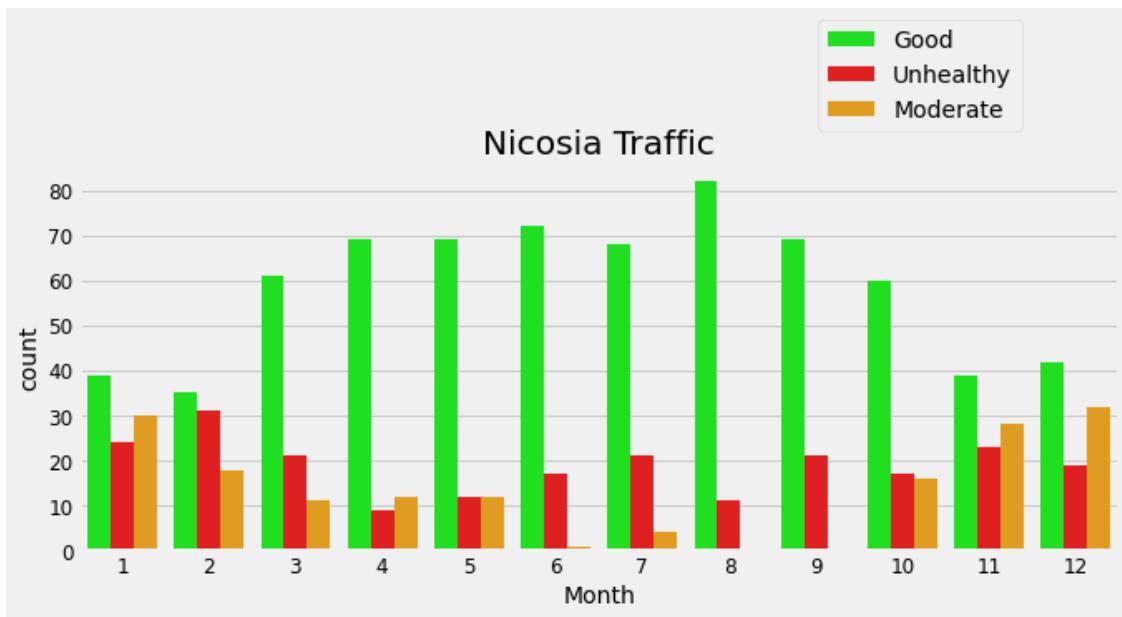
	Month	Day_of_week	Day	Hour	AQI	Pollution Level
Date						
2017-01-01	1	6	1	0	30.40	Good
2017-01-02	1	0	2	0	35.15	Good
2017-01-03	1	1	3	0	38.76	Good
2017-01-04	1	2	4	0	38.54	Good
2017-01-05	1	3	5	0	49.05	Good

[56]:

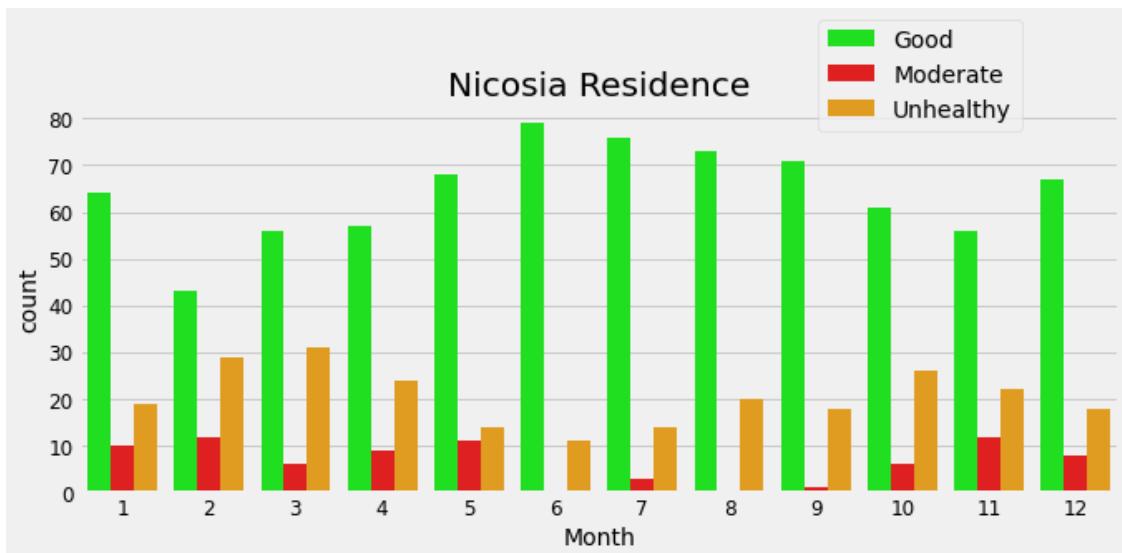
```

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()

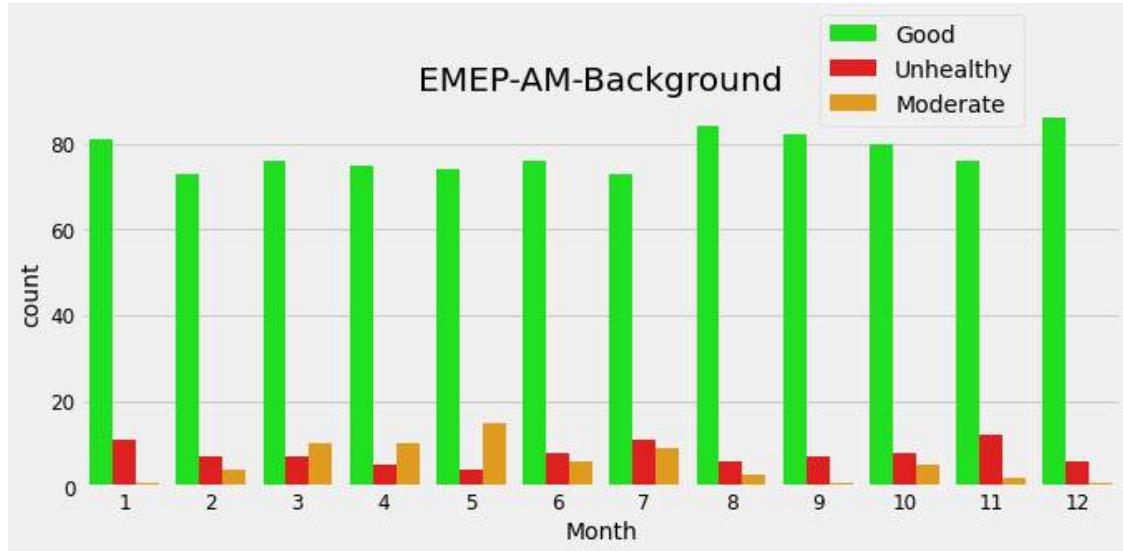
```



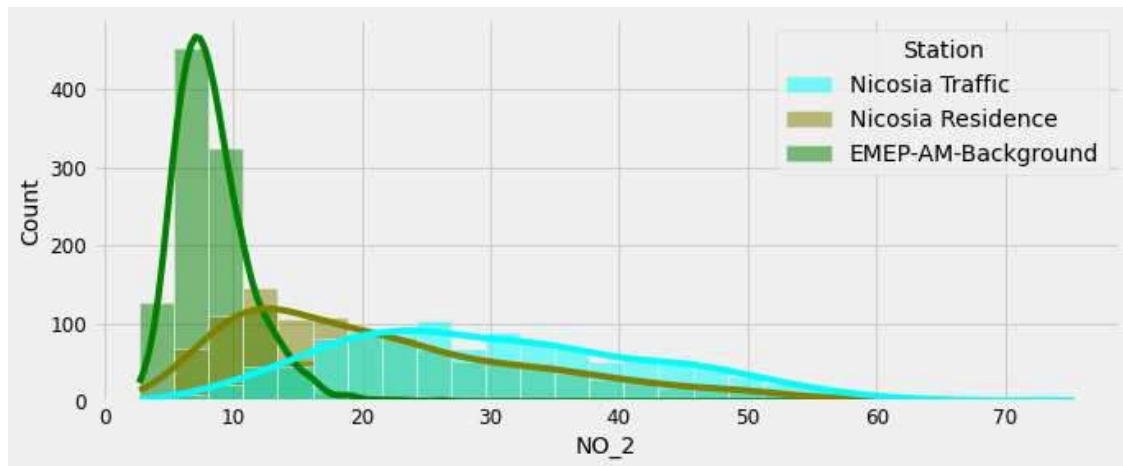
```
[57]: fig= plt.figure(figsize=(10,4))
sns.countplot(x = 'Month', data = data_NR,hue='Pollution_Level',palette=['lime','red','orange'])
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.title('Nicosia Residence')
plt.show()
```



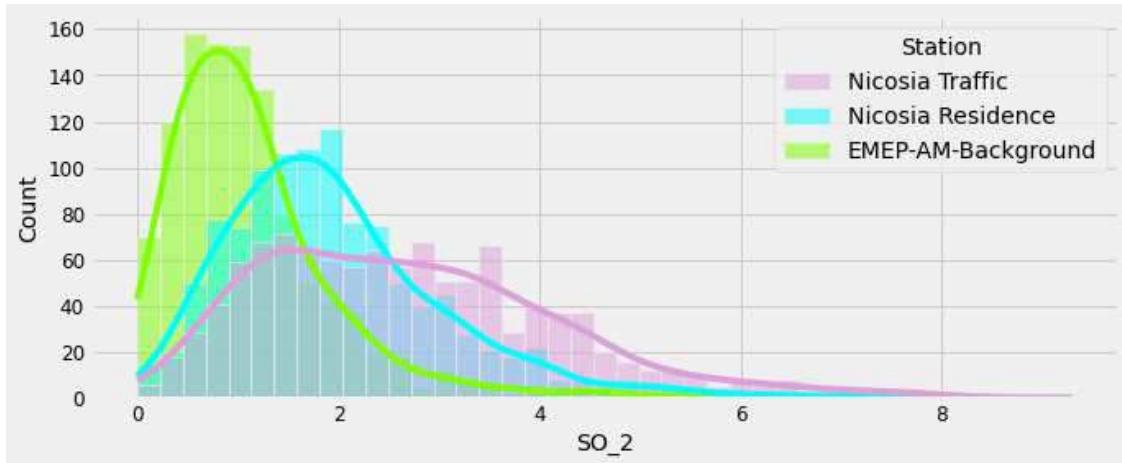
```
[58]: fig= plt.figure(figsize=(10,4))
sns.countplot(x = 'Month', data = data_AM,hue='Pollution_Level',palette=['lime','red','orange'])
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.25))
plt.title('EMEP-AM-Background')
plt.show()
```



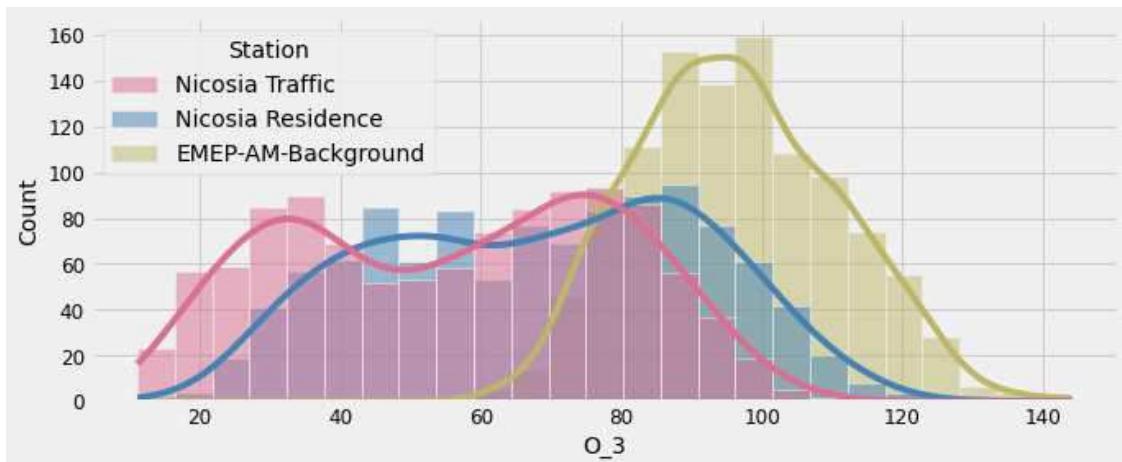
```
[59]: # importing packages
fig= plt.figure(figsize=(10,4))
sns.histplot(x='NO_2', data=result, kde=True, hue='Station', palette=palette[0])
plt.show()
```



```
[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()
```

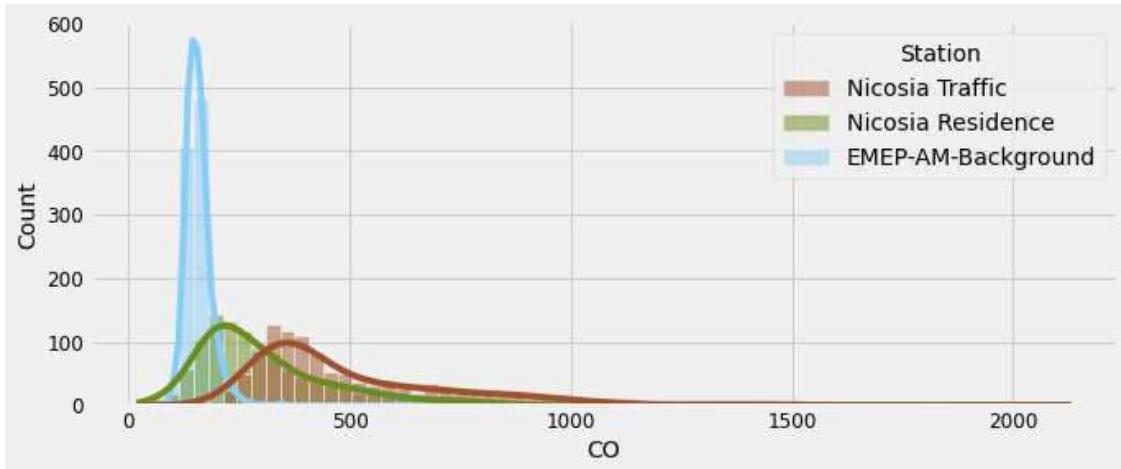


```
[61]: # importing packages
fig= plt.figure(figsize=(10,4))
sns.histplot(x='O_3', data=result, kde=True, hue='Station', palette=palette[2])
plt.show()
```

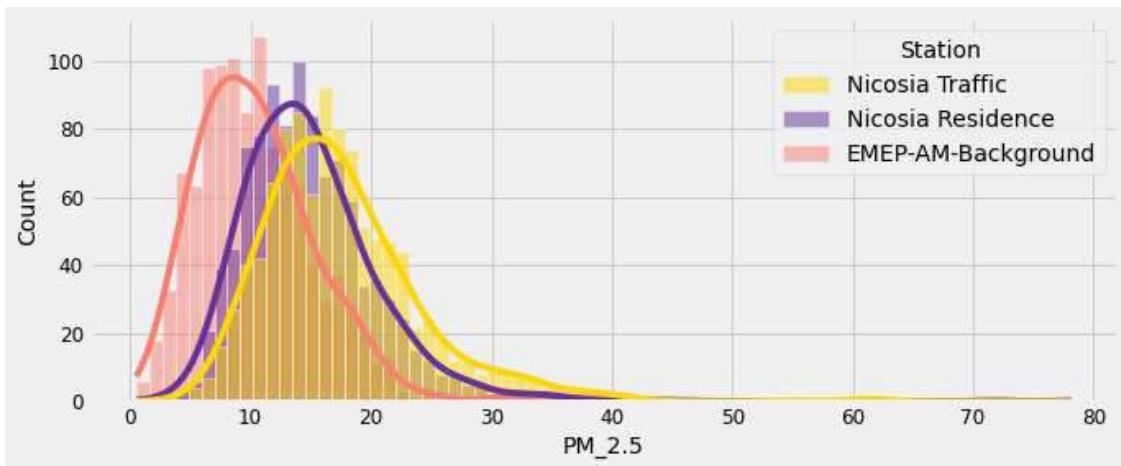


```
[62]: # importing packages
fig= plt.figure(figsize=(10,4))
sns.histplot(x='CO', data=result, kde=True, hue='Station', palette=palette[3])
plt.show()
```

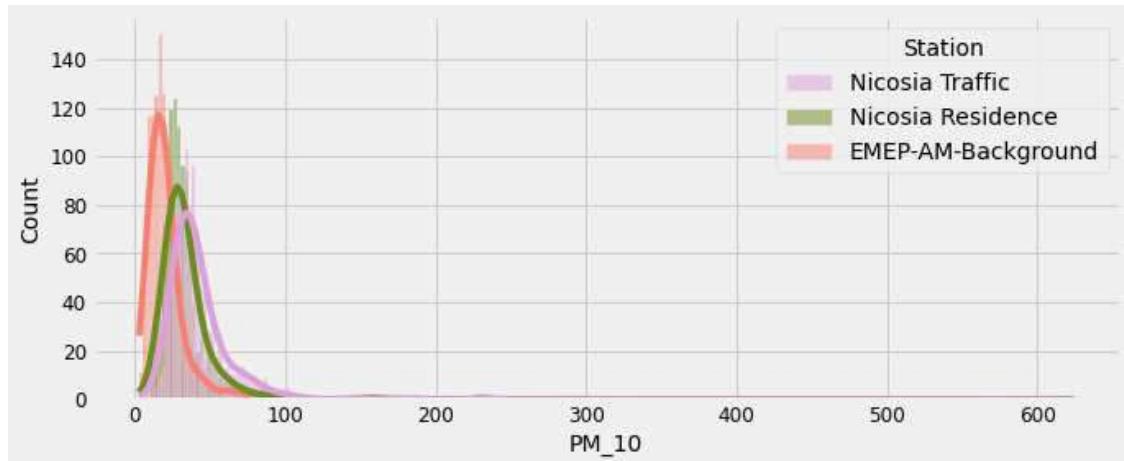
```
plt.show()
```



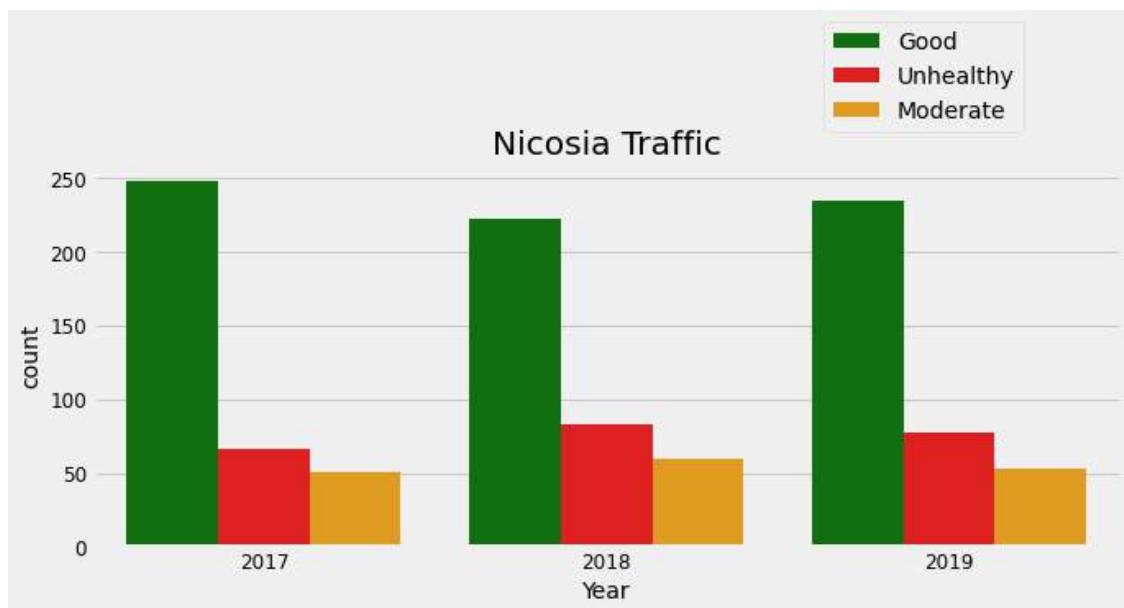
```
[63]: # importing packages
fig= plt.figure(figsize=(10,4))
sns.histplot(x='PM_2.5', data=result, kde=True, hue='Station', palette=palette[4])
plt.show()
```



```
[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()
```

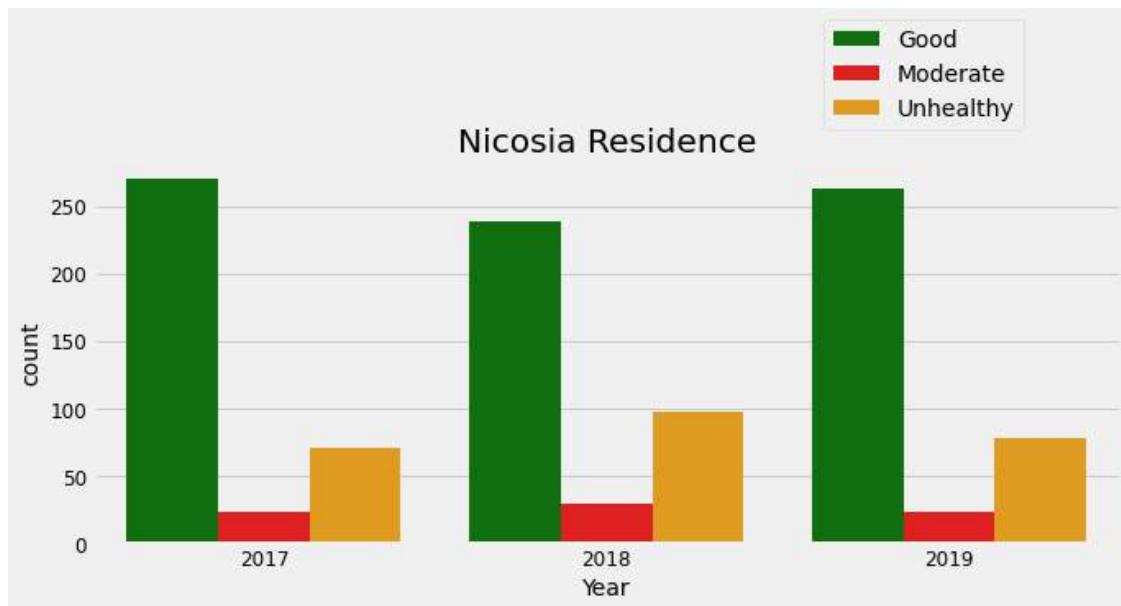


```
[65]: fig= plt.figure(figsize=(10,4))
sns.countplot(x = 'Year', data = data_NT,hue='Pollution_Level',palette=['green','red','orange'])
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.4))
plt.title('Nicosia Traffic')
plt.show()
```

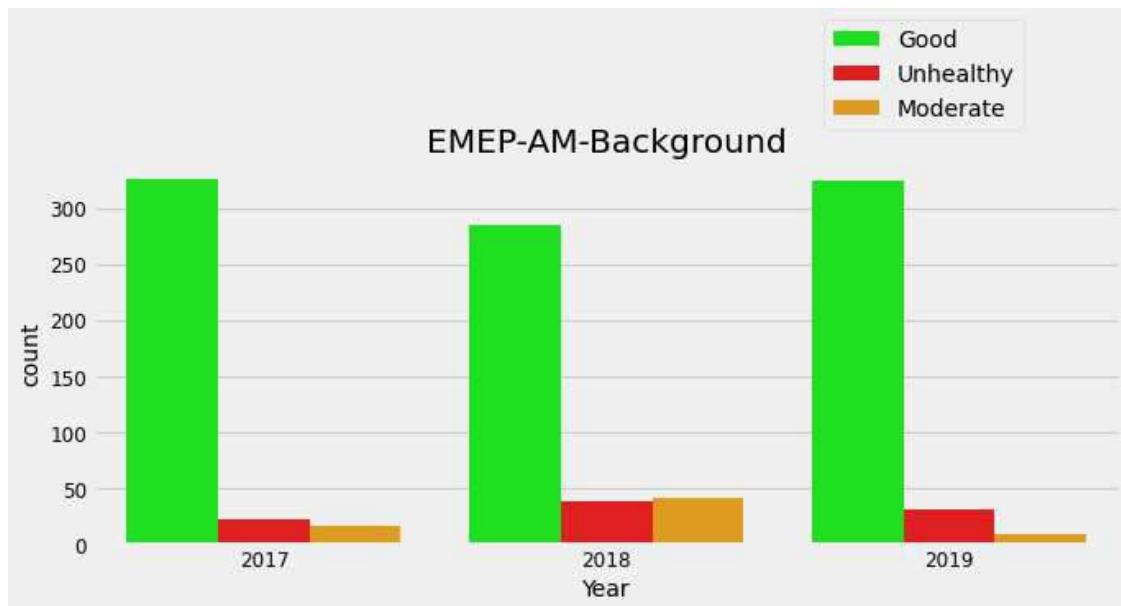


```
[66]: fig= plt.figure(figsize=(10,4))
sns.countplot(x = 'Year', data = data_NR,hue='Pollution_Level',palette=['green','red','orange'])
plt.legend(loc = 2, bbox_to_anchor = (0.7,1.4))
```

```
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()

```

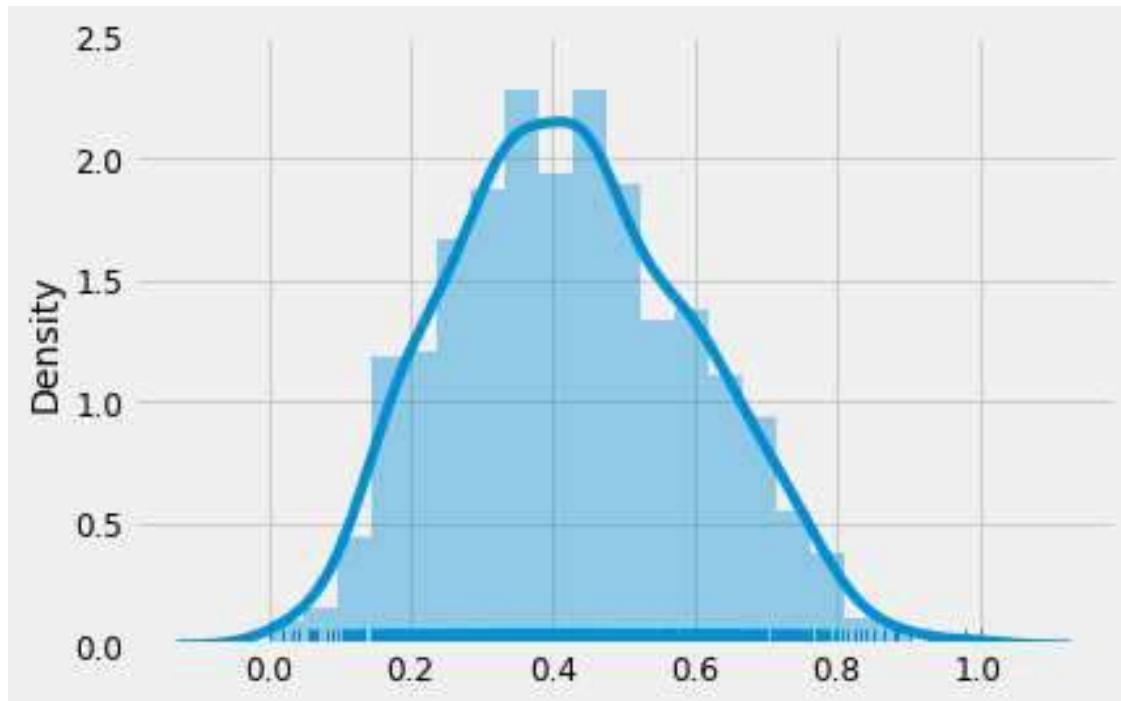
1.10 Checking distribution of data

```
[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=0_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.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
2019-12-31 20:00:00	2269.500000	12		1	31	20
2019-12-31 21:00:00	2996.699951	12		1	31	21
2019-12-31 22:00:00	2517.300049	12		1	31	22
2019-12-31 23:00:00	1723.000000	12		1	31	23

1.11 Determining Skewness and Kurtosis 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())
```

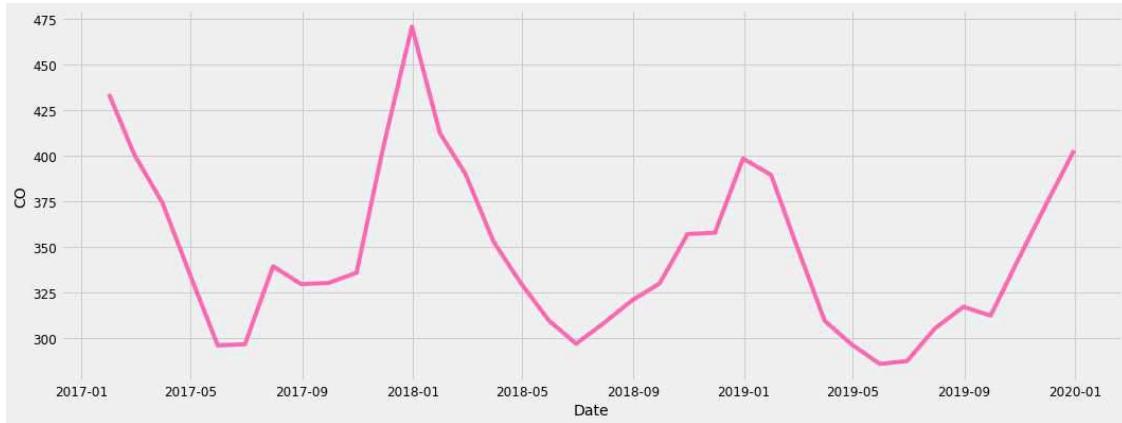
```
[75]: 25.936581574330027
```

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

1.12 Data Visualization

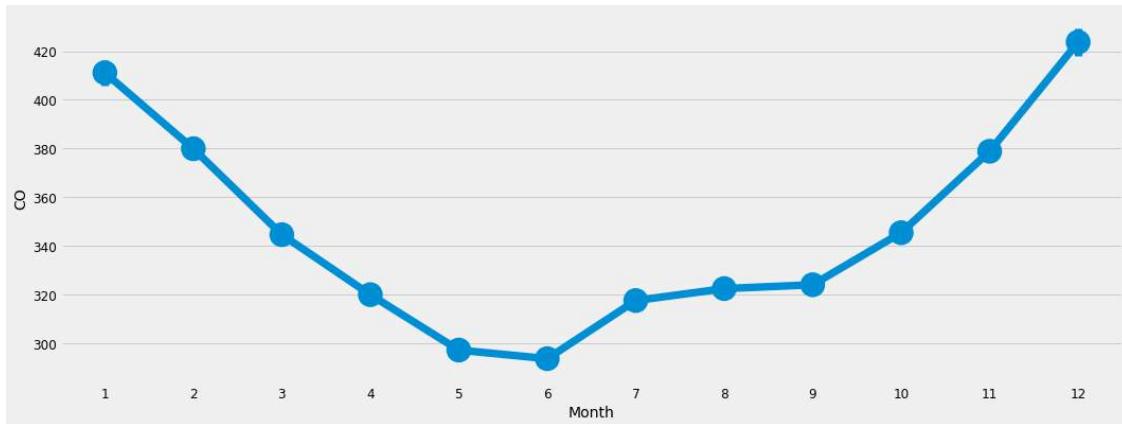
```
[77]: import matplotlib.pyplot as plt
import seaborn as sns
data_by_month=data1.resample('M').mean()
fig= plt.figure(figsize=(16,6))
#colors =
→['lightpink', 'pink', 'fuchsia', 'mistyrose', 'hotpink', 'deppink', 'magenta']
sns.lineplot(x=data_by_month.index,y='CO',data=data_by_month,color='hotpink')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

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



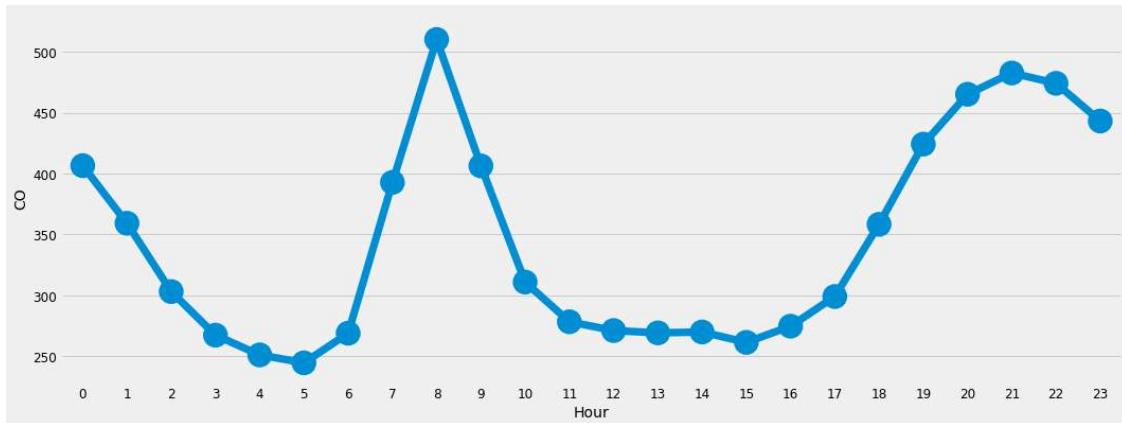
```
[78]: fig= plt.figure(figsize=(16,6))
sns.pointplot(data=data1,x='Month',y='CO',color='magenta')
```

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



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

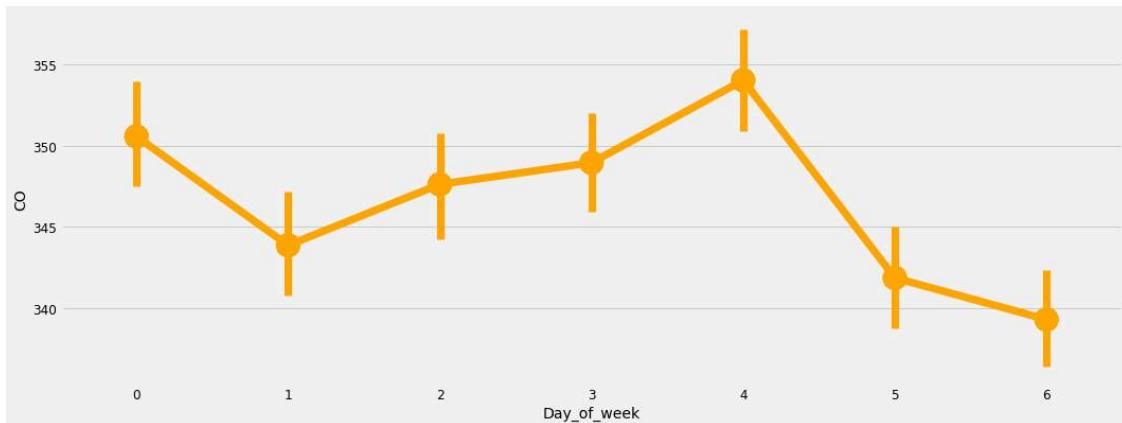
```
[79]: <AxesSubplot:xlabel='Hour', ylabel='CO'>
```



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

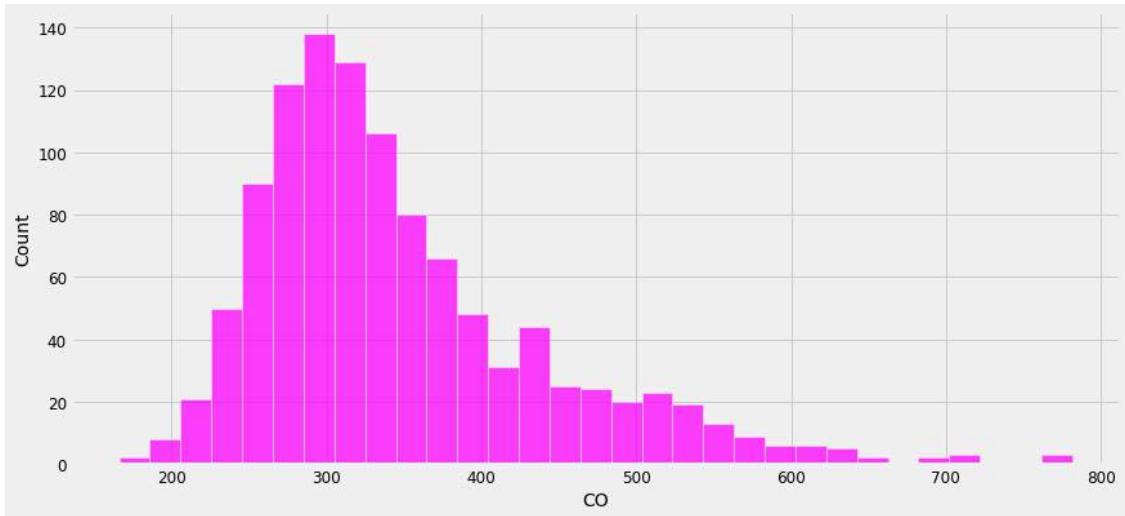
```
[80]: fig= plt.figure(figsize=(16,6))
sns.pointplot(data=data1,x='Day_of_week',y='CO',color='orange')
```

```
[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')
```

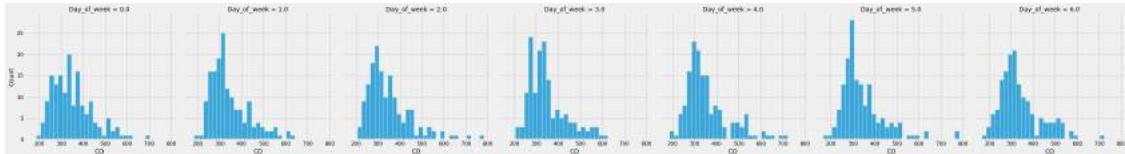
```
[81]: <AxesSubplot:xlabel='CO', ylabel='Count'>
```



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

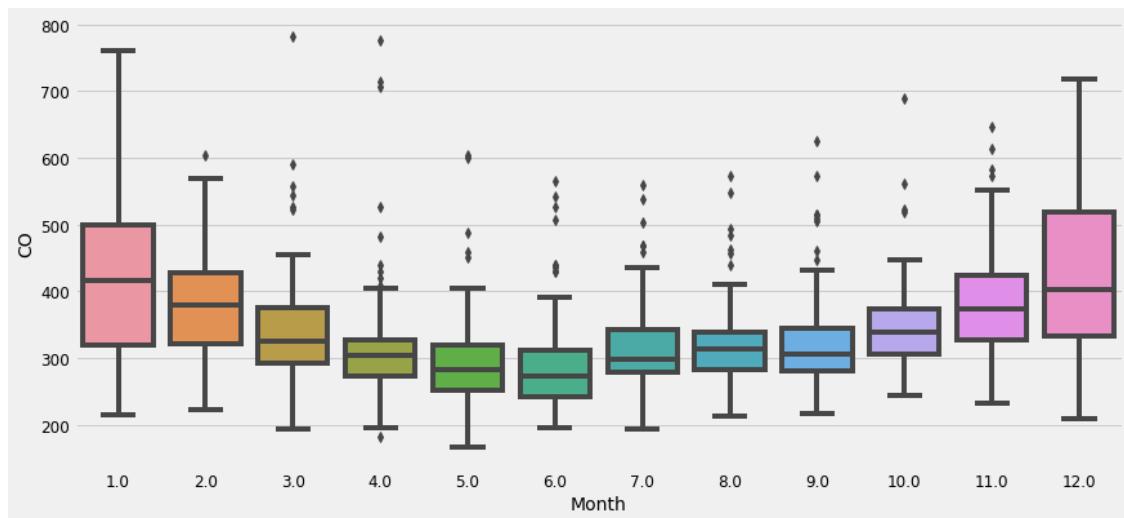
```
[82]: <seaborn.axisgrid.FacetGrid at 0x1563413a0>
```

<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'>
```

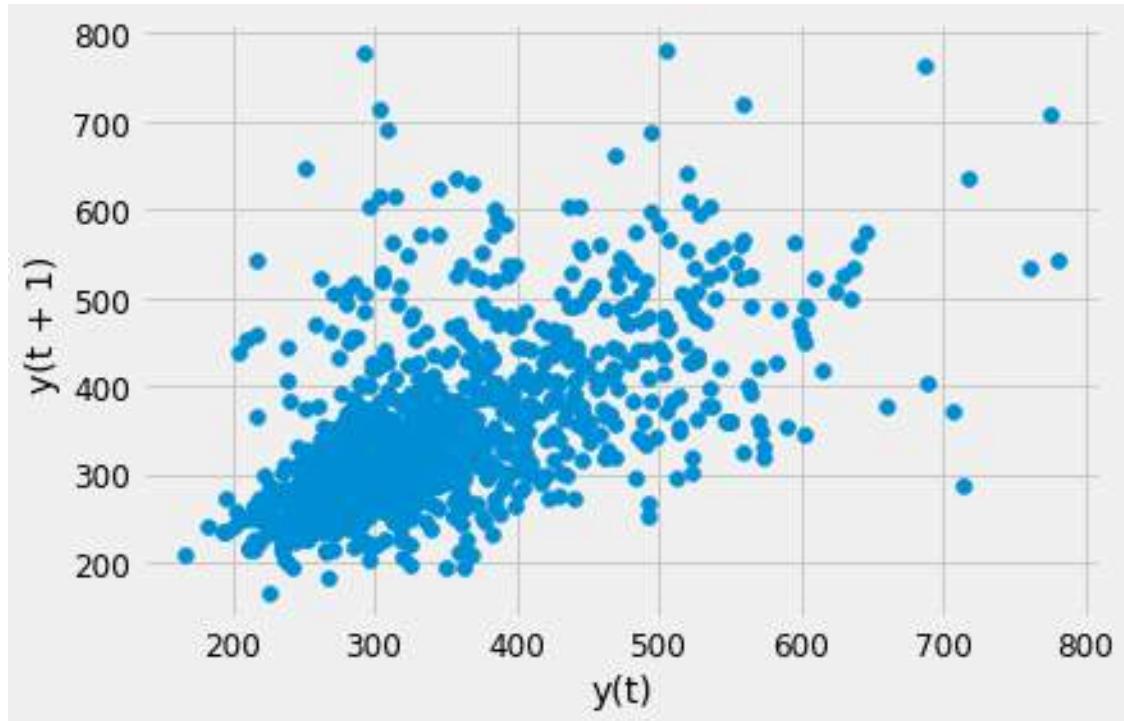


1.13 Time Series Lag Plot

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.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.fillna(column_means)
data.head()
```

```
[85]:          CO      date
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
```

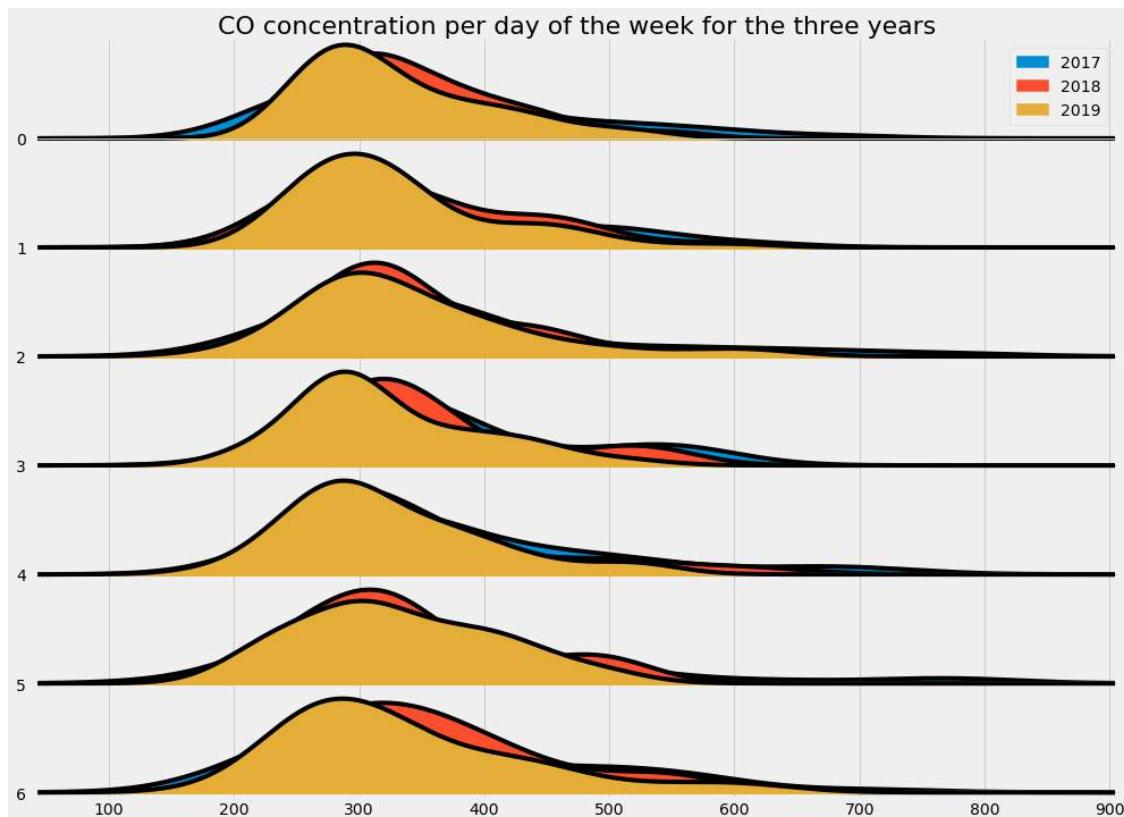
```
[86]: CO =data['CO'].to_list()
date=data['date'].to_list()
CO=pd.Series(CO,date)
C02017=list(CO['2017'])
C02018=list(CO['2018'])
C02019=list(CO['2019'])
a_dict = {'Date':CO.index[:365],
          "2017":C02017,
          "2018":C02018,"2019":C02019}
df = pd.DataFrame(a_dict)
df.set_index('Date')
df['Day_of_week']=df.index.dayofweek
df.head()
```

```
[86]:          2017      2018      2019  Day_of_week
Date
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.joypy(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()
```

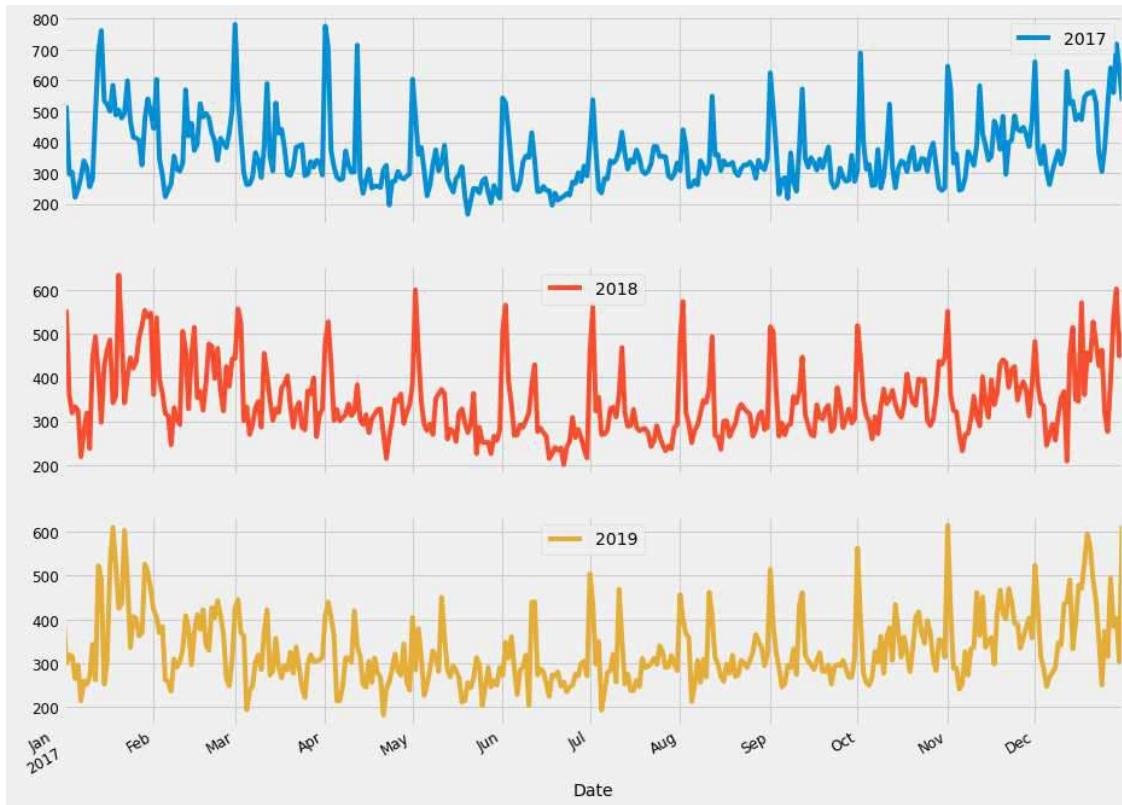
<Figure size 1120x480 with 0 Axes>



```
[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.

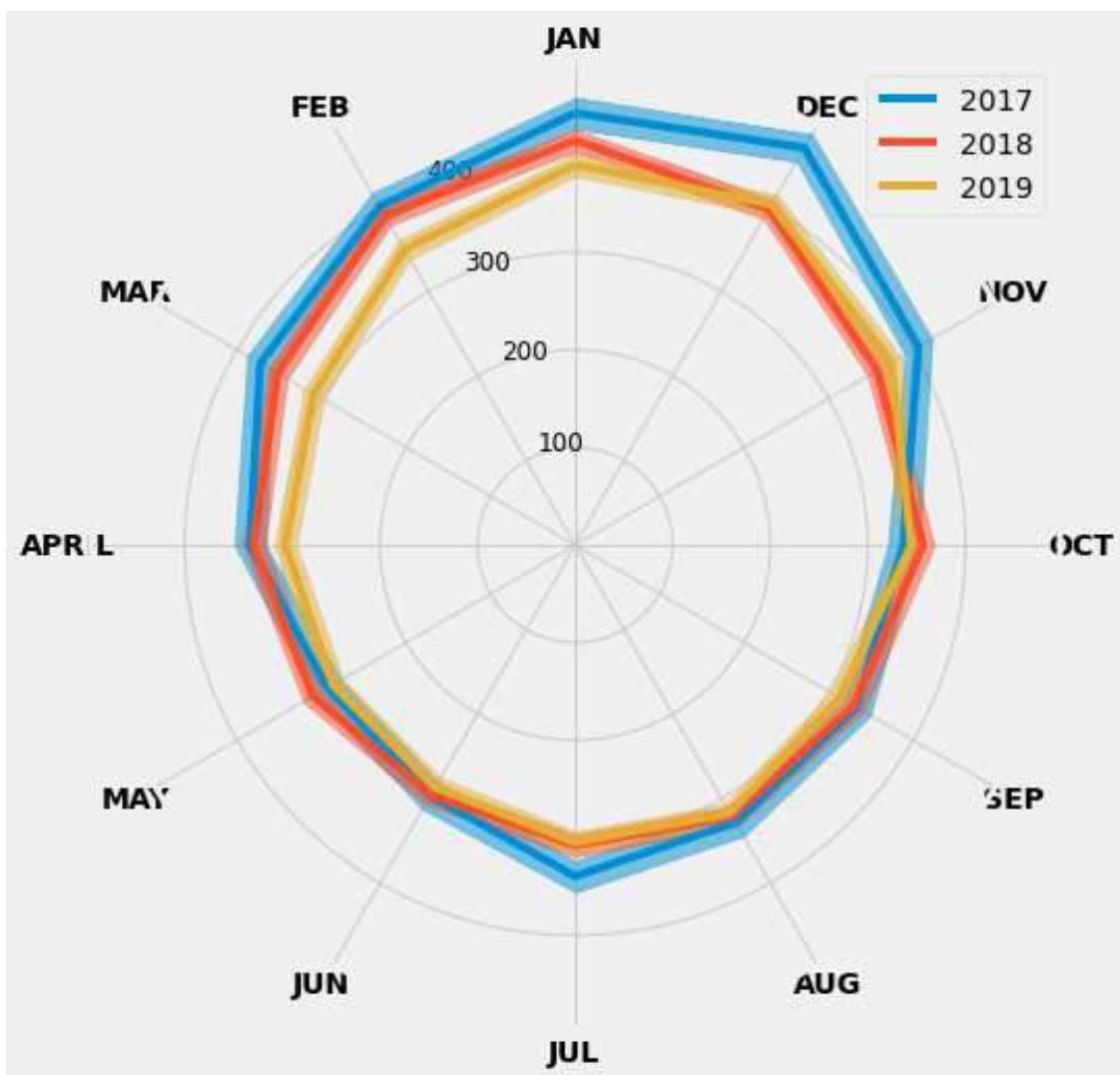
    DataFrame should be indexed 0-6 with 0=Monday, 6=Sunday
    """
    # add last row to complete cycle (otherwise plot lines don't connect)
    df = df.append(df.iloc[0, :])
    #convert index to radians
    radians = np.linspace(0, 2 * np.pi, num=12, endpoint=False)
    df.index = [radians[day] for day 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))

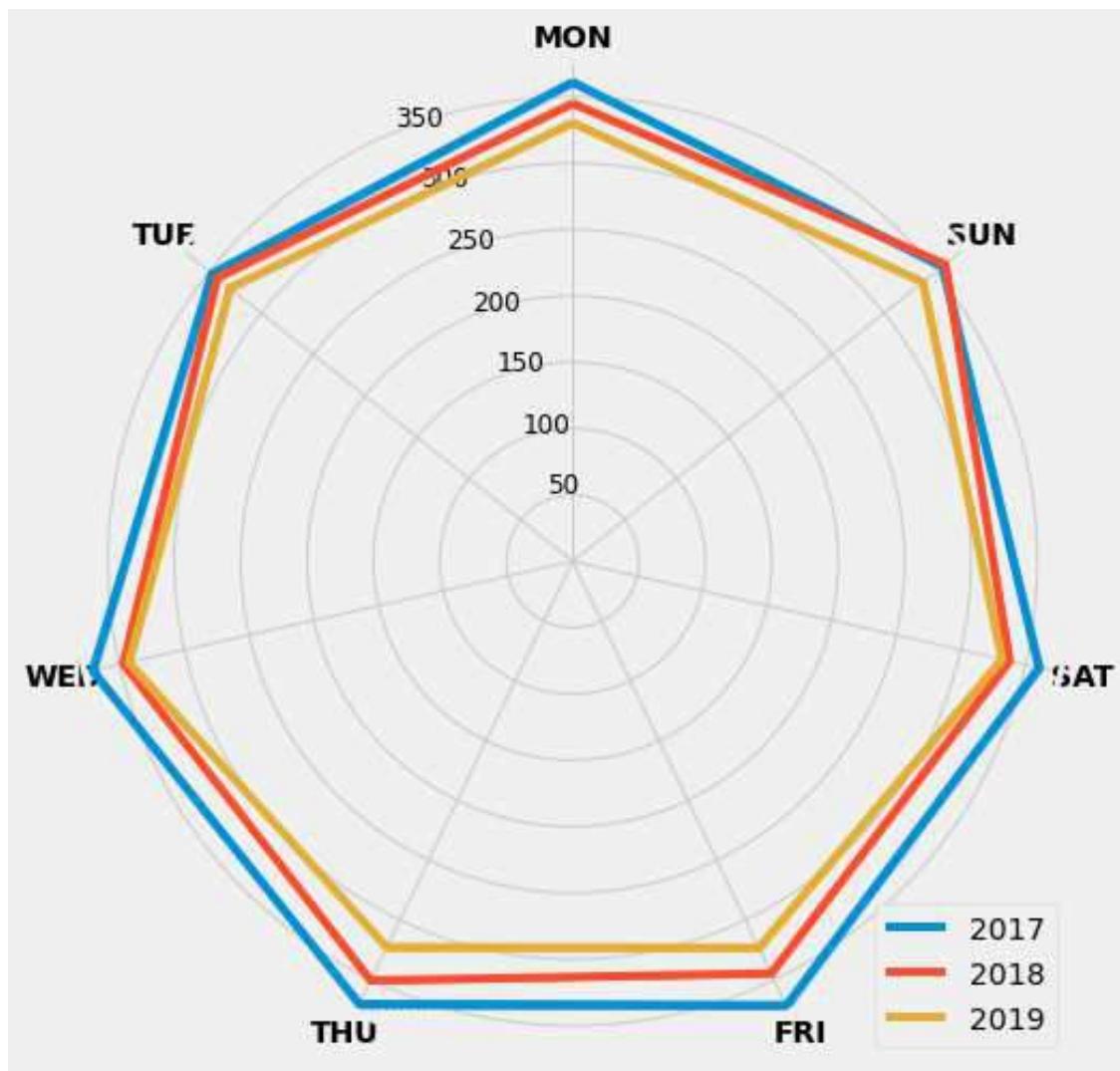
```



```
[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
```

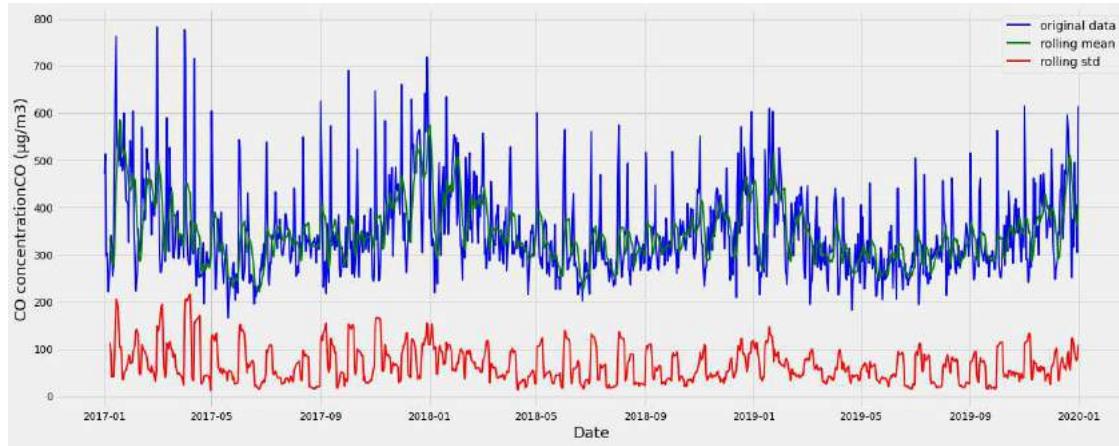
```
[95]:
```

	CO	Month	Day_of_week	Day	Hour	Year
Date						
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:')
dfoutput=pd.Series(dfoutput[0:4], index=['Test statistic','p-value','#lagsUsed','Number of Observations'])
for key,value in dfoutput[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 hypothesis and conclude that the data is stationary.

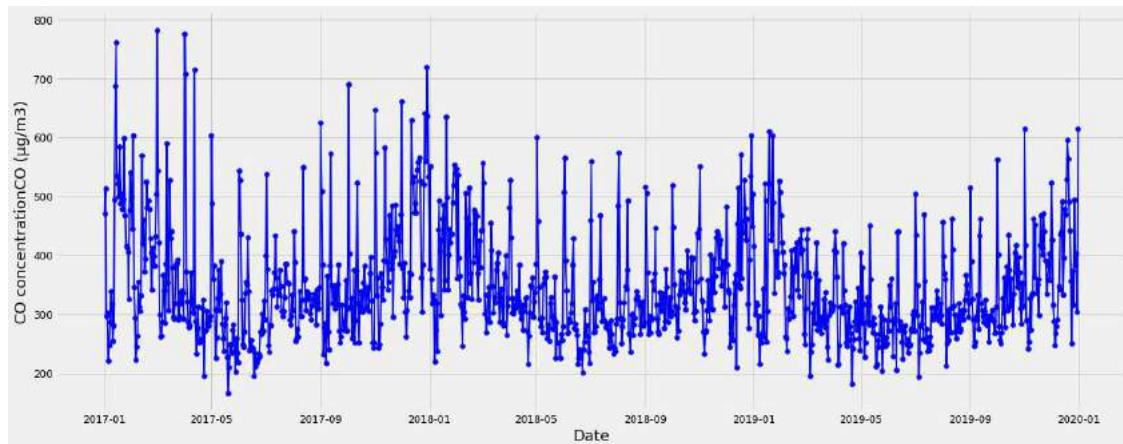
```
[98]: data.head()
```

```
[98]:          CO      date
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)>
```

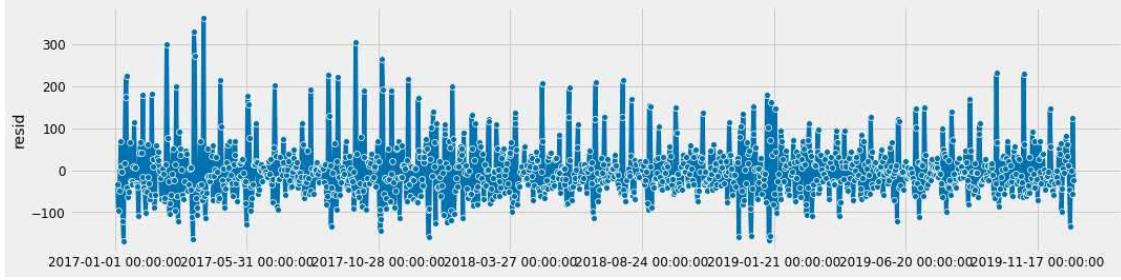
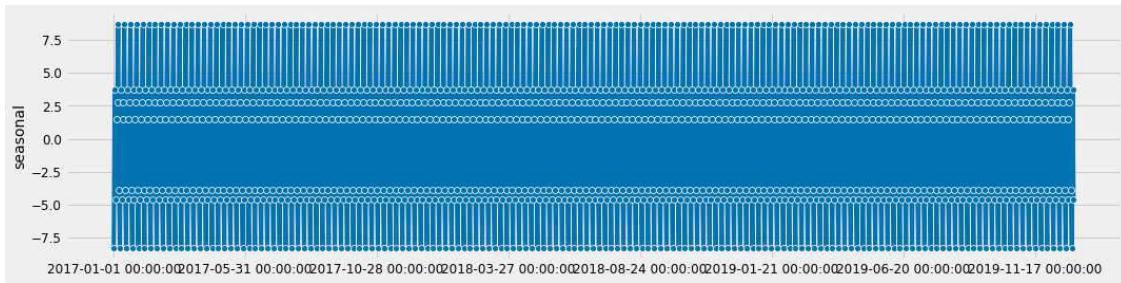
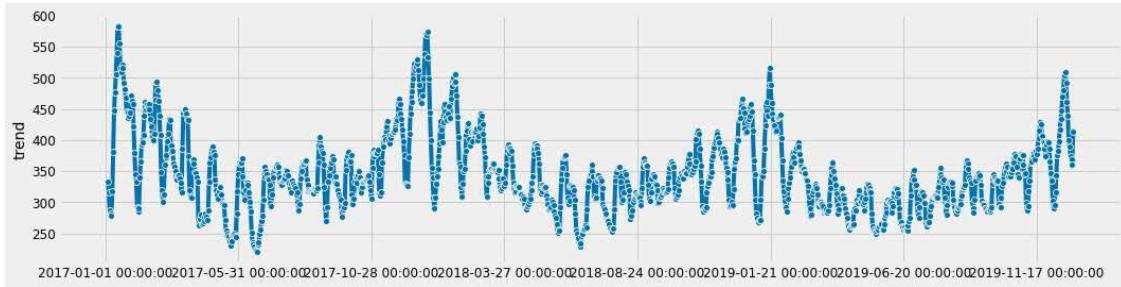


1.14 Decomposing my time series into components

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

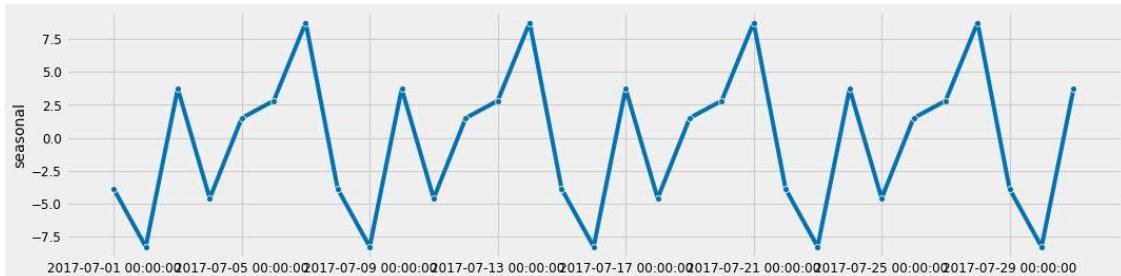
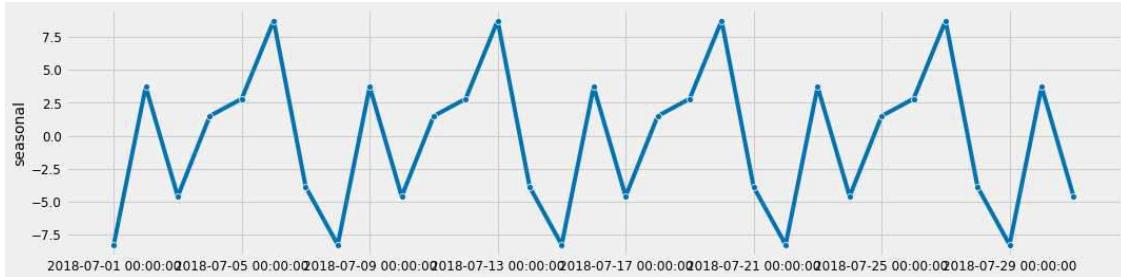
result = seasonal_decompose(data1, model='additive')
plot_series(result.trend)
plot_series(result.seasonal)
plot_series(result.resid)
```

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



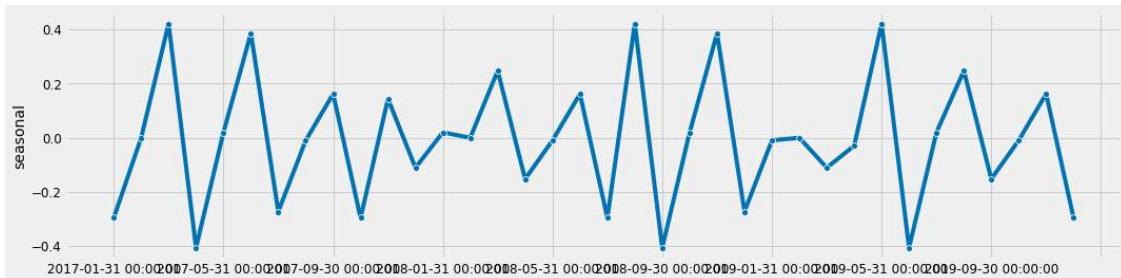
```
[102]: #Seasonality for 2017 January, we might suspect a weekly seasonality.
plot_series(result.seasonal['2018-07'])
plot_series(result.seasonal['2017-07'])
```

```
[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

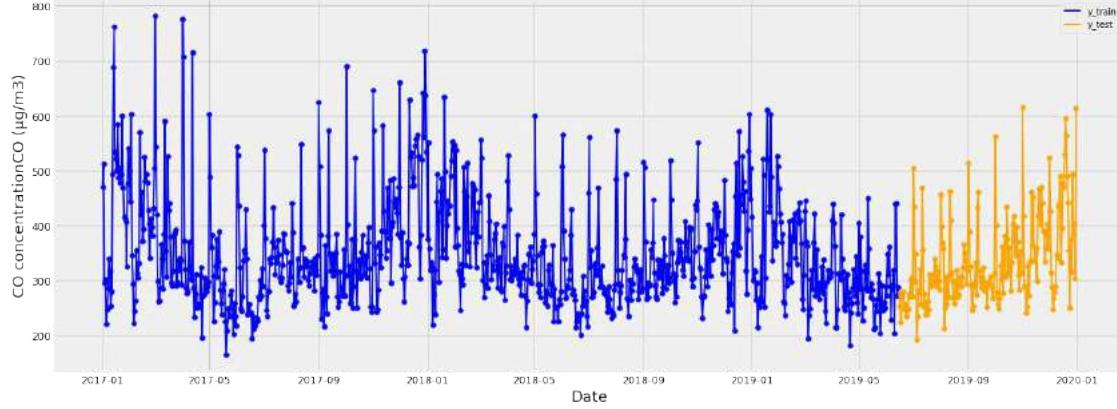
import numpy as np
import pandas as pd

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)>



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

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

```
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.

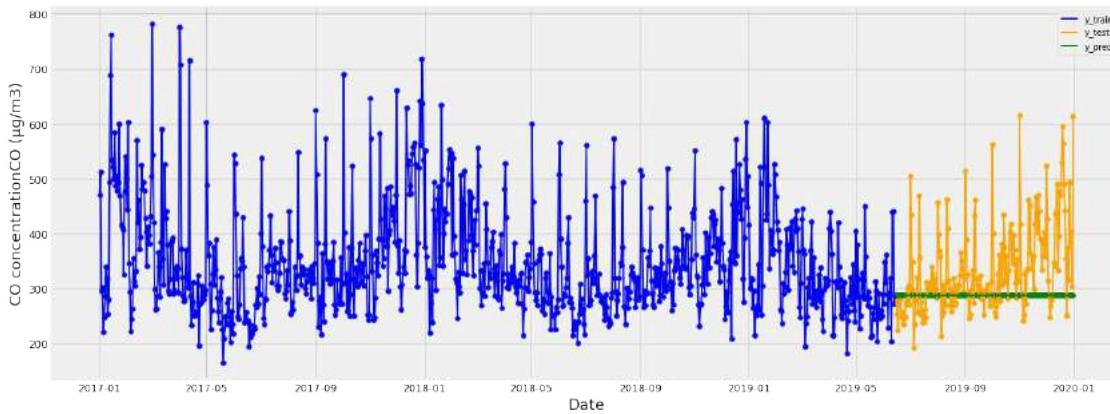
sktime comes with several forecasting algorithms (or forecasters) and tools for composite model building. All forecaster share a common interface. Forecasters are trained on a single series of data and make forecasts for the provided forecasting horizon.

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.

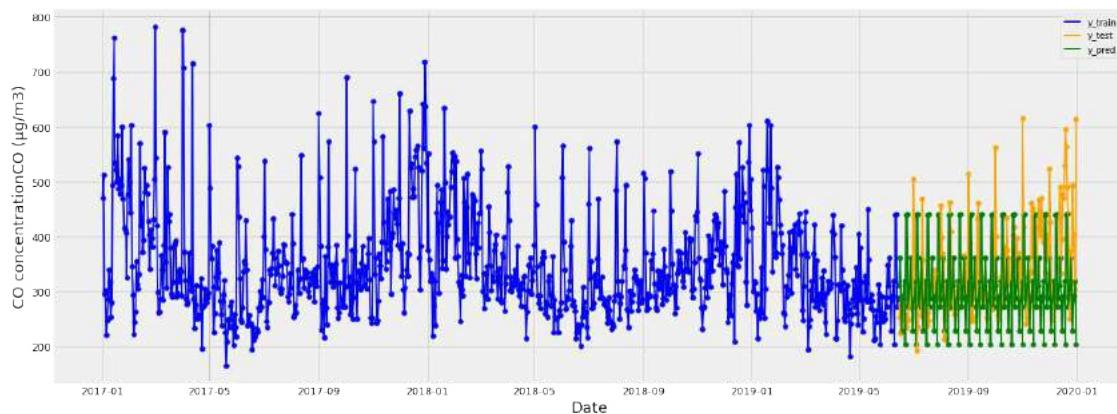
```
[107]: # using sktime
forecaster = NaiveForecaster(strategy="last")
forecaster.fit(y_train)
y_pred = forecaster.predict(fh)
fig= plt.figure(figsize=(20,8))
y_train, y_test = temporal_train_test_split(y, test_size=ts)
y_train_ind, y_test_ind = temporal_train_test_split(y.index, test_size=ts)
C0_train, C0_test = temporal_train_test_split(C0, 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,C0_train, color='blue')
plt.plot(y_test,color='orange',linewidth=2)
plt.scatter(y_test_ind,C0_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 concentrationC0 ( 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:

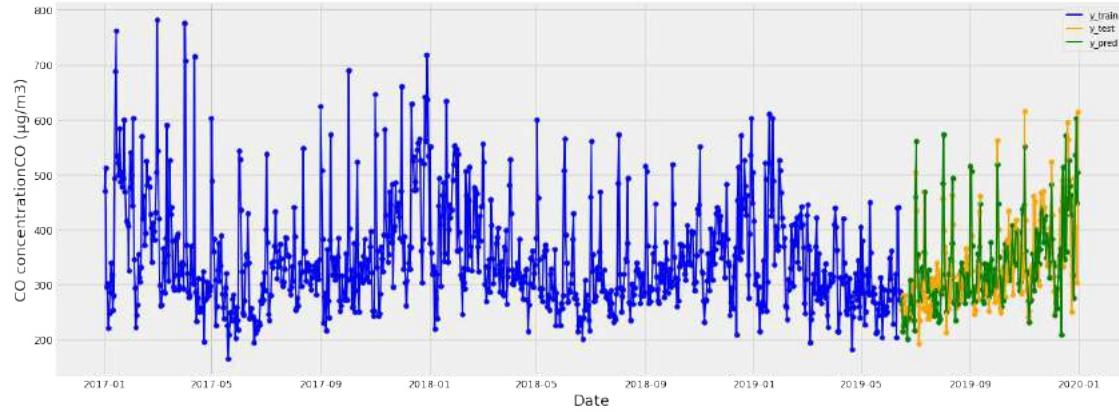
- **modular** and **compatible with scikit-learn**, so that we can easily apply any scikit-learn regressor to solve our forecasting problem,
- **tunable**, allowing us to tune hyper-parameters like the window length or strategy to generate forecasts
- **adaptive**, in the sense that it adapts the scikit-learn's 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

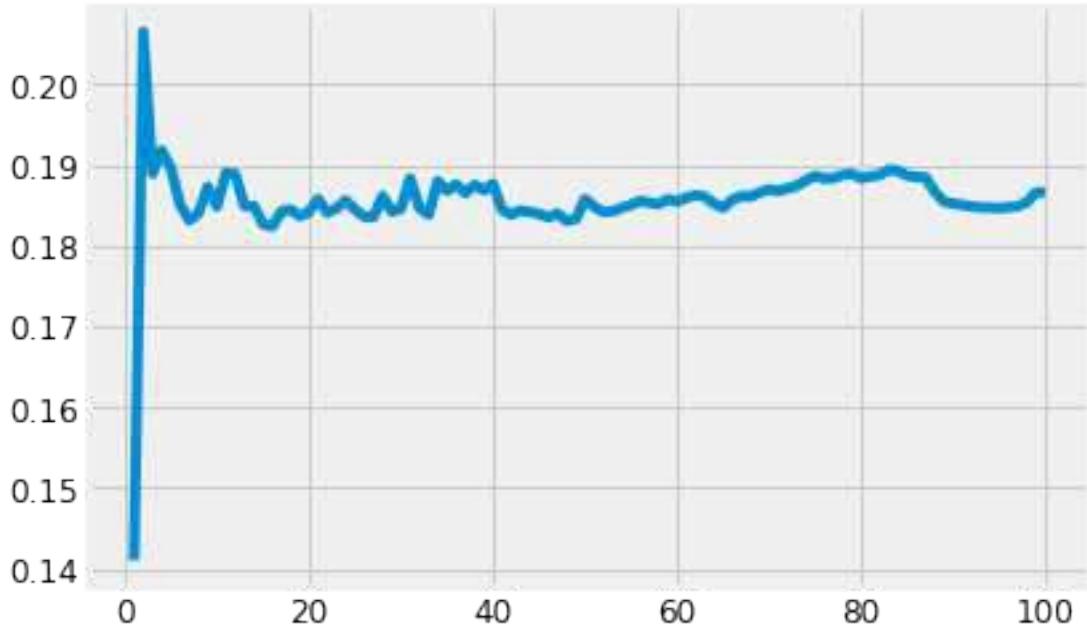
regressor = [KNeighborsRegressor(n_neighbors=k) for k in range(1,101)]
forecaster = [ReducedRegressionForecaster(
    regressor=regressor[i], window_length=12, strategy="recursive"
) for i in range(len(regressor))]
model=[forecaster[i].fit(y_train) for i in range(len(regressor))]
y_pred =[ model[i].predict(fh)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

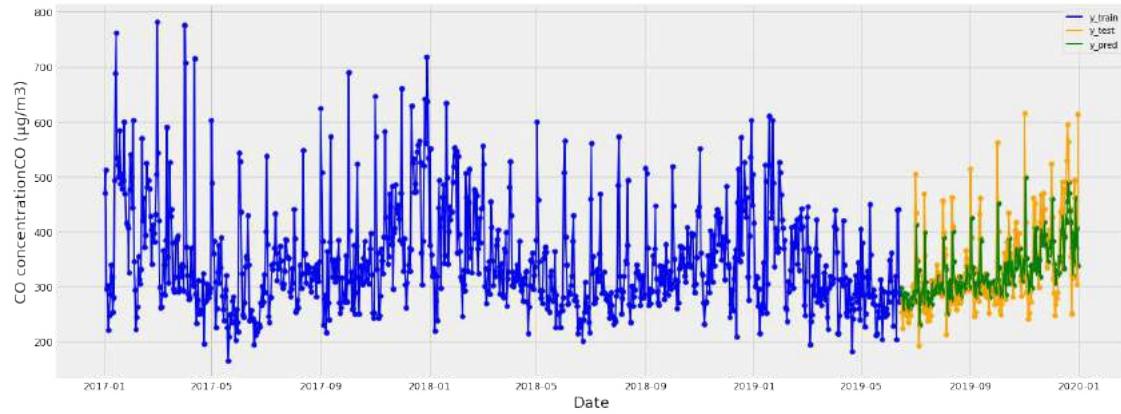
```
[112]: # ARIMA example
from statsmodels.tsa.arima.model import ARIMA
from random import random
model = ARIMA(y, order=(2, 1, 2))
model_fit = model.fit()
# make prediction
y_pred = model_fit.predict(len(y_train), len(y_train)+199, typ='levels')
fig= plt.figure(figsize=(20,8))

plt.plot(y_train,color='blue',linewidth=2)
plt.scatter(y_train_ind,C0_train, color='blue')
plt.plot(y_test,color='orange',linewidth=2)
plt.scatter(y_test_ind,C0_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 - 12-31-2019	HQIC	12572.659			
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).

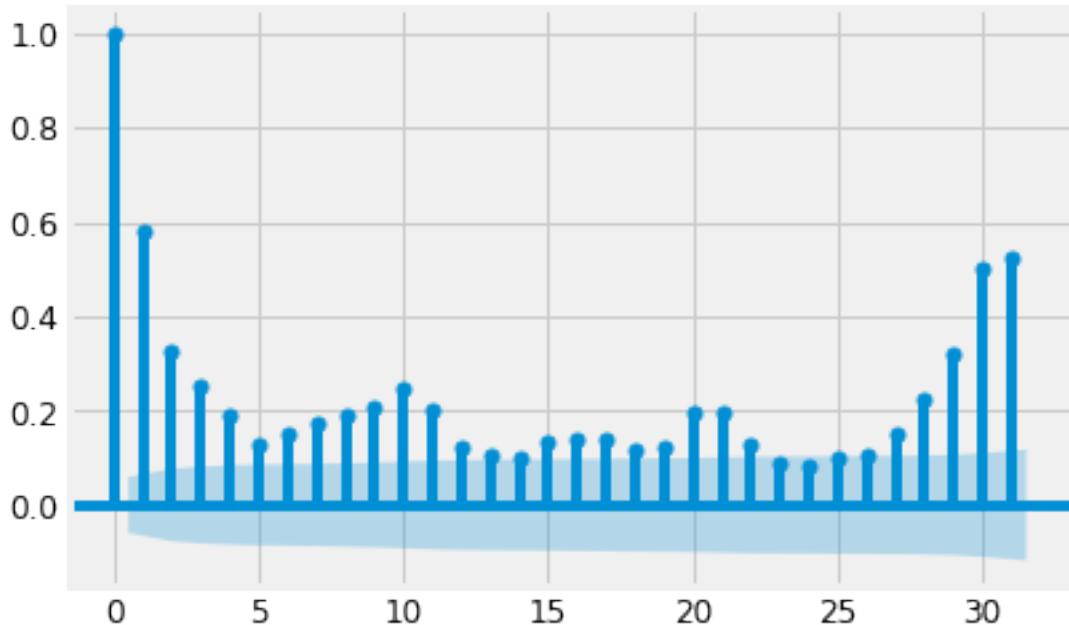
"""

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

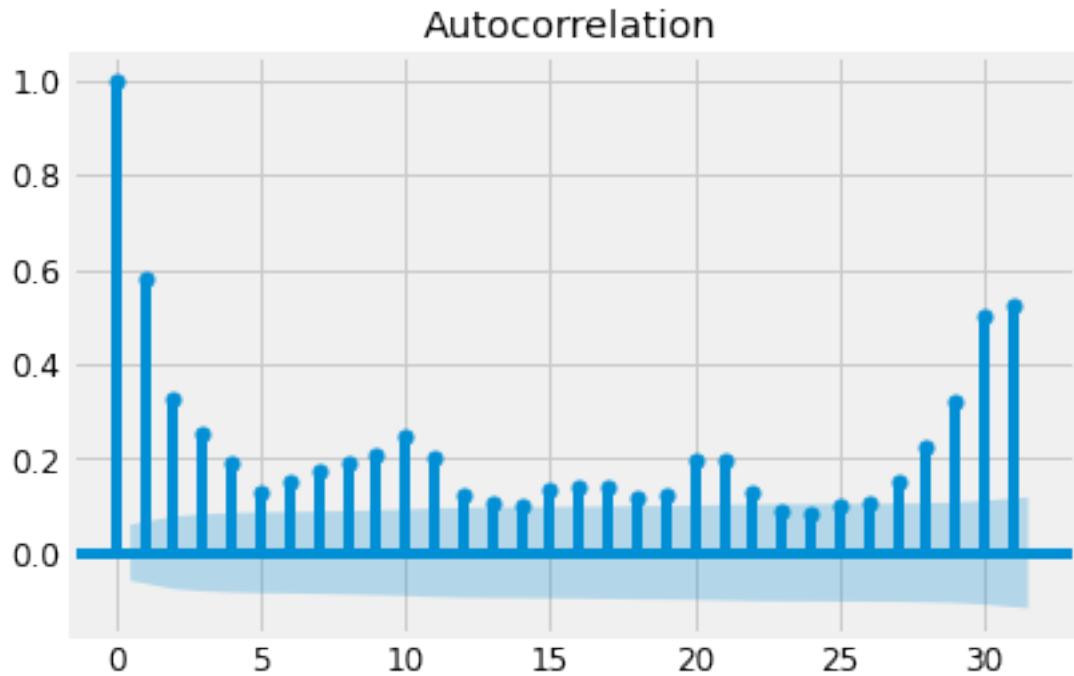
```
[114]: from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
fig= plt.figure(figsize=(20,8))
plot_acf(y)
```

[114]:

Autocorrelation



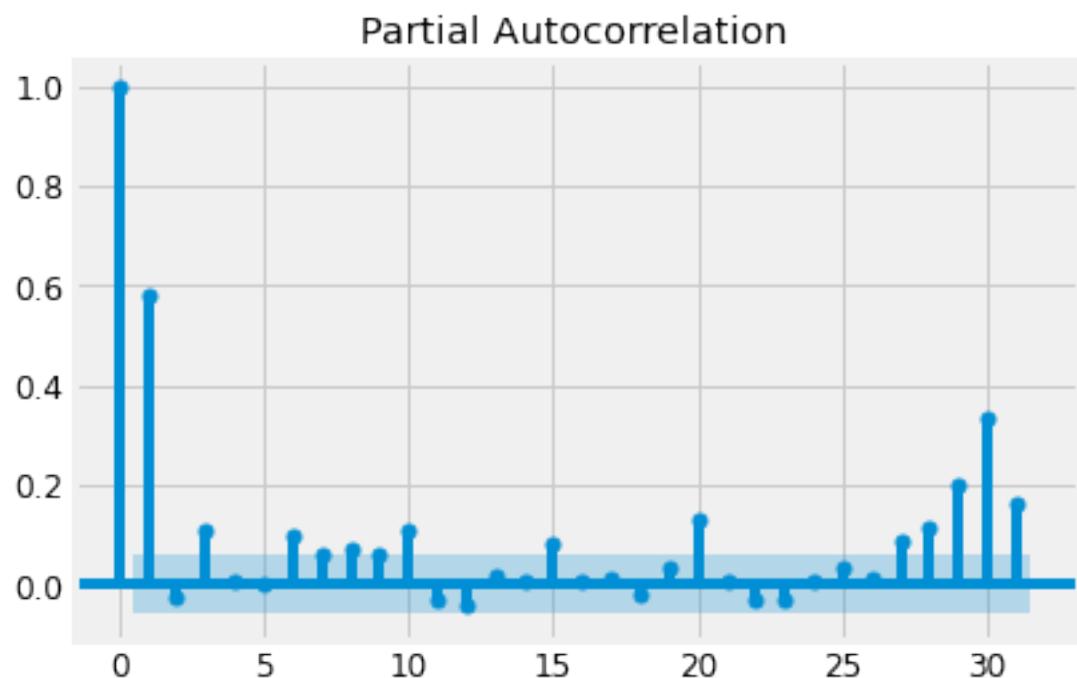
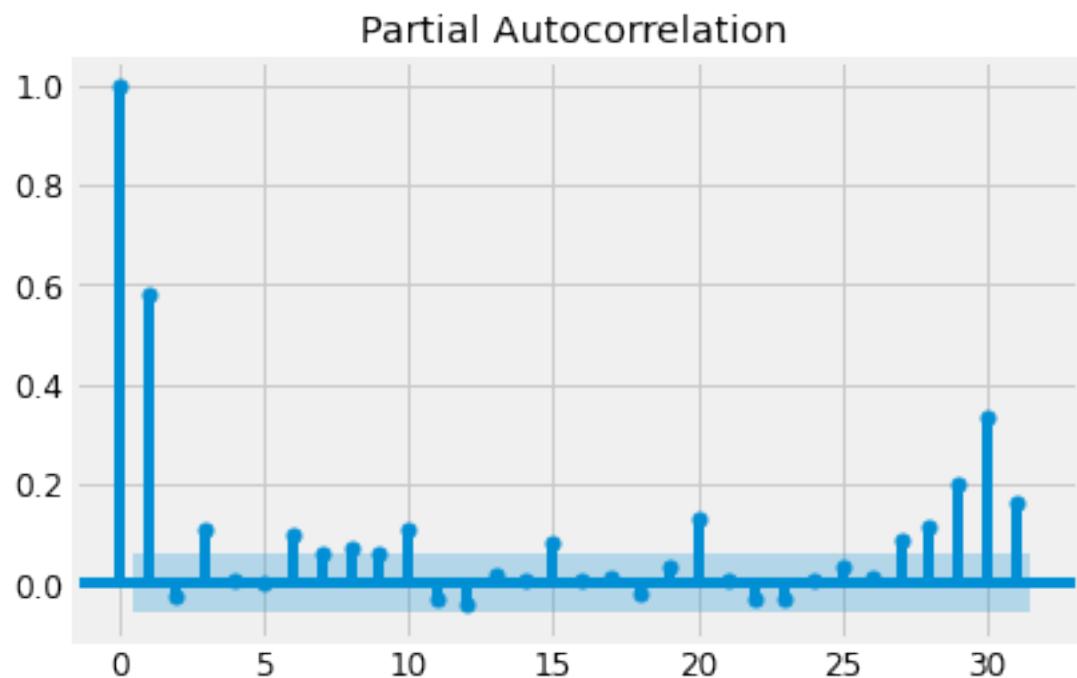
```
<Figure size 1440x576 with 0 Axes>
```



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
```

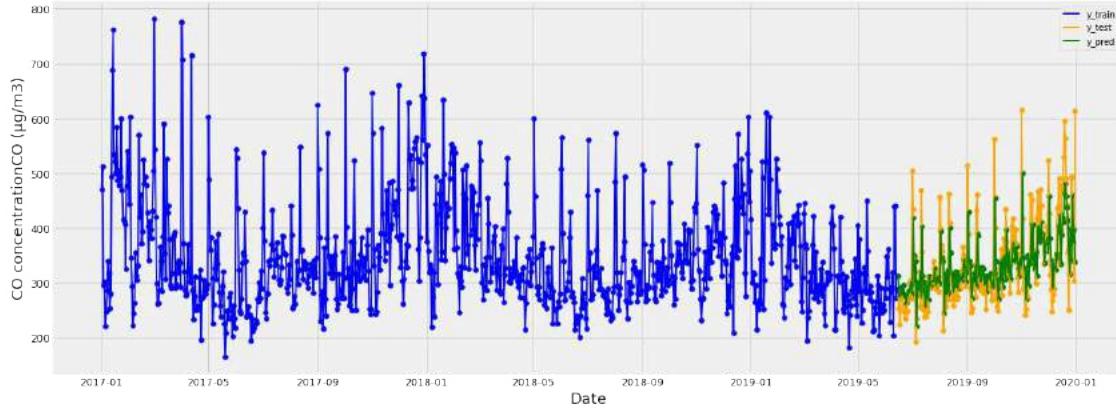
```
[117]: # Accuracy metrics
def forecast_accuracy(forecast, actual):
    mape = np.mean(np.abs(forecast - actual)/np.abs(actual)) # MAPE
    mae = np.mean(np.abs(forecast - actual)) # MAE
    mpe = np.mean((forecast - actual)/actual) # MPE
    rmse = np.mean((forecast - actual)**2)**.5 # RMSE

    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,C0_train, color='blue')
plt.plot(y_test,color='orange',linewidth=2)
plt.scatter(y_test_ind,C0_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 concentrationC0 ( 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)
```

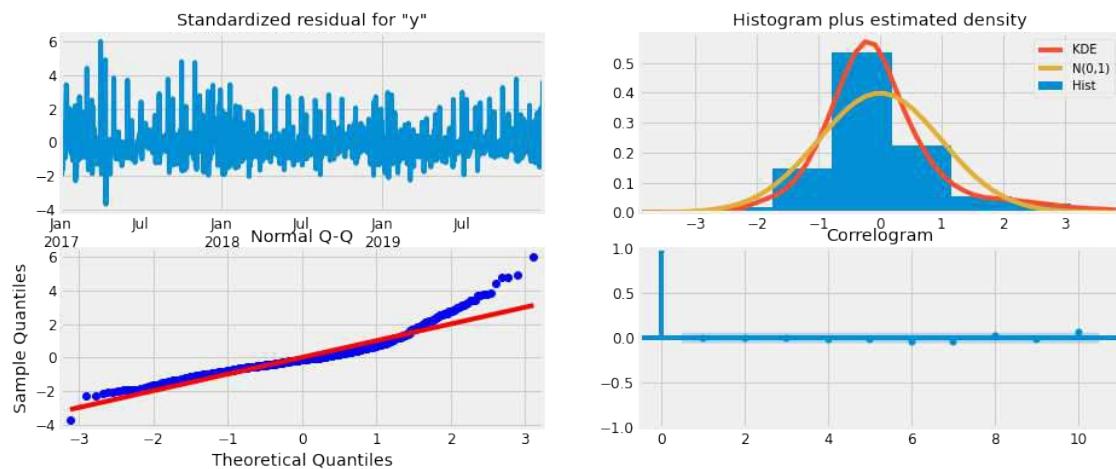


[118]: 0.13147110253474364

[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, u
    →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 u
    →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 u
        →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]: # 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)

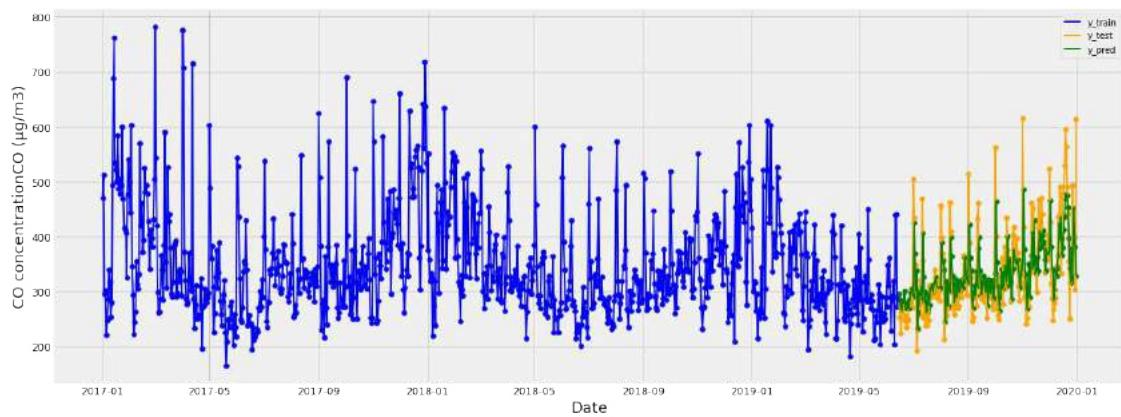
fig= plt.figure(figsize=(20,8))

plt.plot(y_train,color='blue',linewidth=2)
plt.scatter(y_train_ind,C0_train, color='blue')
```

```

plt.plot(y_test,color='orange',linewidth=2)
plt.scatter(y_test_ind,C0_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

```

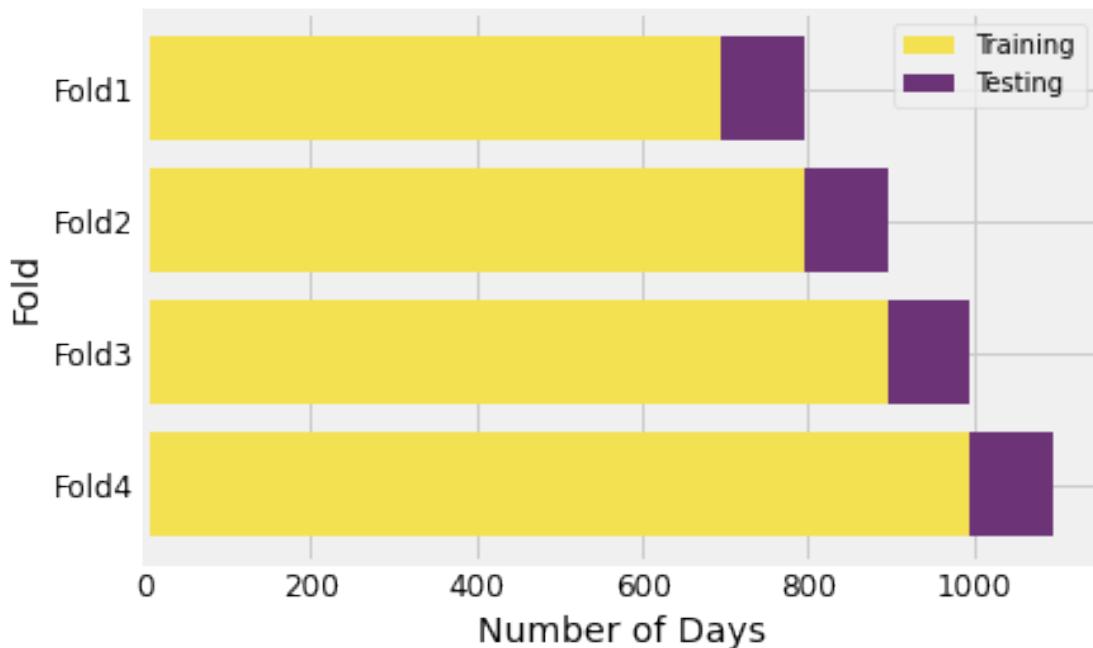
[124]: import matplotlib.pyplot as plt
Fold= ["Fold4","Fold3","Fold2","Fold1"]
Training= [995,895,795,695]
Testing= [100,100,100,100]

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')

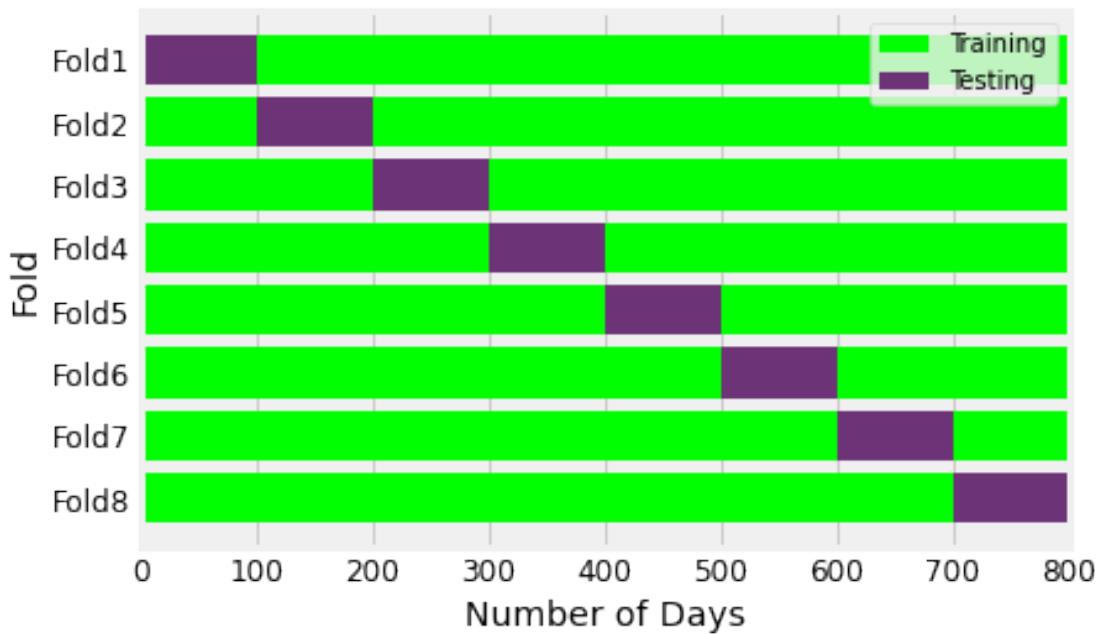
```

```
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
```

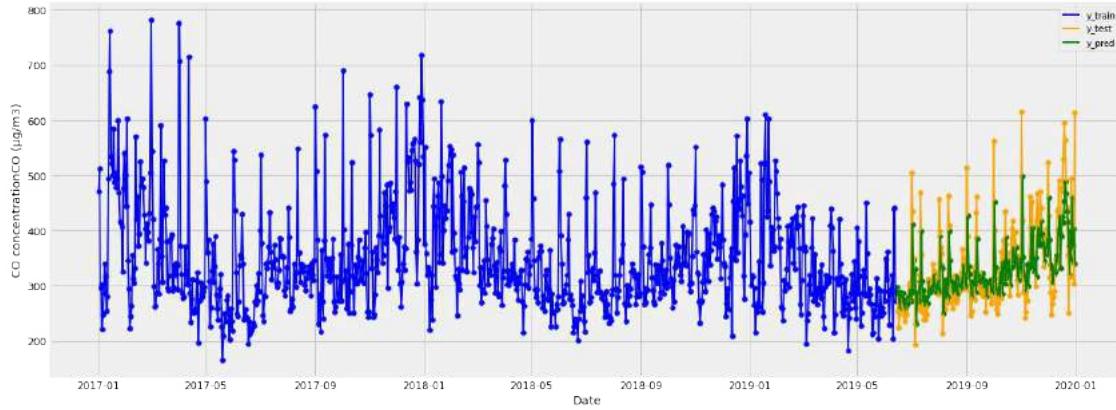
```
Freq: D, Length: 695, dtype: float64
```

```
[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]
```

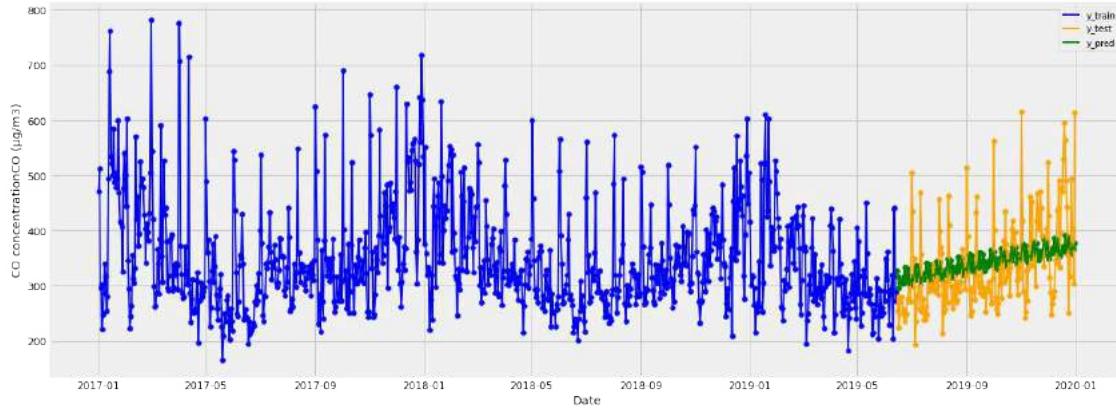
```
[129]: from statsmodels.tsa.statespace.sarimax import SARIMAX
# fit model
model = SARIMAX(y, order=(2, 0, 2), seasonal_order=(0, 0, 0, 12))
model_fit = model.fit(disp=False)
# make prediction
y_pred = model_fit.predict(len(y_train), len(y_train)+199)
fig= plt.figure(figsize=(20,8))

plt.plot(y_train,color='blue',linewidth=2)
plt.scatter(y_train_ind,C0_train, color='blue')
plt.plot(y_test,color='orange',linewidth=2)
plt.scatter(y_test_ind,C0_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 concentrationC0 ( 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

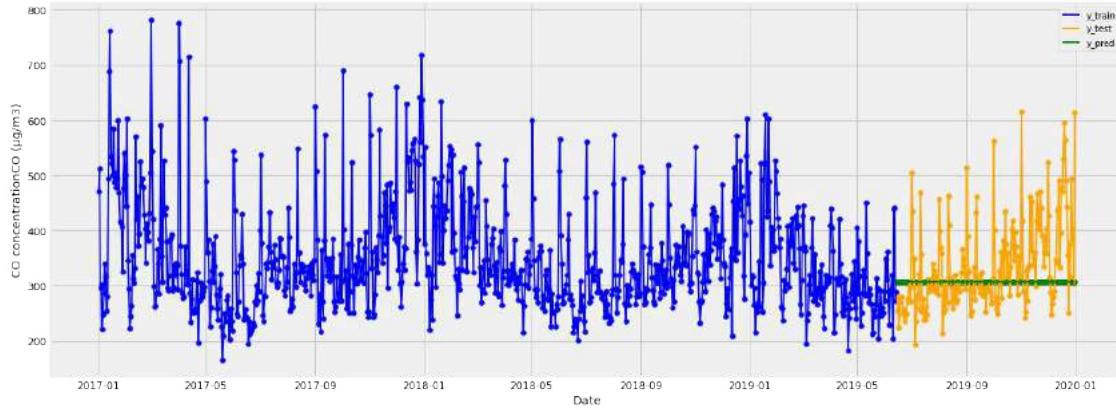
```
[130]: forecaster = ExponentialSmoothing(trend="add", seasonal='add', sp=12)
forecaster.fit(y_train)
y_pred = forecaster.predict(fh)
fig= plt.figure(figsize=(20,8))
y_train, y_test = temporal_train_test_split(y, test_size=ts)
y_train_ind, y_test_ind = temporal_train_test_split(y.index, test_size=ts)
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.

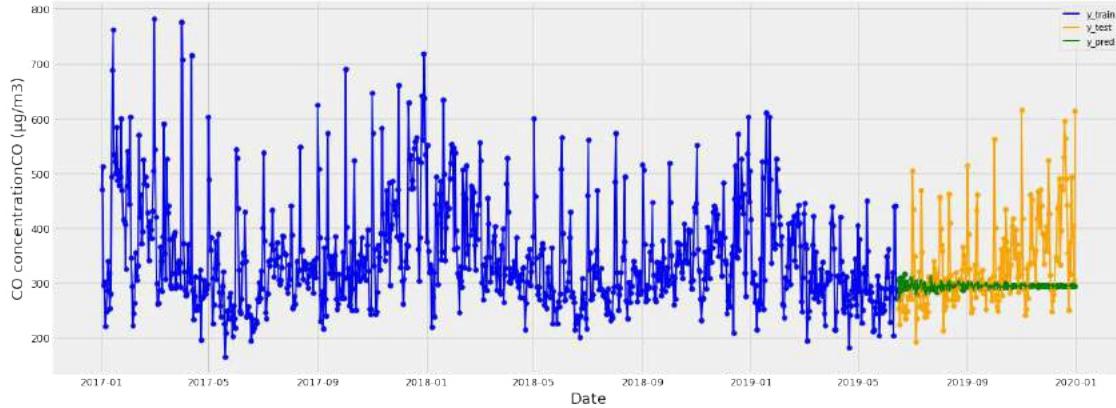
```
[131]: from sktime.forecasting.ets import AutoETS
forecaster = AutoETS(auto=False, sp=12, n_jobs=-1)
forecaster.fit(y_train)
y_pred = forecaster.predict(fh)
fig= plt.figure(figsize=(20,8))
y_train, y_test = temporal_train_test_split(y, test_size=ts)
y_train_ind, y_test_ind = temporal_train_test_split(y.index, 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)
```

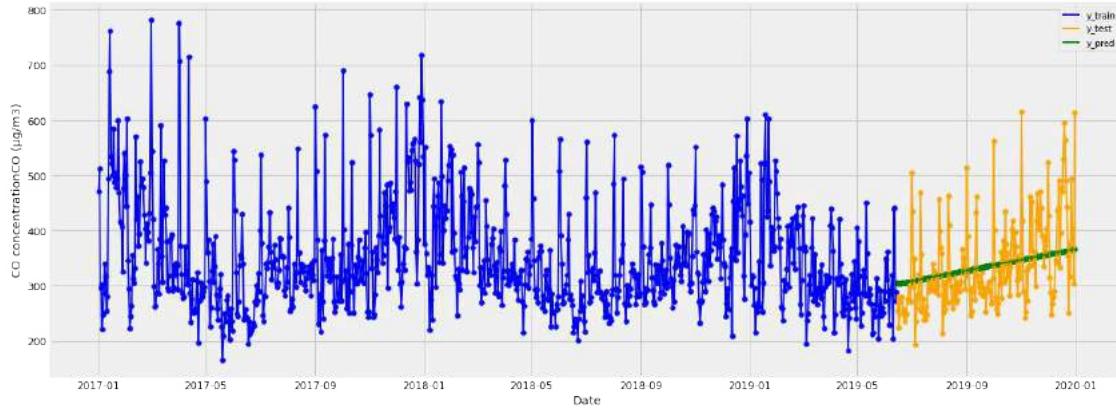


[132]: 0.1775422840877976

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.

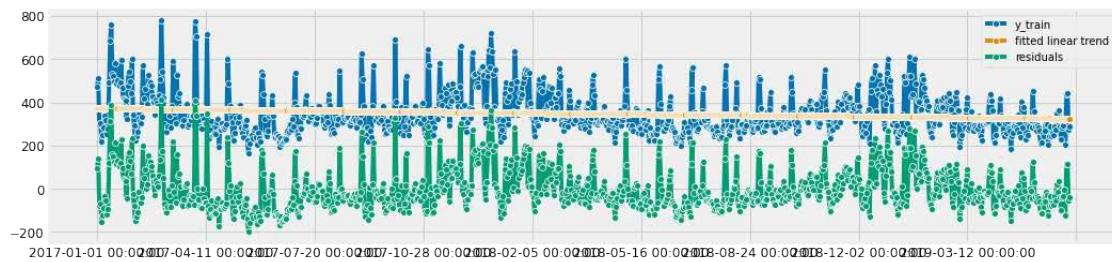
`sktime` provides a generic `Detrender`, a transformer which uses any forecaster and returns the in-sample residuals of the forecaster's predicted values. For example, to remove the linear trend of a time series, we can write:

```
[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

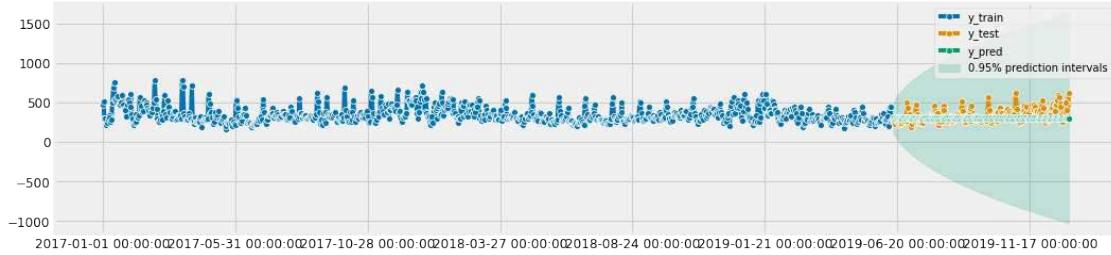
So far, we've only looked at point forecasts. In many cases, we're also interested in prediction intervals. `sktime`'s interface support prediction intervals, but we haven't implemented them for all algorithms yet.

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

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 ...

weakly   weakly_lower  weakly_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

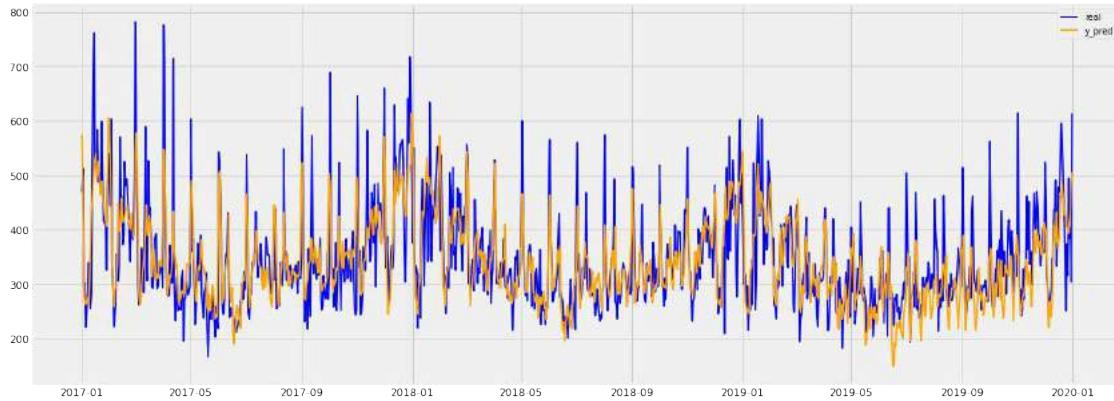
```

[5 rows x 28 columns]

```

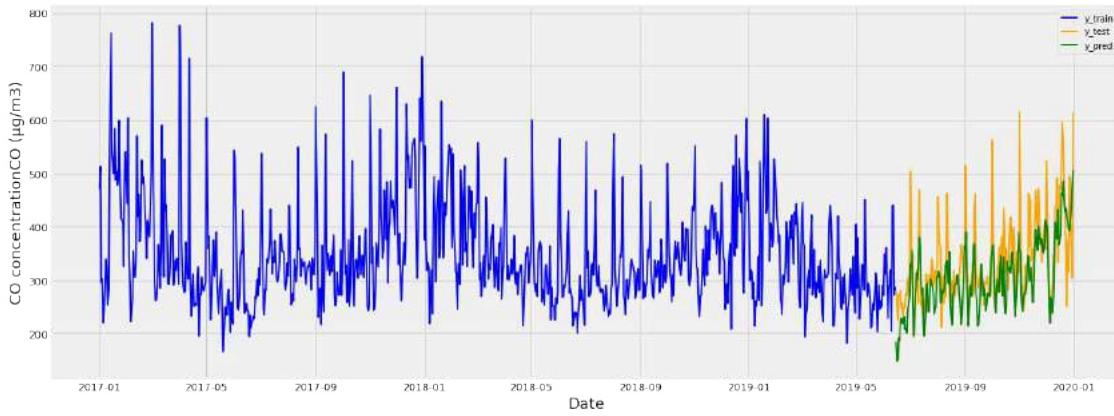
[147]: import matplotlib.pyplot as plt
fig= plt.figure(figsize=(20,8))
plt.plot(data1,color='blue',linewidth=2)
# plt.scatter(y_train_ind,CO_train, color='blue')
plt.plot(pred,color='orange',linewidth=2)
plt.legend(['real','y_pred'], loc='upper right')
# plt.xlabel('Days from Jan 9th to Nov 9th')
# plt.ylabel(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

```



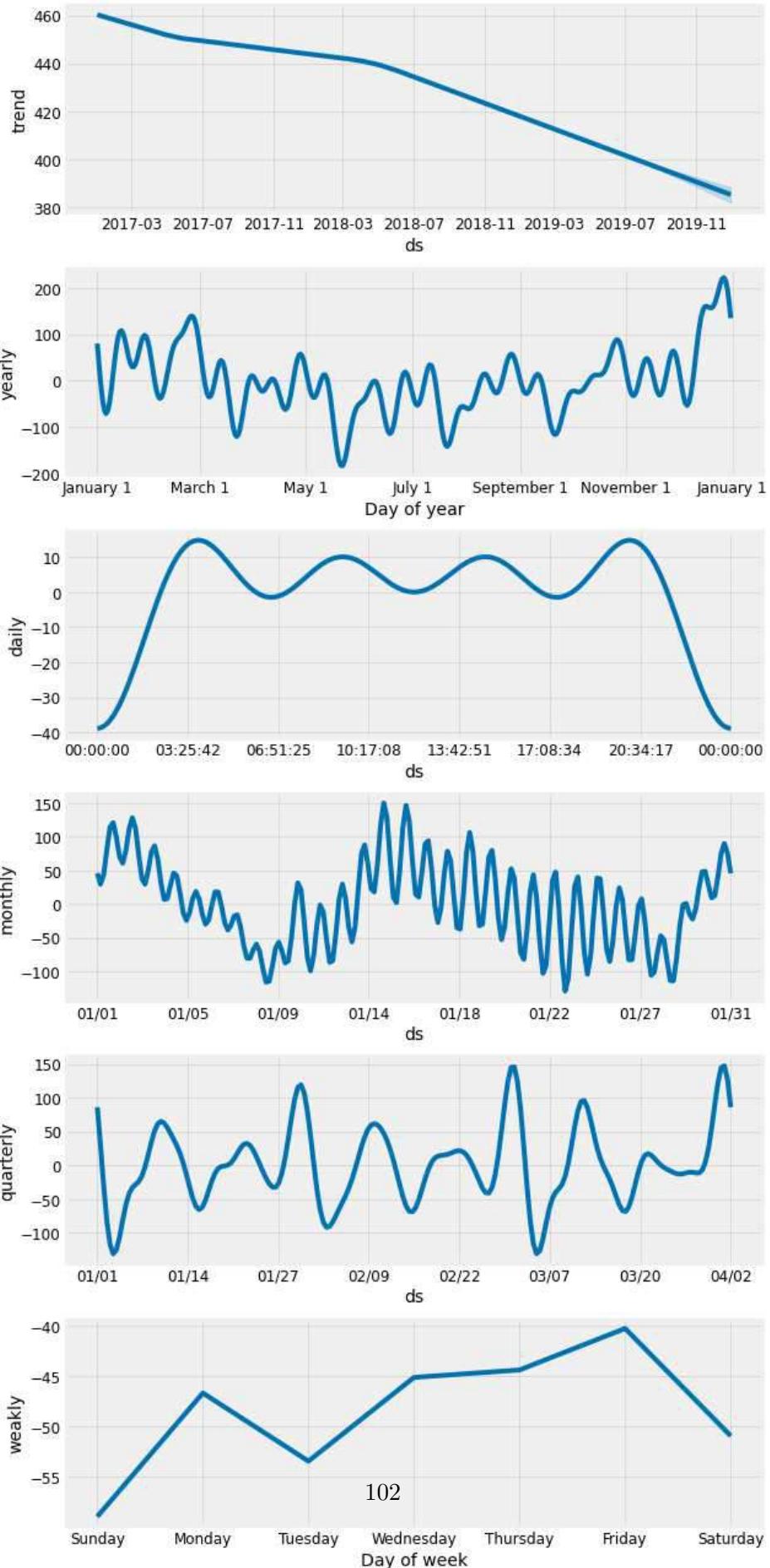
```
[148]: z_pred=forecast.iloc[895:,:]
z_pred=z_pred['yhat'].to_list()
z_pred= pd.Series(z_pred,Date_test)
```

```
[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)
```



```
[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>

	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					

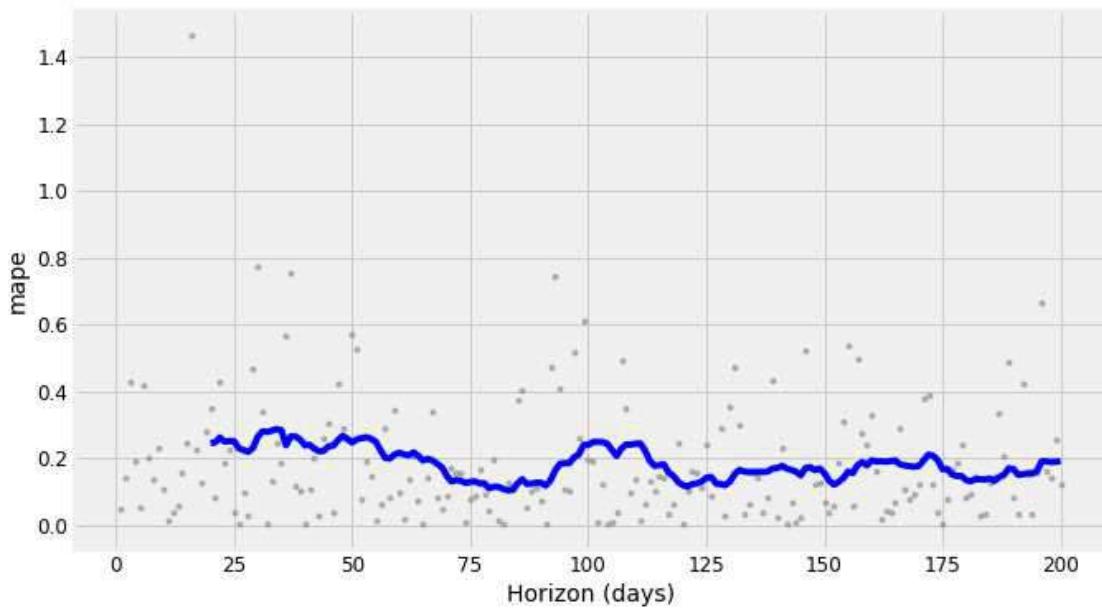
```
178      0.90
179      0.90
180      0.90
```

[181 rows x 7 columns]

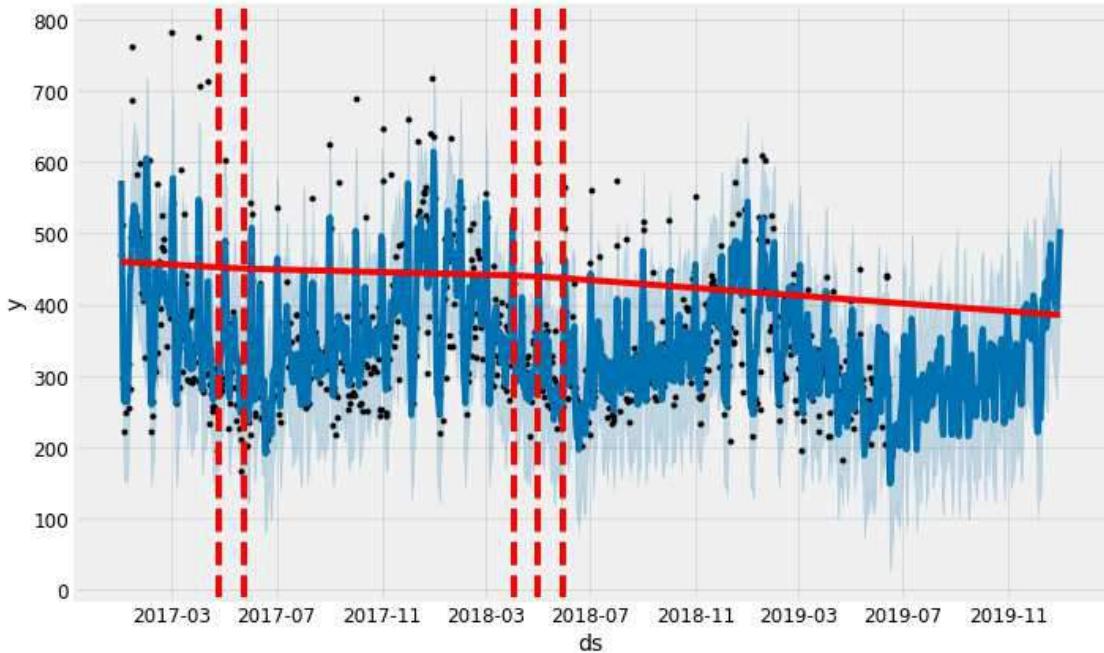
```
[152]: fb_cv.head()
```

```
[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)]
#rint(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=[(C0[i]-min(C0))/(max(C0)-min(C0)) for i in range(len(C0))]
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=[C0[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)
```

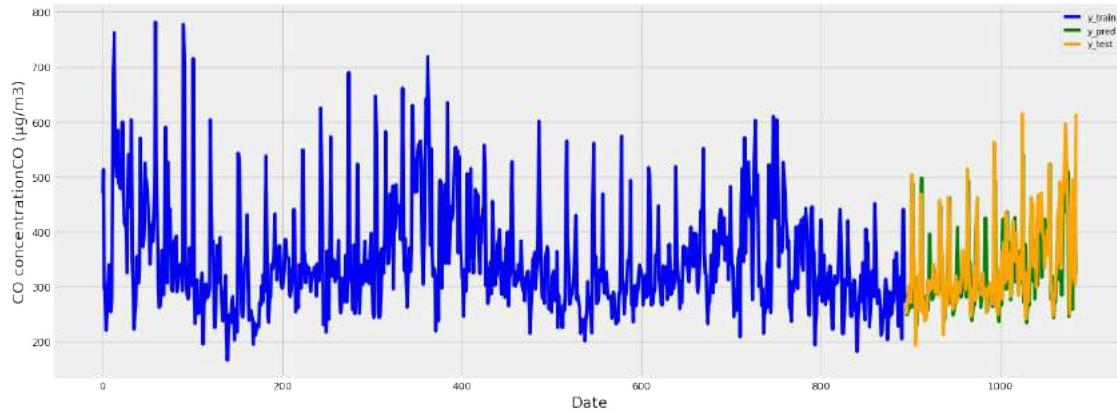
WARNING:tensorflow:Model was constructed with shape (None, 20, 1) for input KerasTensor(type_spec=TensorSpec(shape=(None, 20, 1), dtype=tf.float32, name='lstm_input'), name='lstm_input', description="created by layer 'lstm_input'"), but it was called on an input with incompatible shape (None, 10, 1).

```
[171]: y_pred=[y_pred[i]*(max(C0)-min(C0))+min(C0) 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,C0_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 concentration C0 ( 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"]的数据 = 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]

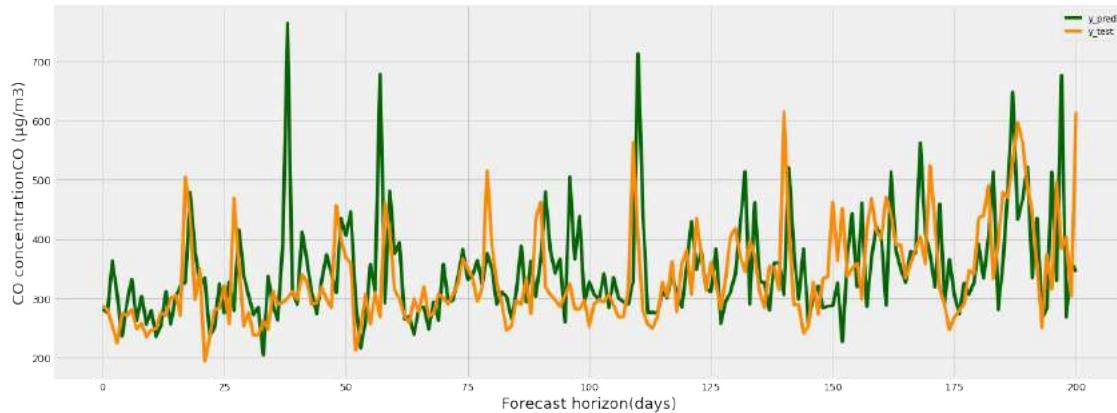
[183]: from sklearn.metrics import mean_squared_error
def validate(data,perc):
    predictions=[]
    train, test = train_test_split(data,perc)
    history=[x for x in train]
    for i in range(len(test)):
        test_X,test_y=test[i, :-1],test[i, -1]
        pred=xgb_predict(history,test_X[0])
   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

[]:

[]: