VAR Model

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')
from matplotlib.pyplot import figure


from sklearn.model_selection import TimeSeriesSplit
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler, Normalizer, Ordi

from sklearn.compose import ColumnTransformer
from sklearn.metrics import mean_squared_error


from statsmodels.tsa.stattools import adfuller
from statsmodels.regression.linear_model import OLS
```

In [2]:
```
ipe_df = pd.read_csv('data/15_min_data_HFF/IPE 15 min 2022-08-08.csv')
ipe_df
```

Out[2]:

	contTime	Turb_FNU	TurbDailyMn	TurbSamp_NTU	Chloro_RFU	ChloroDailyMn	BGA_RFU
0	2014-06-20 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN
1	2014-06-20 00:15:00	NaN	NaN	NaN	NaN	NaN	NaN
2	2014-06-20 00:30:00	NaN	NaN	NaN	NaN	NaN	NaN
3	2014-06-20 00:45:00	NaN	NaN	NaN	NaN	NaN	NaN
4	2014-06-20 01:00:00	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...
285217	2022-08-07 23:00:00	5.48	NaN	NaN	0.34	NaN	0.02
285218	2022-08-07 23:15:00	5.74	NaN	NaN	0.27	NaN	0.03
285219	2022-08-07 23:30:00	6.37	NaN	NaN	0.29	NaN	0.03
285220	2022-08-07 23:45:00	6.40	NaN	NaN	0.34	NaN	0.02
285221	2022-08-08 00:00:00	5.79	NaN	NaN	0.31	NaN	0.04

285222 rows × 33 columns

In [3]:
```
ipe_df = ipe_df.rename(columns={'contTime': 'date'})
ipe_df['date'] = pd.to_datetime(ipe_df.date)
data = ipe_df.drop(['date'], axis=1)
data.index = ipe_df.date
```

In [4]:
```
ipe_daily = data.resample('D').mean()
```

In [5]:
```
ipe_daily = ipe_daily.drop(['TurbDailyMn', 'TurbSamp_NTU',
       'ChloroDailyMn', 'BGADailyMn', 'ODODailyMn',
       'TempDailyMn', 'CondDailyMn', 'TDS_mgL',
       'TotalPres_psi', 'AshtonAirPres_psi', 'Depth_ft', 'Shift_psi',
       'Turb_FNU_raw', 'Chloro_RFU_raw', 'BGA_RFU_raw', 'ODO_mgL_raw',
       'Temp_C_raw', 'Cond_muSCm_raw', 'TDS_mgL_raw', 'TotalPres_psi_raw',
       'Turb_avdymn', 'Chlor_avdymn', 'Cyano_avdymn', 'Temp_avdymn',
       'ODO_avdymn', 'Cond_avdymn'], axis=1)
```

In [6]:
```
ipe_interp = ipe_daily.interpolate(method='spline', order=2)
```

In [7]:
```
ipe_train = ipe_interp['2016':'2020']
```

In [8]:
```
ipe_test = ipe_interp['2021']
```

```
C:\Users\harri\AppData\Local\Temp\ipykernel_20060\3605300567.py:1: FutureWarnin
g: Indexing a DataFrame with a datetimelike index using a single string to slic
e the rows, like `frame[string]`, is deprecated and will be removed in a future
version. Use `frame.loc[string]` instead.
  ipe_test = ipe_interp['2021']
```

In [9]:
```
# import island park dam hydrology data
hydro_df = pd.read_csv('data/IslandPark.TS.csv')

# set the datetime to the index
hydro_df['date'] = pd.to_datetime(hydro_df['date'])
hydro_df.set_index(['date'], inplace=True)
hydro_df.index.names = ['date']
hydro_df.columns
```

Out[9]:
```
Index(['elevation.ft', 'volume.af', 'smoothed.vol', 'smoothed.elev',
       'surfacearea.acres', 'net.evap.af', 'delta.V.af', 'regQ.cfs',
       'gain.cfs', 'smoothed.natQ.cfs'],
      dtype='object')
```

In [10]:
```
# drop redundant columns
hydro_df.drop(['volume.af', 'smoothed.vol',
                   'smoothed.elev', 'surfacearea.acres'], axis=1, inplace=True)
```

In [11]:
```
hydro_df = hydro_df['2014':'2022']
```

In [12]:
```
# set the range of the data to the same as the sonde data
hydro_interp = hydro_df.interpolate(method='spline', order=2)
```

In [13]:
```
# calculate exposed shoreline
#hydro_interp['exposed_shore'] = 8000 - hydro_interp['surfacearea.acres']
```

In [14]:
```
hydro_df_train = hydro_interp['2016':'2020']
```

In [15]:
```
hydro_df_test = hydro_interp['2021']
```

```
C:\Users\harri\AppData\Local\Temp\ipykernel_20060\1673822024.py:1: FutureWarnin
g: Indexing a DataFrame with a datetimelike index using a single string to slic
e the rows, like `frame[string]`, is deprecated and will be removed in a future
version. Use `frame.loc[string]` instead.
  hydro_df_test = hydro_interp['2021']
```

In [16]:
```
climate_df = pd.read_csv('data/Clean.Climate.TS.csv')
climate_df['Date'] = pd.to_datetime(climate_df['Date'])
climate_df.set_index(['Date'], inplace=True)
climate_df.drop(['GT.TAVE', 'GT.TMIN', 'GT.TMAX', 'GT.DP', 'GT.AP', 'GT.SWE', 'PE
       'PB.TMIN', 'PB.TMAX', 'PB.DP','HFW.AP', 'TR.SWE', 'FR.SWE', 'HF.SWE', 'HFW
       'HFW.Cum.P', 'AG.Cum.P', 'AG.Cum.ET', 'PB.AP', 'PB.SWE', 'BB.TAVE', 'BB.TM
       'BB.SWE', 'LL.TAVE', 'LL.TMIN', 'LL.TMAX', 'LL.DP', 'LL.AP', 'LL.SWE', 'GL
       'GL.TMIN', 'GL.TMAX', 'GL.DP', 'GL.AP', 'GL.SWE', 'PC.TAVE', 'PC.TMIN', 'P
       'PC.DP', 'PC.AP', 'PC.SWE', 'AL.TAVE', 'AL.TMIN', 'AL.TMAX', 'AL.DP', 'AL.
       'AS.TAVE', 'AS.TMIN', 'AS.TMAX', 'AS.DP', 'AS.AP', 'AS.ET', 'RX.TAVE',
       'RX.TMIN', 'RX.TMAX', 'RX.DP', 'RX.AP', 'RX.ET', 'TR.TAVE', 'FR.TAVE',
       'HF.TAVE', 'VA.TAVE', 'HFW.TAVE', 'TR.TMIN', 'FR.TMIN', 'HF.TMIN',
       'VA.TMIN', 'HFW.TMIN', 'TR.TMAX', 'FR.TMAX', 'HF.TMAX', 'VA.TMAX',
       'HFW.TMAX', 'TR.DP', 'FR.DP', 'HF.DP', 'VA.DP', 'HFW.DP', 'TR.AP',
       'FR.AP', 'HF.AP', 'VA.AP', 'WE.TMIN', 'WE.TMAX', 'CC.TMIN', 'CC.TMAX',
       'IP.TMIN', 'IP.TMAX'], axis=1, inplace=True)
climate_df.index.names = ['date']
climate_df.columns
```

Out[16]:
```
Index(['WE.TAVE', 'WE.DP', 'WE.AP', 'WE.SWE', 'CC.TAVE', 'CC.DP', 'CC.AP',
       'CC.SWE', 'IP.TAVE', 'IP.DP', 'IP.AP', 'IP.SWE'],
      dtype='object')
```

In [17]:
```
climate_df = climate_df['2014':'2022']
```

In [18]:
```
clim_interp = climate_df.interpolate(method='spline', order=2)
```

In [19]:
```
climate_df_train = clim_interp['2016':'2020']
```

In [20]:
```
climate_df_test = clim_interp['2021']
```

```
C:\Users\harri\AppData\Local\Temp\ipykernel_20060\3356747189.py:1: FutureWarnin
g: Indexing a DataFrame with a datetimelike index using a single string to slic
e the rows, like `frame[string]`, is deprecated and will be removed in a future
version. Use `frame.loc[string]` instead.
  climate_df_test = clim_interp['2021']
```

In [21]:
```
ipe_train
```

Out[21]:

	Turb_FNU	Chloro_RFU	BGA_RFU	ODO_mgL	Temp_C	Cond_muSCm
date						
2016-01-01	3.102660	2.183437	0.414479	10.745625	3.837323	141.426042
2016-01-02	3.067604	1.616771	0.343750	10.761146	3.853583	141.876042
2016-01-03	2.683125	1.437579	0.331771	10.710938	4.008771	143.057292
2016-01-04	2.285312	1.182526	0.215104	10.576771	4.066344	143.029167
2016-01-05	2.225938	1.189271	0.254583	10.527188	4.116604	142.786458
...	...	...	...	...	...	...
2020-12-27	4.713474	0.442796	0.638495	6.953474	3.495432	138.068421
2020-12-28	4.657604	0.481429	0.708132	6.885625	3.478885	137.610417
2020-12-29	4.382604	0.449896	0.640842	6.684062	3.506271	137.482292
2020-12-30	4.088478	0.421979	0.603958	6.691354	3.560729	137.288542
2020-12-31	4.057789	0.368000	0.578526	6.784105	3.595811	137.181053

1827 rows × 6 columns

In [22]:
```
final_train = pd.merge(climate_df_train, hydro_df_train, on=['date'])
final_train = pd.merge(final_train, ipe_train, on=['date'])
```

In [23]:
```
final_test = pd.merge(climate_df_test, hydro_df_test, on=['date'])
final_test = pd.merge(final_test, ipe_test, on=['date'])
```

In [24]:
```
final_test
```

Out[24]:

date	WE.TAVE	WE.DP	WE.AP	WE.SWE	CC.TAVE	CC.DP	CC.AP	CC.SWE	IP.TAVE	IP.DP
2021-01-01	16.0	0.0	8.2	7.3	19.0	0.0	4.0	3.6	23.0	0.0
2021-01-02	19.0	0.2	8.4	7.6	20.0	0.3	4.3	3.9	24.0	0.3
2021-01-03	22.0	0.4	8.8	7.9	24.0	0.1	4.4	4.0	27.0	0.3
2021-01-04	25.0	0.7	9.5	8.6	26.0	0.4	4.8	4.5	30.0	0.3
2021-01-05	23.0	0.6	10.1	9.0	25.0	0.2	5.0	4.7	29.0	0.4
...	...	...	...	...	...	...	...	...	...	...
2021-	10.0	0.5	17.3	12.9	13.0	0.4	11.5	6.5	17.0	0.6

In [25]:
```
jimmies = OLS(exog=final_train.drop(['Turb_FNU'], axis=1), endog=final_train['Tur
jim = jimmies.fit()
print(jim.summary())
```

```
                                 OLS Regression Results
================================================================================
========
Dep. Variable:                Turb_FNU   R-squared (uncentered):
0.788
Model:                             OLS   Adj. R-squared (uncentered):
0.785
Method:                  Least Squares   F-statistic:
291.6
Date:                 Mon, 22 Aug 2022   Prob (F-statistic):
0.00
Time:                         13:52:29   Log-Likelihood:
-3552.1
No. Observations:                 1827   AIC:
7150.
Df Residuals:                     1804   BIC:
7277.
Df Model:                           23
Covariance Type:             nonrobust
================================================================================
======
                       coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
------
WE.TAVE             -0.0659      0.030     -2.166      0.030      -0.126
-0.006
WE.DP                0.0140      0.236      0.059      0.953      -0.449
0.477
WE.AP               -0.3370      0.025    -13.249      0.000      -0.387
-0.287
WE.SWE               0.0008      0.014      0.058      0.953      -0.026
0.027
CC.TAVE              0.1215      0.036      3.363      0.001       0.051
0.192
CC.DP                0.1194      0.298      0.401      0.688      -0.464
0.703
CC.AP               -0.0702      0.035     -2.015      0.044      -0.139
-0.002
CC.SWE               0.1900      0.037      5.069      0.000       0.116
0.263
IP.TAVE             -0.0359      0.012     -2.874      0.004      -0.060
-0.011
IP.DP               -0.5790      0.557     -1.040      0.299      -1.671
0.513
IP.AP                0.5817      0.061      9.592      0.000       0.463
0.701
IP.SWE              -0.2209      0.035     -6.325      0.000      -0.289
-0.152
elevation.ft         0.0006      0.000      5.140      0.000       0.000
0.001
net.evap.af         -0.0003      0.001     -0.336      0.737      -0.002
0.002
```

```
delta.V.af             0.0011      0.001      1.819      0.069   -8.45e-05
0.002
regQ.cfs            8.986e-05      0.001      0.078      0.937      -0.002
0.002
gain.cfs               0.0001      0.001      0.090      0.928      -0.002
0.003
smoothed.natQ.cfs     -0.0002      0.001     -0.238      0.812      -0.002
0.001
Chloro_RFU            -0.0238      0.054     -0.440      0.660      -0.130
0.082
BGA_RFU                0.1184      0.157      0.755      0.450      -0.189
0.426
ODO_mgL               -0.2957      0.044     -6.740      0.000      -0.382
-0.210
Temp_C                 0.0380      0.032      1.203      0.229      -0.024
0.100
Cond_muSCm             0.0085      0.004      2.310      0.021       0.001
0.016
==============================================================================
Omnibus:                     1377.636   Durbin-Watson:                   0.298
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            36089.692
Skew:                           3.286   Prob(JB):                         0.00
Kurtosis:                      23.758   Cond. No.                     9.11e+04
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not cont
ain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctl
y specified.
[3] The condition number is large, 9.11e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

This R2 value shows that the variance in the turbidity is about 79% captured by the explanatory variables according to this model. This does not necessarily mean our model can't get any better than this, but it may represent a loose ballpark for the upper bounds of how accurately we can predict the turbidity using basic methods. Additionally, multicollinearity is high here, which is to be expected. This can be mitigated by eliminating columns or using Principal Component Analysis depending on how we decide to move forward.

In [26]:
```
# determine the lowest negative value so we can add that to all values to make th
final_train.min()
```

Out[26]:
```
WE.TAVE                 -5.000000
WE.DP                    0.000000
WE.AP                    0.000000
WE.SWE                   0.000000
CC.TAVE                 -4.000000
CC.DP                    0.000000
CC.AP                    0.000000
CC.SWE                   0.000000
IP.TAVE                 -8.000000
IP.DP                    0.000000
IP.AP                    0.000000
IP.SWE                   0.000000
elevation.ft          6276.840000
net.evap.af          -1486.560444
delta.V.af           -2171.061429
regQ.cfs                71.900000
gain.cfs               -64.733023
smoothed.natQ.cfs      295.220945
Turb_FNU                -0.118526
Chloro_RFU              -0.332188
BGA_RFU                 -0.453542
ODO_mgL                  3.373684
Temp_C                   2.639990
Cond_muSCm              71.887474
dtype: float64
```

In [27]:
```
final_train_log = np.log((final_train+2172))
final_test_log = np.log((final_test+2172))
```

In [28]:
```
#missing value treatment
cols = final_train.columns
#checking stationarity
from statsmodels.tsa.vector_ar.vecm import coint_johansen
#since the test works for only 12 variables, I have randomly dropped
#in the next iteration, I would drop another and check the eigenvalues
johan_test_temp = final_train.drop(['Turb_FNU'], axis=1)
coint_johansen(johan_test_temp,-1,1).eig
```

```
C:\Users\harri\anaconda3\lib\site-packages\statsmodels\tsa\vector_ar\vecm.py:64
8: HypothesisTestWarning: Critical values are only available for time series wi
th 12 variables at most.
  warnings.warn(
```

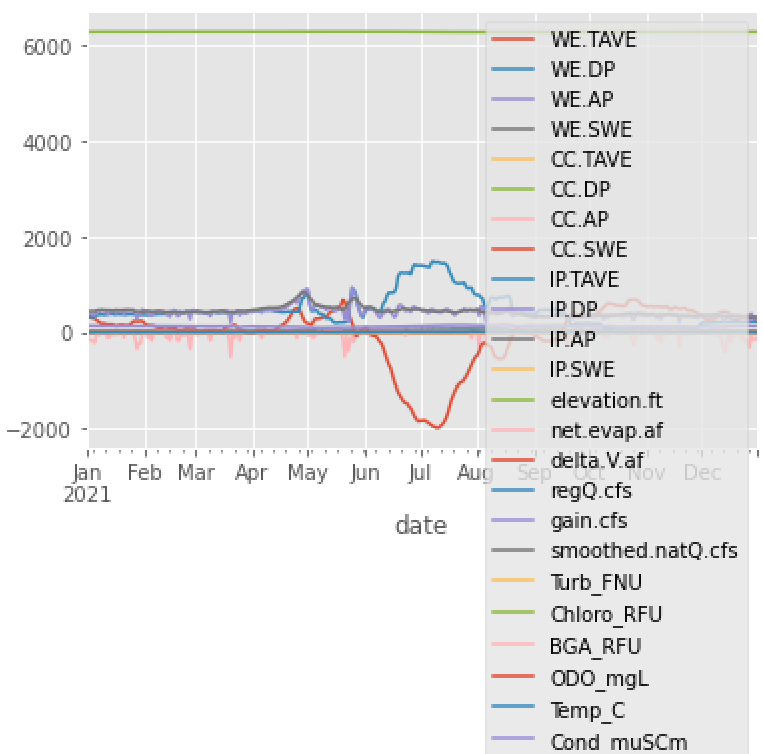
Out[28]:
```
array([4.99705672e-01, 4.80664449e-01, 4.67499562e-01, 3.93824494e-01,
       3.50359999e-01, 3.16128057e-01, 2.79566805e-01, 2.38908795e-01,
       1.86654581e-01, 1.41297090e-01, 1.27468499e-01, 1.15509695e-01,
       1.09540637e-01, 6.24503493e-02, 5.38383009e-02, 3.86332068e-02,
       3.40533136e-02, 2.66377505e-02, 2.49116464e-02, 1.87726605e-02,
       7.94646823e-03, 5.38922582e-03, 6.91992490e-07])
```

In [29]:
```
final_test.isna().sum()
```

Out[29]:
```
WE.TAVE             0
WE.DP               0
WE.AP               0
WE.SWE              0
CC.TAVE             0
CC.DP               0
CC.AP               0
CC.SWE              0
IP.TAVE             0
IP.DP               0
IP.AP               0
IP.SWE              0
elevation.ft        0
net.evap.af         0
delta.V.af          0
regQ.cfs            0
gain.cfs            0
smoothed.natQ.cfs   0
Turb_FNU            0
Chloro_RFU          0
BGA_RFU             0
ODO_mgL             0
Temp_C              0
Cond_muSCm          0
dtype: int64
```

In [30]:
```
final_test.plot()
```

Out[30]:
```
<AxesSubplot:xlabel='date'>
```



In [31]:
```
#creating the train and validation set
train = final_train_log.dropna()
valid = final_test_log.dropna()

#fit the model
from statsmodels.tsa.vector_ar.var_model import VAR

model = VAR(endog=train)
model_fit = model.fit(365)

# make prediction on validation
prediction = model_fit.forecast(model_fit.endog, steps=len(valid))
```

```
C:\Users\harri\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:47
1: ValueWarning: No frequency information was provided, so inferred frequency D
will be used.
  self._init_dates(dates, freq)
```

In [32]:
```
# #converting predictions to dataframe
# pred = pd.DataFrame(index=range(0,len(prediction)),columns=[cols])
# for j in range(0,6):
#     for i in range(0, len(prediction)):
#        pred.iloc[i][j] = prediction[i][j]

# #check rmse
# for i in cols:
#     print('rmse value for', i, 'is : ', np.sqrt(mean_squared_error(pred[i], val
```

In [33]:
```
#make final predictions
model = VAR(endog=train.dropna())
model_fit = model.fit(365)
yhat = model_fit.forecast(model_fit.endog, steps=365)
print(yhat)
```

```
C:\Users\harri\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:47
1: ValueWarning: No frequency information was provided, so inferred frequency D
will be used.
  self._init_dates(dates, freq)

[[7.69415424 7.68282346 7.68852301 ... 7.68693467 7.68500386 7.7427986 ]
 [7.69356783 7.68310111 7.68561762 ... 7.68686827 7.68525655 7.7442404 ]
 [7.69611082 7.68305348 7.68645388 ... 7.68704759 7.6854341  7.746909  ]
 ...
 [7.68510894 7.68326215 7.68584053 ... 7.68755501 7.68452014 7.73519853]
 [7.70032479 7.68330207 7.68621462 ... 7.68702958 7.68441796 7.7381127 ]
 [7.70621816 7.68332365 7.68831199 ... 7.68630425 7.68419827 7.74284427]]
```

In [34]:
```
model_fit.endog_lagged
```

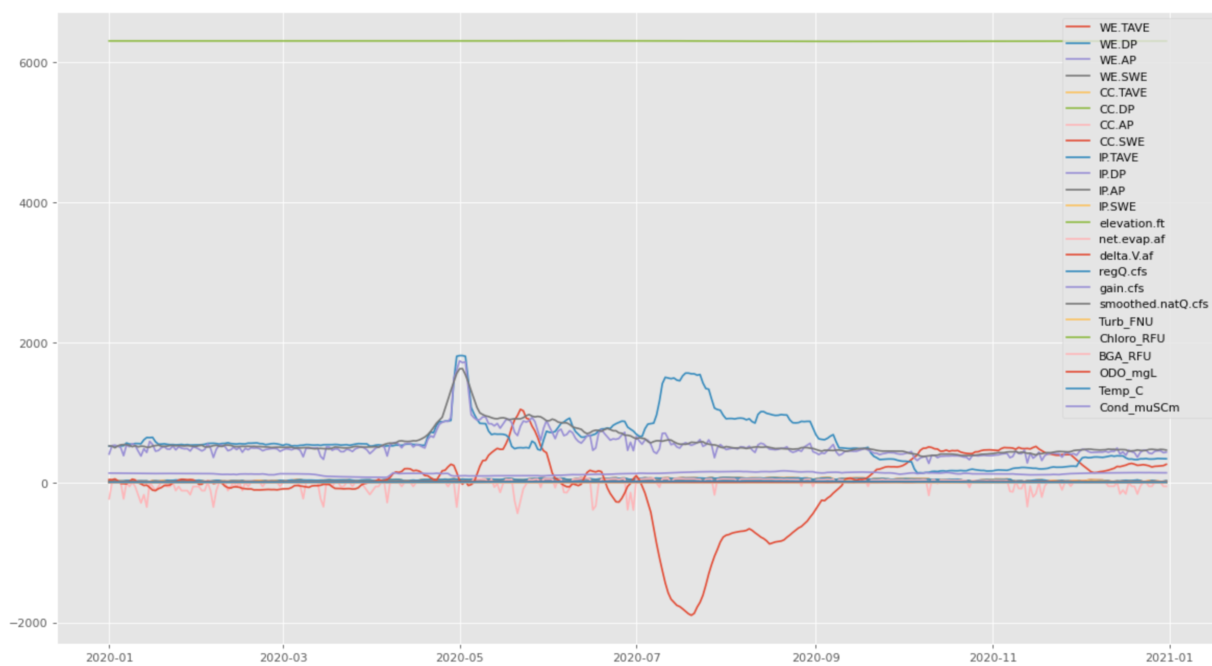
Out[34]:
```
array([[1.        , 7.69256965, 7.68340368, ..., 7.68833882, 7.68516885,
        7.74648484],
       [1.        , 7.68753877, 7.68340368, ..., 7.68834594, 7.68517632,
        7.74667934],
       [1.        , 7.68845536, 7.68340368, ..., 7.68832293, 7.68524764,
        7.74718971],
       ...,
       [1.        , 7.6912001 , 7.68340368, ..., 7.68694123, 7.68548697,
        7.74267656],
       [1.        , 7.68937111, 7.68340368, ..., 7.68680274, 7.68549509,
        7.74278172],
       [1.        , 7.68891334, 7.68349576, ..., 7.68674954, 7.68550768,
        7.74276771]])
```

In [35]:
```
figure(figsize=(18,10), dpi=80)

plt.plot(final_train['2020'])
plt.legend(final_train)
```

```
C:\Users\harri\AppData\Local\Temp\ipykernel_20060\1703906445.py:3: FutureWarnin
g: Indexing a DataFrame with a datetimelike index using a single string to slic
e the rows, like `frame[string]`, is deprecated and will be removed in a future
version. Use `frame.loc[string]` instead.
  plt.plot(final_train['2020'])
```

Out[35]:
```
<matplotlib.legend.Legend at 0x1d9e1ad2490>
```



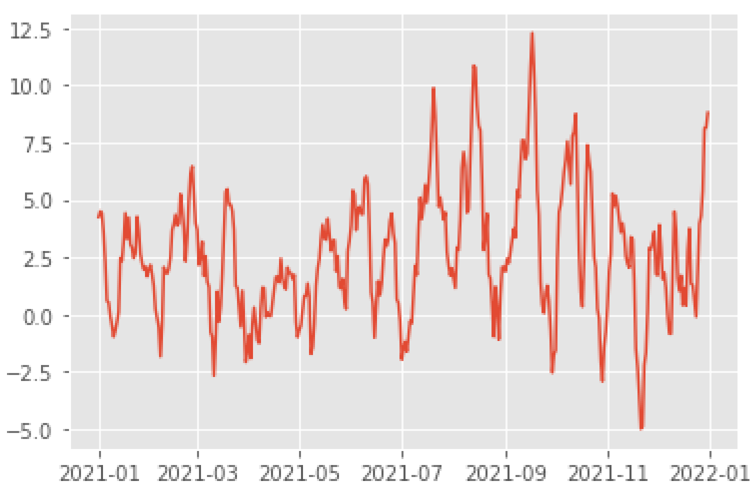
In [36]:
```
yhat_unt = pd.DataFrame((np.exp(yhat))-2172)
yhat_unt.set_index(final_test.index, inplace=True)
yhat_unt.columns = final_test.columns
yhat_unt
```

Out[36]:

date	WE.TAVE	WE.DP	WE.AP	WE.SWE	CC.TAVE	CC.DP	CC.AP	CC.SWE	II
2021-01-01	23.476178	-1.259875	11.147700	6.212654	18.764627	-0.703377	5.645577	2.817585	34.
2021-01-02	22.189106	-0.657093	4.813996	5.939483	18.555601	-0.093837	1.931726	2.741977	38.
2021-01-03	27.776006	-0.760503	6.635139	5.654675	22.982500	0.080217	2.978301	2.707002	29.
2021-01-04	34.698788	-0.869546	12.267858	5.597390	29.097652	-0.067308	5.943940	2.755053	36.
2021-01-05	38.001717	-0.384683	20.333256	5.960021	31.028817	0.333334	11.015121	3.234978	35.
...	...	...	...	...	...	...	...	...	
2021-	5.732763	0.481692	8.096230	6.213060	8.575081	0.315927	3.369080	2.589652	12.

In [37]:
```
plt.plot(((yhat_unt['Turb_FNU'])), label='Predicted')
```

Out[37]: [<matplotlib.lines.Line2D at 0x1d9dfa387f0>]



In [38]:
```
dibrinse = (final_test-yhat_unt)
dibb = dibrinse.sum()
mse = dibb/365
rmse = np.sqrt(mse)
rmse

#verify the right way to do this
```

```
CC.DP                 0.565091
CC.AP                      NaN
CC.SWE                     NaN
IP.TAVE               1.225996
IP.DP                 0.572623
IP.AP                      NaN
IP.SWE                     NaN
elevation.ft               NaN
net.evap.af           0.750756
delta.V.af                 NaN
regQ.cfs                   NaN
gain.cfs                   NaN
smoothed.natQ.cfs          NaN
Turb_FNU              1.332638
Chloro_RFU            0.676285
BGA_RFU               0.716176
ODO_mgL               0.735837
Temp_C                0.900165
Cond_muSCm            3.251065
```

In [39]:
```
final_test
```

Out[39]:

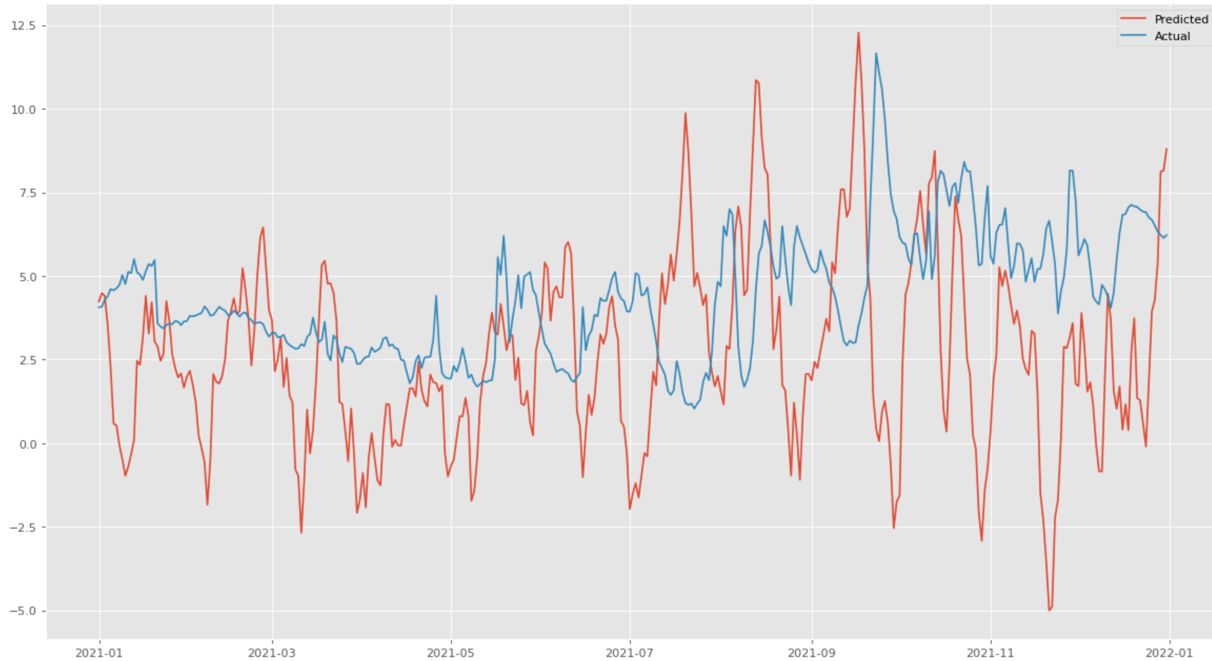
date	WE.TAVE	WE.DP	WE.AP	WE.SWE	CC.TAVE	CC.DP	CC.AP	CC.SWE	IP.TAVE	IP.DP	...
2021-01-01	16.0	0.0	8.2	7.3	19.0	0.0	4.0	3.6	23.0	0.0	...
2021-01-02	19.0	0.2	8.4	7.6	20.0	0.3	4.3	3.9	24.0	0.3	...
2021-01-03	22.0	0.4	8.8	7.9	24.0	0.1	4.4	4.0	27.0	0.3	...
2021-01-04	25.0	0.7	9.5	8.6	26.0	0.4	4.8	4.5	30.0	0.3	...
2021-01-05	23.0	0.6	10.1	9.0	25.0	0.2	5.0	4.7	29.0	0.4	...
...	...	...	...	...	...	...	...	...	...	...	...
2021-12-27	10.0	0.5	17.3	12.9	13.0	0.4	11.5	6.5	17.0	0.6	...
2021-12-28	2.0	0.0	17.3	13.0	3.0	0.0	11.5	6.5	9.0	0.0	...
2021-12-29	6.0	0.3	17.6	13.3	8.0	0.1	11.6	6.6	11.0	0.3	...
2021-12-30	10.0	0.4	18.0	13.7	11.0	0.1	11.7	6.7	16.0	0.3	...
2021-12-31	8.0	0.0	18.0	13.7	11.0	0.0	11.7	6.7	11.0	0.2	...

365 rows × 24 columns

In [40]:
```
figure(figsize=(18,10), dpi=80)


plt.plot(yhat_unt['Turb_FNU'], label='Predicted')
plt.plot(final_test['Turb_FNU'], label='Actual')
plt.rcParams["figure.figsize"] = (20,3)
plt.legend()
```

Out[40]: <matplotlib.legend.Legend at 0x1d9e1d70a90>



In [41]:
```
yhat_unt['Turb_FNU'].corr(final_test['Turb_FNU']**2)
```

Out[41]: -0.047142765276430756

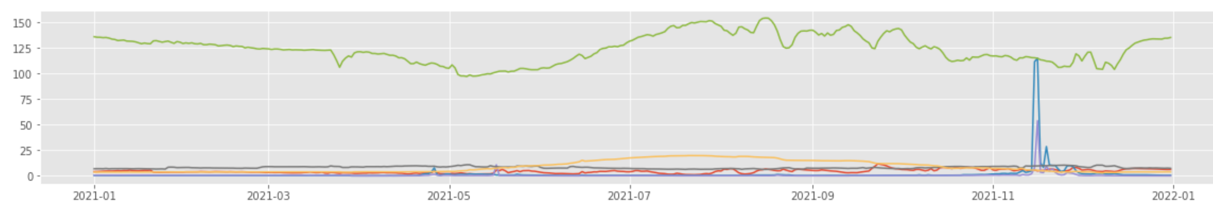
The above is the predicted values for Turbidity Using VAR. This result is if we were to predict the entire year without knowing any of the values for the predictor variables for the year. This shows that although the explanatory variables do contribute to the turbidity value, they themselves are not predictable for the time of year due to unpredictability in the environment, such as the weather.

In [42]:
```
yhat
```

Out[42]:
```
array([[7.69415424, 7.68282346, 7.68852301, ..., 7.68693467, 7.68500386,
        7.7427986 ],
       [7.69356783, 7.68310111, 7.68561762, ..., 7.68686827, 7.68525655,
        7.7442404 ],
       [7.69611082, 7.68305348, 7.68645388, ..., 7.68704759, 7.6854341 ,
        7.746909  ],
       ...,
       [7.68510894, 7.68326215, 7.68584053, ..., 7.68755501, 7.68452014,
        7.73519853],
       [7.70032479, 7.68330207, 7.68621462, ..., 7.68702958, 7.68441796,
        7.7381127 ],
       [7.70621816, 7.68332365, 7.68831199, ..., 7.68630425, 7.68419827,
        7.74284427]])
```

In [43]:
```
plt.plot(ipe_test)
```

Out[43]:
```
[<matplotlib.lines.Line2D at 0x1d9e1dc19a0>,
 <matplotlib.lines.Line2D at 0x1d9e20df3d0>,
 <matplotlib.lines.Line2D at 0x1d9e20df1f0>,
 <matplotlib.lines.Line2D at 0x1d9e20df5b0>,
 <matplotlib.lines.Line2D at 0x1d9e20df6d0>,
 <matplotlib.lines.Line2D at 0x1d9e20df7f0>]
```



In [44]:
```
# redo this within the scope of one dataset

jimmies = OLS(exog=yhat_unt, endog=final_test['Turb_FNU'])
jim = jimmies.fit()
print(jim.summary())
```

```
WE.SWE             -0.1679      0.024     -7.084      0.000      -0.215
-0.121
CC.TAVE             0.0731      0.034      2.130      0.034       0.006
0.141
CC.DP              -0.3635      0.310     -1.172      0.242      -0.974
0.247
CC.AP              -0.0647      0.085     -0.758      0.449      -0.232
0.103
CC.SWE              0.1361      0.086      1.576      0.116      -0.034
0.306
IP.TAVE             0.0330      0.012      2.810      0.005       0.010
0.056
IP.DP              -0.9223      1.404     -0.657      0.512      -3.684
1.839
IP.AP              -0.0999      0.122     -0.819      0.413      -0.340
0.140
IP.SWE             -0.0351      0.081     -0.431      0.667      -0.195
0.125
elevation.ft        0.0008      0.000      4.038      0.000       0.000
```

In [45]:
```
X_train = final_train.drop(['Turb_FNU'], axis=1)
y_train = final_train.Turb_FNU
```

In [46]:
```
X_test = final_test.drop(['Turb_FNU'], axis=1)
y_test = final_test.Turb_FNU
```

As others have stated, you need to have a common frequency of measurement (i.e. the time between observations). With that in place I would identify a common model that would reasonably describe each series separately. This might be an ARIMA model or a multiply-trended Regression Model with possible Level Shifts or a composite model integrating both memory (ARIMA) and dummy variables. This common model could be estimated globally and separately for each of the two series and then one could construct an F test to test the hypothesis of a common set of parameters.

Use LSTM neuran network RNN

In [65]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout

m = Sequential()
m.add(LSTM(units=50, return_sequences=True,
          input_shape=(X_train.shape[1],1)))
m.add(Dropout(0.2))
m.add(LSTM(units=50))
m.add(Dropout(0.1))
m.add(Dense(units=1))
m.compile(optimizer = 'adam', loss = 'mean_squared_error')
```

In [68]:
```
history = m.fit(X_train, y_train, epochs=50, batch_size=50, verbose=1)

plt.figure()
plt.ylabel('Loss'); plt.xlabel('Epoch')
plt.semilogy(history.history['loss'])
```

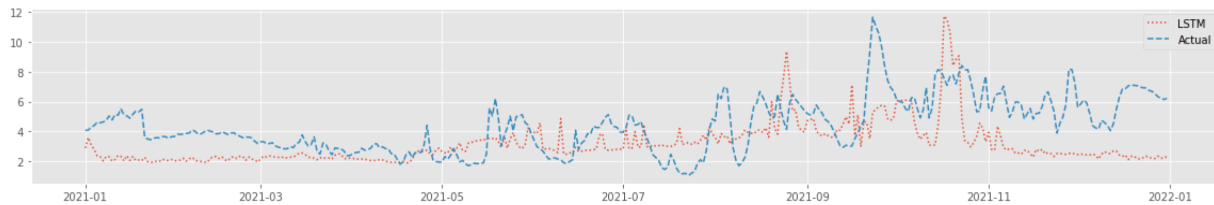
```
Epoch 36/50
37/37 [==============================] - 0s 11ms/step - loss: 2.1325
Epoch 37/50
37/37 [==============================] - 0s 11ms/step - loss: 2.0466
Epoch 38/50
37/37 [==============================] - 0s 11ms/step - loss: 2.2459
Epoch 39/50
37/37 [==============================] - 0s 11ms/step - loss: 2.0911
Epoch 40/50
37/37 [==============================] - 0s 11ms/step - loss: 2.2538
Epoch 41/50
37/37 [==============================] - 0s 11ms/step - loss: 2.1178
Epoch 42/50
37/37 [==============================] - 0s 11ms/step - loss: 2.1462
Epoch 43/50
37/37 [==============================] - 0s 11ms/step - loss: 1.9684
Epoch 44/50
37/37 [==============================] - 0s 11ms/step - loss: 1.9462
Epoch 45/50
37/37 [==============================] - 0s 11ms/step - loss: 2.3955
```

In [69]:
```
y_pred = m.predict(X_test)
y_pred = pd.DataFrame(y_pred)
y_pred.set_index(X_test.index, inplace=True)


plt.figure()
plt.plot(y_pred,':',label='LSTM')
plt.plot(y_test,'--',label='Actual')
plt.legend()
```

```
12/12 [==============================] - 0s 3ms/step
```

Out[69]: <matplotlib.legend.Legend at 0x1d9881ebe20>



In [ ]: