Covid-19 Anomaly Detection Models

Algorithms and Code

WPI

Principal Component Analysis

Estimation Method Details

LOF

An unsupervised anomaly detection method which computes the local density deviation of a given data point with respect to its neighbors. It considers as outliers the samples that have a substantially lower density than their neighbors.

PCA

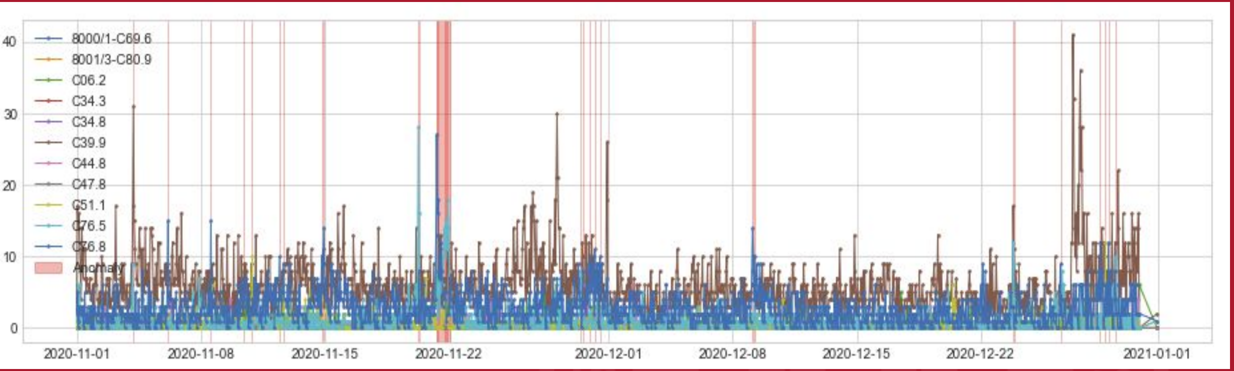
Solves the problem by analyzing available features to determine what constitutes a "normal" class. The component then applies distance metrics to identify cases that represent anomalies. This approach lets you train a model by using existing imbalanced data

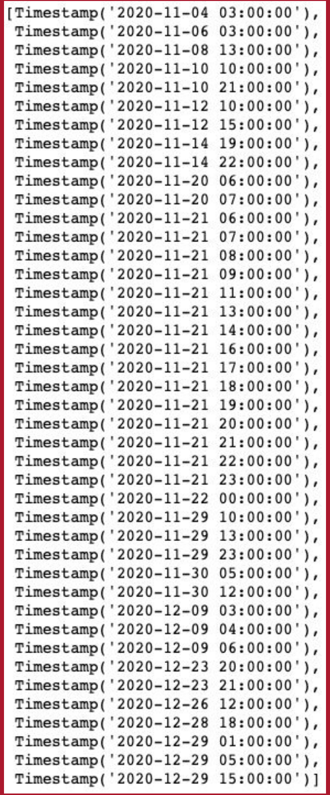
Review of PCA for Anomaly Detection

- Every time point is a vector in high dimensional space
- We track the reconstruction error of these vectors
- Analyze available features to determine what constitutes a "normal" class
- The component then applies distance metrics to identify cases that represent anomalies

Before running the PCA outlier detection code, we run a basic PCA algorithm to get the number of principal components to use on our data.

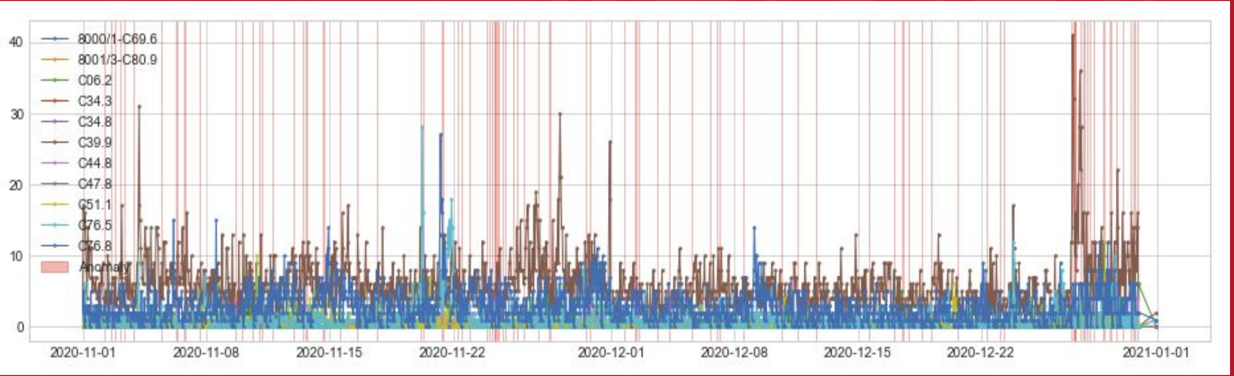
K = 1

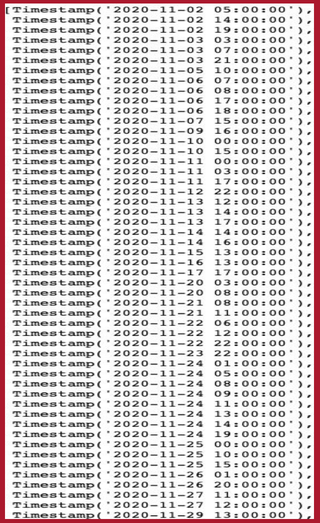




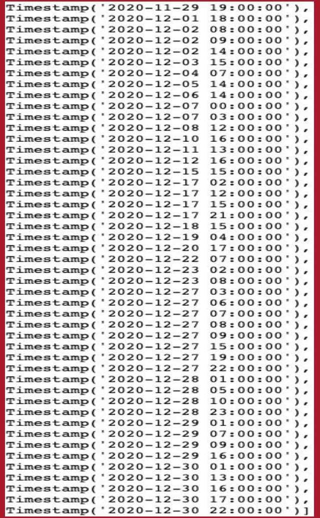
```
[Timestamp('2020-11-04 03:00:00'),
 Timestamp('2020-11-06 03:00:00'),
 Timestamp('2020-11-08 13:00:00'),
 Timestamp('2020-11-10 10:00:00'),
 Timestamp('2020-11-10 21:00:00'),
 Timestamp('2020-11-12 10:00:00'),
 Timestamp('2020-11-12 15:00:00'),
 Timestamp('2020-11-14 19:00:00'),
 Timestamp('2020-11-14 22:00:00'),
 Timestamp('2020-11-20 06:00:00'),
 Timestamp('2020-11-20 07:00:00'),
 Timestamp('2020-11-21 06:00:00'),
 Timestamp('2020-11-21 07:00:00'),
 Timestamp('2020-11-21 08:00:00'),
 Timestamp('2020-11-21 09:00:00'),
 Timestamp('2020-11-21 11:00:00'),
 Timestamp('2020-11-21 13:00:00'),
 Timestamp('2020-11-21 14:00:00'),
 Timestamp('2020-11-21 16:00:00'),
 Timestamp('2020-11-21 17:00:00'),
 Timestamp('2020-11-21 18:00:00'),
 Timestamp('2020-11-21 19:00:00'),
 Timestamp('2020-11-21 20:00:00'),
 Timestamp('2020-11-21 21:00:00'),
 Timestamp('2020-11-21 22:00:00'),
 Timestamp('2020-11-21 23:00:00'),
 Timestamp('2020-11-22 00:00:00'),
 Timestamp('2020-11-29 10:00:00'),
 Timestamp('2020-11-29 13:00:00'),
 Timestamp('2020-11-29 23:00:00'),
 Timestamp('2020-11-30 05:00:00'),
 Timestamp('2020-11-30 12:00:00'),
 Timestamp('2020-12-09 03:00:00'),
 Timestamp('2020-12-09 04:00:00'),
 Timestamp('2020-12-09 06:00:00'),
 Timestamp('2020-12-23 20:00:00'),
 Timestamp('2020-12-23 21:00:00'),
 Timestamp('2020-12-26 12:00:00'),
 Timestamp('2020-12-28 18:00:00'),
 Timestamp('2020-12-29 01:00:00'),
 Timestamp('2020-12-29 05:00:00'),
 Timestamp('2020-12-29 15:00:00')]
```

K = 5



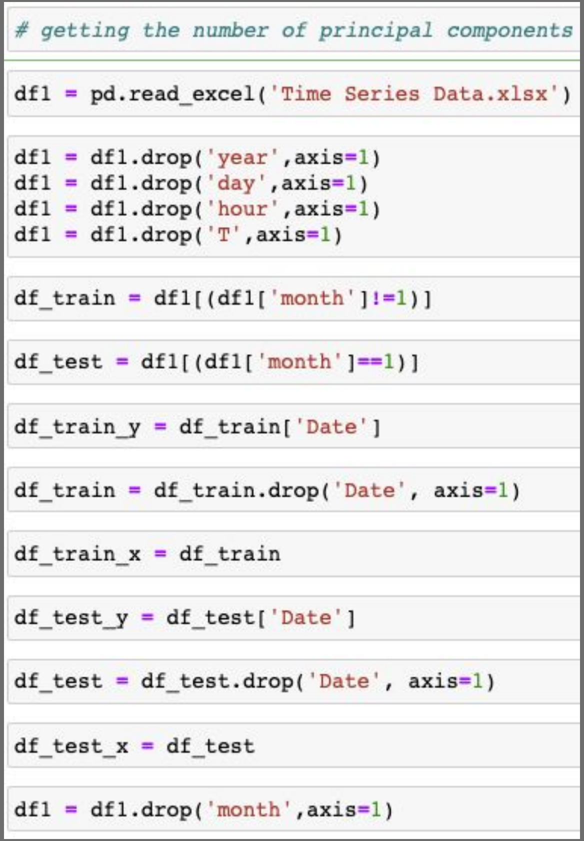


```
[Timestamp('2020-11-02 05:00:00'),
 Timestamp('2020-11-02 14:00:00'),
 Timestamp('2020-11-02 19:00:00'),
 Timestamp('2020-11-03 03:00:00'),
 Timestamp('2020-11-03 07:00:00'),
 Timestamp('2020-11-03 21:00:00'),
 Timestamp('2020-11-05 10:00:00'),
 Timestamp('2020-11-06 07:00:00'),
 Timestamp('2020-11-06 08:00:00'),
 Timestamp('2020-11-06 17:00:00'),
 Timestamp('2020-11-06 18:00:00'),
 Timestamp('2020-11-07 15:00:00'),
 Timestamp('2020-11-09 16:00:00'),
 Timestamp('2020-11-10 00:00:00'),
 Timestamp('2020-11-10 15:00:00'),
 Timestamp('2020-11-11 00:00:00'),
 Timestamp('2020-11-11 03:00:00'),
 Timestamp('2020-11-11 17:00:00'),
 Timestamp('2020-11-12 22:00:00'),
 Timestamp('2020-11-13 12:00:00'),
 Timestamp('2020-11-13 14:00:00'),
 Timestamp('2020-11-13 17:00:00'),
 Timestamp('2020-11-14 14:00:00'),
 Timestamp('2020-11-14 16:00:00'),
 Timestamp('2020-11-15 13:00:00'),
 Timestamp('2020-11-16 13:00:00'),
 Timestamp('2020-11-17 17:00:00'),
 Timestamp('2020-11-20 03:00:00'),
 Timestamp('2020-11-20 08:00:00'),
 Timestamp('2020-11-21 08:00:00'),
 Timestamp('2020-11-21 11:00:00'),
 Timestamp('2020-11-22 06:00:00'),
 Timestamp('2020-11-22 12:00:00'),
 Timestamp('2020-11-22 22:00:00'),
 Timestamp('2020-11-23 22:00:00'),
 Timestamp('2020-11-24 01:00:00'),
 Timestamp('2020-11-24 05:00:00'),
 Timestamp('2020-11-24 08:00:00'),
 Timestamp('2020-11-24 09:00:00'),
 Timestamp('2020-11-24 11:00:00'),
 Timestamp('2020-11-24 13:00:00'),
 Timestamp('2020-11-24 14:00:00'),
 Timestamp('2020-11-24 19:00:00'),
 Timestamp('2020-11-25 00:00:00'),
 Timestamp('2020-11-25 10:00:00'),
 Timestamp('2020-11-25 15:00:00'),
 Timestamp('2020-11-26 01:00:00'),
 Timestamp('2020-11-26 20:00:00'),
 Timestamp('2020-11-27 11:00:00'),
 Timestamp('2020-11-27 12:00:00'),
 Timestamp('2020-11-29 13:00:00'),
```

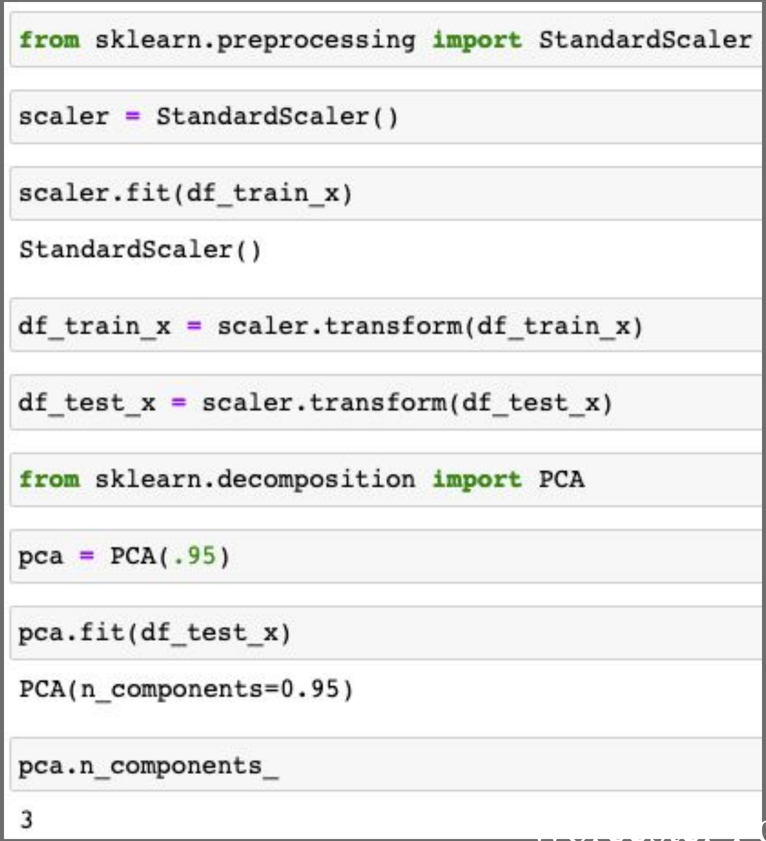


```
Timestamp('2020-11-29 19:00:00'),
Timestamp('2020-12-01 18:00:00'),
Timestamp('2020-12-02 08:00:00'),
Timestamp('2020-12-02 09:00:00'),
Timestamp('2020-12-02 14:00:00'),
Timestamp('2020-12-03 15:00:00'),
Timestamp('2020-12-04 07:00:00'),
Timestamp('2020-12-05 14:00:00'),
Timestamp('2020-12-06 14:00:00'),
Timestamp('2020-12-07 00:00:00'),
Timestamp('2020-12-07 03:00:00'),
Timestamp('2020-12-08 12:00:00'),
Timestamp('2020-12-10 16:00:00'),
Timestamp('2020-12-11 13:00:00'),
Timestamp('2020-12-12 16:00:00'),
Timestamp('2020-12-15 15:00:00'),
Timestamp('2020-12-17 02:00:00'),
Timestamp('2020-12-17 12:00:00'),
Timestamp('2020-12-17 15:00:00'),
Timestamp('2020-12-17 21:00:00'),
Timestamp('2020-12-18 15:00:00'),
Timestamp('2020-12-19 04:00:00'),
Timestamp('2020-12-20 17:00:00'),
Timestamp('2020-12-22 07:00:00'),
Timestamp('2020-12-23 02:00:00'),
Timestamp('2020-12-23 08:00:00'),
Timestamp('2020-12-27 03:00:00'),
Timestamp('2020-12-27 06:00:00'),
Timestamp('2020-12-27 07:00:00'),
Timestamp('2020-12-27 08:00:00'),
Timestamp('2020-12-27 09:00:00'),
Timestamp('2020-12-27 15:00:00'),
Timestamp('2020-12-27 19:00:00'),
Timestamp('2020-12-27 22:00:00'),
Timestamp('2020-12-28 01:00:00'),
Timestamp('2020-12-28 05:00:00'),
Timestamp('2020-12-28 10:00:00'),
Timestamp('2020-12-28 23:00:00'),
Timestamp('2020-12-29 01:00:00'),
Timestamp('2020-12-29 07:00:00'),
Timestamp('2020-12-29 09:00:00'),
Timestamp('2020-12-29 16:00:00'),
Timestamp('2020-12-30 01:00:00'),
Timestamp('2020-12-30 13:00:00'),
Timestamp('2020-12-30 16:00:00'),
Timestamp('2020-12-30 17:00:00'),
Timestamp('2020-12-30 22:00:00')]
```

Getting the optimal numbers of principal components



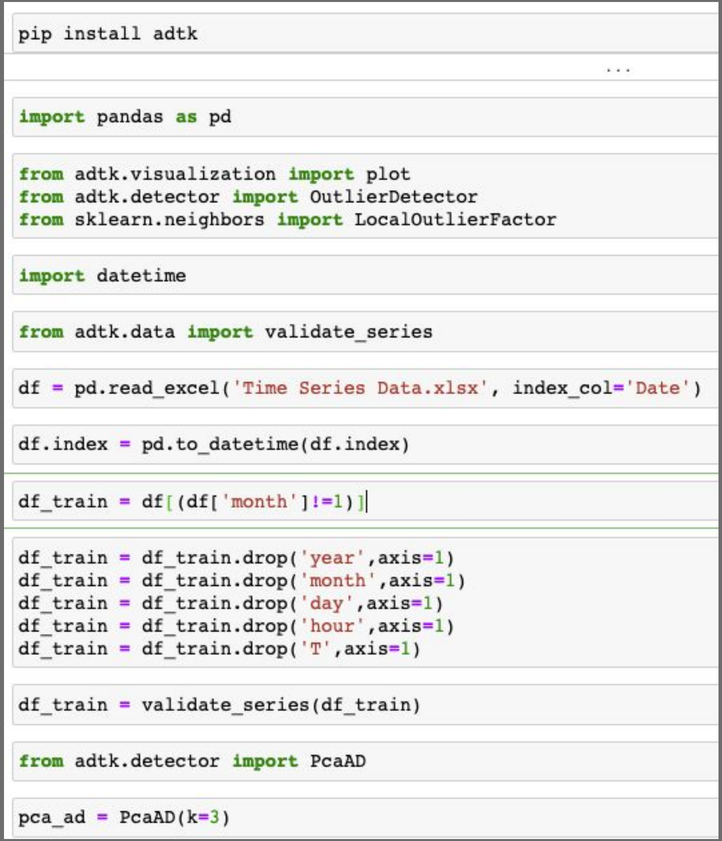
```
# getting the number of principal components

df1 = pd.read_excel('Time Series Data.xlsx')

df1 = df1.drop('year',axis=1)
df1 = df1.drop('day',axis=1)
df1 = df1.drop('hour',axis=1)
df1 = df1.drop('T',axis=1)

df_train = df1[(df1['month']!=1)]

df_test = df1[(df1['month']==1)]

df_train_y = df_train['Date']

df_train = df_train.drop('Date', axis=1)

df_train_x = df_train

df_test_y = df_test['Date']

df_test = df_test.drop('Date', axis=1)

df_test_x = df_test

df1 = df1.drop('month',axis=1)
```



```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(df_train_x)

StandardScaler()

df_train_x = scaler.transform(df_train_x)

df_test_x = scaler.transform(df_test_x)

from sklearn.decomposition import PCA

pca = PCA(.95)

pca.fit(df_test_x)

PCA(n_components=0.95)

pca.n_components_

3
```

3 principal components will be used for our algorithm

ADTK PCA Process



```
pip install adtk
```

```
...
```

```
import pandas as pd
```

```
from adtk.visualization import plot
from adtk.detector import OutlierDetector
from sklearn.neighbors import LocalOutlierFactor
```

```
import datetime
```

```
from adtk.data import validate_series
```

```
df = pd.read_excel('Time Series Data.xlsx', index_col='Date')
```

```
df.index = pd.to_datetime(df.index)
```

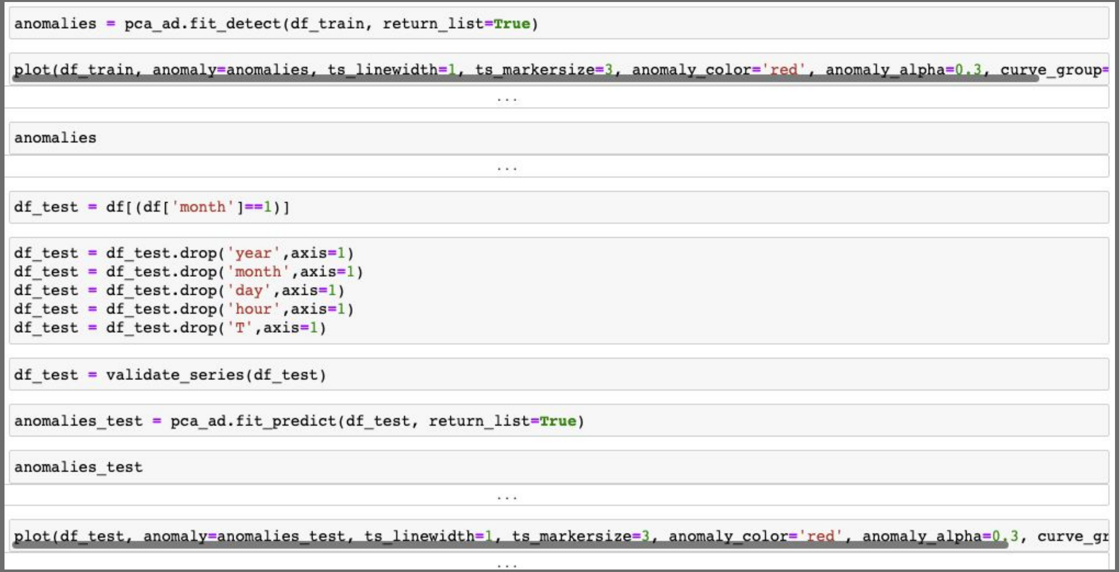
```
df_train = df[(df['month']!=1)]
```

```
df_train = df_train.drop('year',axis=1)
df_train = df_train.drop('month',axis=1)
df_train = df_train.drop('day',axis=1)
df_train = df_train.drop('hour',axis=1)
df_train = df_train.drop('T',axis=1)
```

```
df_train = validate_series(df_train)
```

```
from adtk.detector import PcaAD
```

```
pca_ad = PcaAD(k=3)
```



```
anomalies = pca_ad.fit_detect(df_train, return_list=True)
```

```
plot(df_train, anomaly=anomalies, ts_linewidth=1, ts_markersize=3, anomaly_color='red', anomaly_alpha=0.3, curve_group=
```

```
...
```

```
anomalies
```

```
...
```

```
df_test = df[(df['month']==1)]
```

```
df_test = df_test.drop('year',axis=1)
df_test = df_test.drop('month',axis=1)
df_test = df_test.drop('day',axis=1)
df_test = df_test.drop('hour',axis=1)
df_test = df_test.drop('T',axis=1)
```

```
df_test = validate_series(df_test)
```

```
anomalies_test = pca_ad.fit_predict(df_test, return_list=True)
```

```
anomalies_test
```

```
...
```

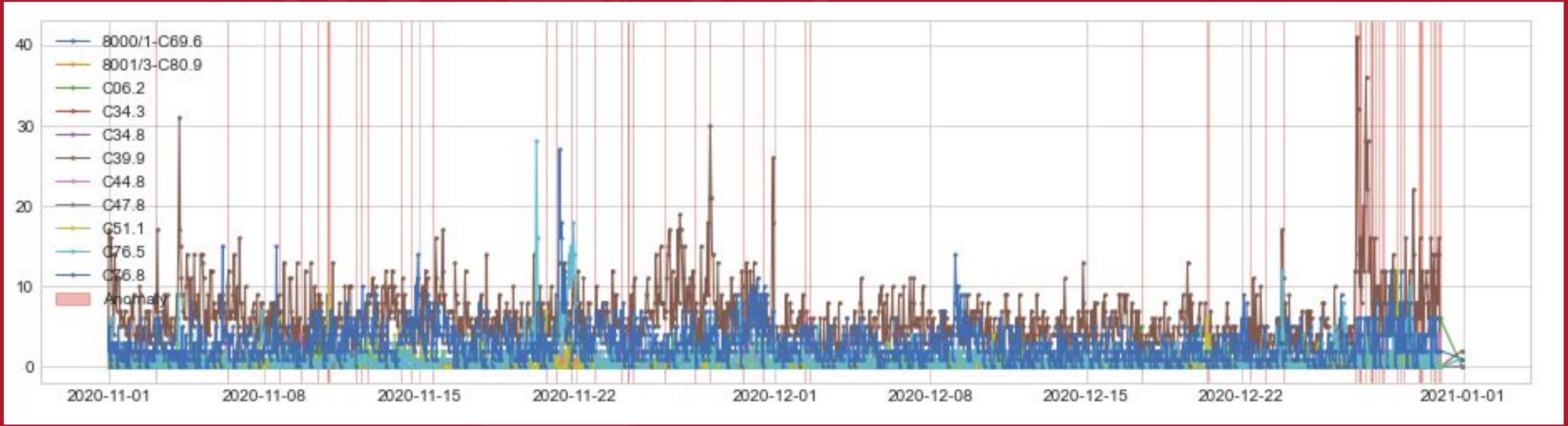
```
plot(df_test, anomaly=anomalies_test, ts_linewidth=1, ts_markersize=3, anomaly_color='red', anomaly_alpha=0.3, curve_gr
```

```
...
```

Worcester Polytechnic Institute

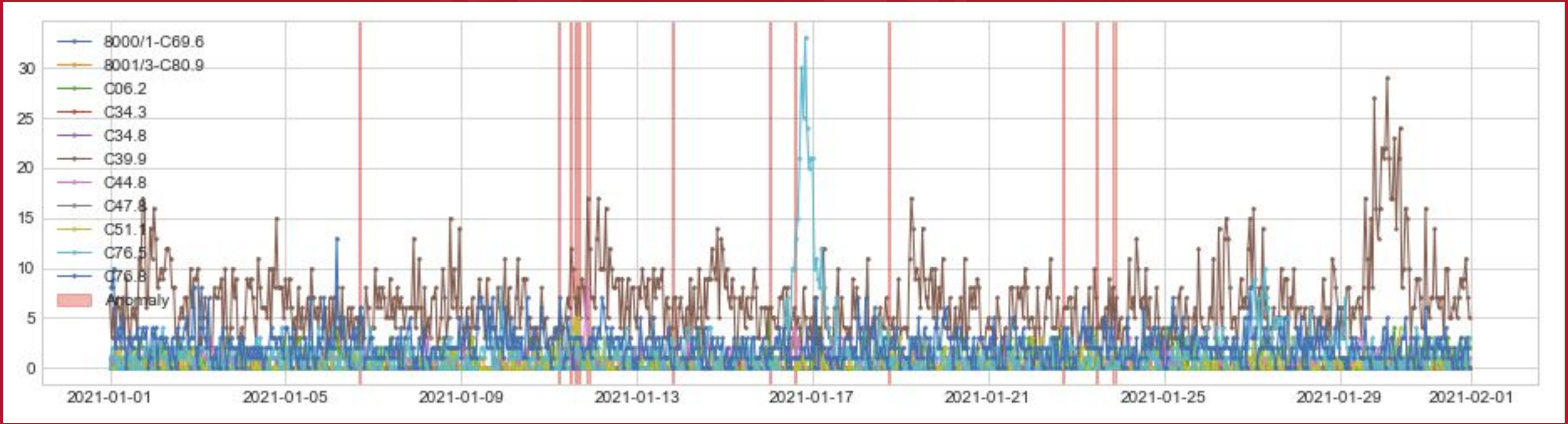
Training Data K = 3

```
[Timestamp('2020-11-03 03:00:00'),
 Timestamp('2020-11-06 08:00:00'),
 Timestamp('2020-11-08 15:00:00'),
 Timestamp('2020-11-09 16:00:00'),
 Timestamp('2020-11-10 10:00:00'),
 Timestamp('2020-11-10 20:00:00'),
 Timestamp('2020-11-10 21:00:00'),
 Timestamp('2020-11-10 22:00:00'),
 Timestamp('2020-11-12 03:00:00'),
 Timestamp('2020-11-12 09:00:00'),
 Timestamp('2020-11-12 15:00:00'),
 Timestamp('2020-11-14 03:00:00'),
 Timestamp('2020-11-14 14:00:00'),
 Timestamp('2020-11-15 13:00:00'),
 Timestamp('2020-11-20 16:00:00'),
 Timestamp('2020-11-21 03:00:00'),
 Timestamp('2020-11-21 19:00:00'),
 Timestamp('2020-11-22 20:00:00'),
 Timestamp('2020-11-24 01:00:00'),
 Timestamp('2020-11-24 08:00:00'),
 Timestamp('2020-11-24 09:00:00'),
 Timestamp('2020-11-24 14:00:00'),
 Timestamp('2020-11-26 01:00:00'),
 Timestamp('2020-11-27 09:00:00'),
 Timestamp('2020-11-28 01:00:00'),
 Timestamp('2020-11-29 13:00:00'),
 Timestamp('2020-11-30 11:00:00'),
 Timestamp('2020-12-02 08:00:00'),
 Timestamp('2020-12-02 14:00:00'),
 Timestamp('2020-12-17 12:00:00'),
 Timestamp('2020-12-20 10:00:00'),
 Timestamp('2020-12-20 13:00:00'),
 Timestamp('2020-12-22 10:00:00'),
 Timestamp('2020-12-23 02:00:00'),
 Timestamp('2020-12-23 22:00:00'),
 Timestamp('2020-12-27 03:00:00'),
 Timestamp('2020-12-27 07:00:00'),
 Timestamp('2020-12-27 08:00:00'),
 Timestamp('2020-12-27 09:00:00'),
 Timestamp('2020-12-27 14:00:00'),
 Timestamp('2020-12-27 19:00:00'),
 Timestamp('2020-12-27 21:00:00'),
 Timestamp('2020-12-27 22:00:00'),
 Timestamp('2020-12-28 01:00:00'),
 Timestamp('2020-12-28 05:00:00'),
 Timestamp('2020-12-28 09:00:00'),
 Timestamp('2020-12-28 10:00:00'),
 Timestamp('2020-12-29 01:00:00'),
 Timestamp('2020-12-29 04:00:00'),
 Timestamp('2020-12-29 07:00:00'),
 Timestamp('2020-12-30 00:00:00'),
 Timestamp('2020-12-30 01:00:00'),
 Timestamp('2020-12-30 02:00:00'),
 Timestamp('2020-12-30 04:00:00'),
 Timestamp('2020-12-30 13:00:00'),
 Timestamp('2020-12-30 16:00:00'),
 Timestamp('2020-12-30 18:00:00'),
 Timestamp('2020-12-30 22:00:00'),
 Timestamp('2020-12-30 23:00:00')]
```



```
[(Timestamp('2021-01-06 16:00:00', freq='H'),
  Timestamp('2021-01-06 16:59:59.999999999', freq='H')),
 (Timestamp('2021-01-11 05:00:00', freq='H'),
  Timestamp('2021-01-11 05:59:59.999999999', freq='H')),
 (Timestamp('2021-01-11 11:00:00', freq='H'),
  Timestamp('2021-01-11 11:59:59.999999999', freq='H')),
 (Timestamp('2021-01-11 14:00:00', freq='H'),
  Timestamp('2021-01-11 14:59:59.999999999', freq='H')),
 (Timestamp('2021-01-11 16:00:00', freq='H'),
  Timestamp('2021-01-11 16:59:59.999999999', freq='H')),
 (Timestamp('2021-01-11 20:00:00', freq='H'),
  Timestamp('2021-01-11 21:59:59.999999999', freq='H')),
 (Timestamp('2021-01-13 19:00:00', freq='H'),
  Timestamp('2021-01-13 19:59:59.999999999', freq='H')),
 (Timestamp('2021-01-16 00:00:00', freq='H'),
  Timestamp('2021-01-16 00:59:59.999999999', freq='H')),
 (Timestamp('2021-01-16 14:00:00', freq='H'),
  Timestamp('2021-01-16 14:59:59.999999999', freq='H')),
 (Timestamp('2021-01-18 17:00:00', freq='H'),
  Timestamp('2021-01-18 17:59:59.999999999', freq='H')),
 (Timestamp('2021-01-22 16:00:00', freq='H'),
  Timestamp('2021-01-22 16:59:59.999999999', freq='H')),
 (Timestamp('2021-01-23 11:00:00', freq='H'),
  Timestamp('2021-01-23 11:59:59.999999999', freq='H')),
 (Timestamp('2021-01-23 20:00:00', freq='H'),
  Timestamp('2021-01-23 21:59:59.999999999', freq='H'))]
```

Testing Data K = 3



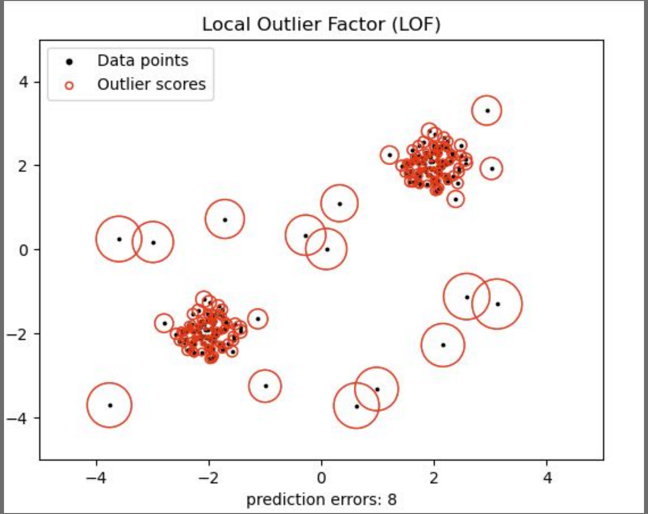
Potential “gaps” of PCA Anomaly Detection

- Data standardization is necessary to run the algorithm
- Principal components are not readable / interpretable compared to original features
- There is always the fear of information loss during the PCA process
  - We do our best by running the dimensionality reduction code and picking the optimal number of principal components, but it is never perfect

Local Outlier Factor

Anomaly Detection Methodology Review

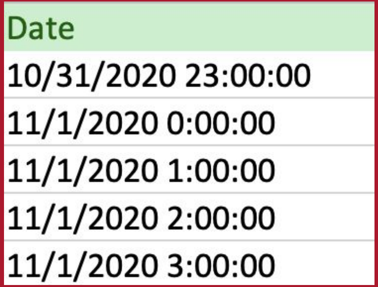
1. Compute the local density deviation of a given data point with respect to its neighbors
2. It considers as outliers the samples that have a substantially lower density than their neighbors



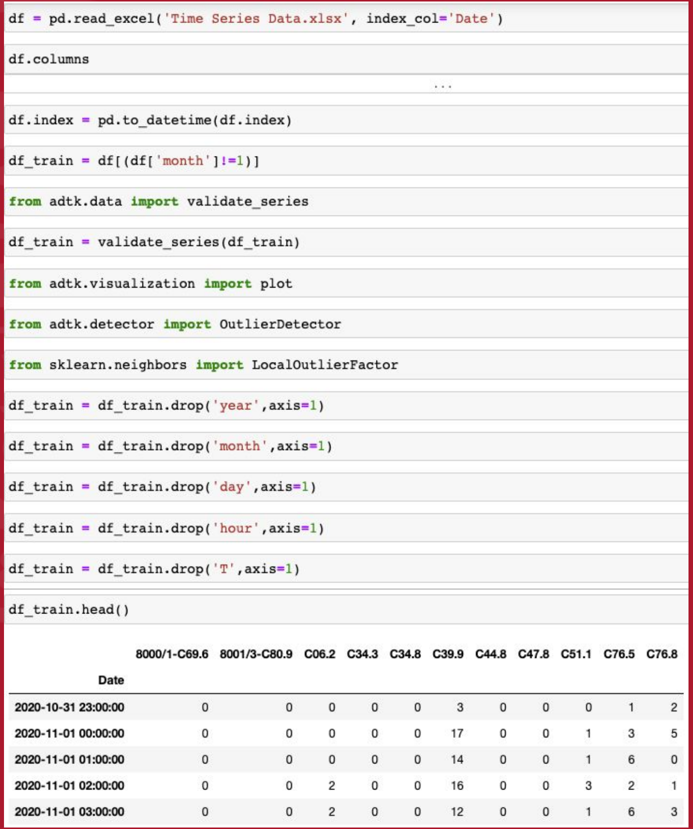
Worcester Polytechnic Institute

Create a column that has day-month-year-hour

Read in the data and set this new date column to the index, change the index to pd.datetime, create a training set, import necessary libraries, validate the training set, drop the un-important columns



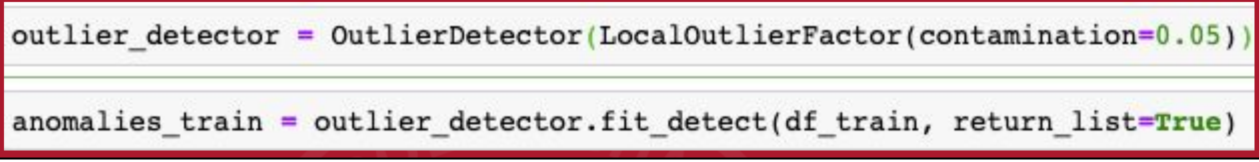
Date
10/31/2020 23:00:00
11/1/2020 0:00:00
11/1/2020 1:00:00
11/1/2020 2:00:00
11/1/2020 3:00:00



```
df = pd.read_excel('Time Series Data.xlsx', index_col='Date')

df.columns

...

df.index = pd.to_datetime(df.index)

df_train = df[(df['month']!=1)]

from adtk.data import validate_series

df_train = validate_series(df_train)

from adtk.visualization import plot

from adtk.detector import OutlierDetector

from sklearn.neighbors import LocalOutlierFactor

df_train = df_train.drop('year',axis=1)

df_train = df_train.drop('month',axis=1)

df_train = df_train.drop('day',axis=1)

df_train = df_train.drop('hour',axis=1)

df_train = df_train.drop('T',axis=1)

df_train.head()
```

```
                     8000/1-C69.6  8001/3-C80.9  C06.2  C34.3  C34.8  C39.9  C44.8  C47.8  C51.1  C76.5  C76.8
               Date
2020-10-31 23:00:00             0             0      0      0      0      3      0      0      0      1      2
2020-11-01 00:00:00             0             0      0      0      0     17      0      0      1      3      5
2020-11-01 01:00:00             0             0      0      0      0     14      0      0      1      6      0
2020-11-01 02:00:00             0             0      2      0      0     16      0      0      3      2      1
2020-11-01 03:00:00             0             0      2      0      0     12      0      0      1      6      3
```

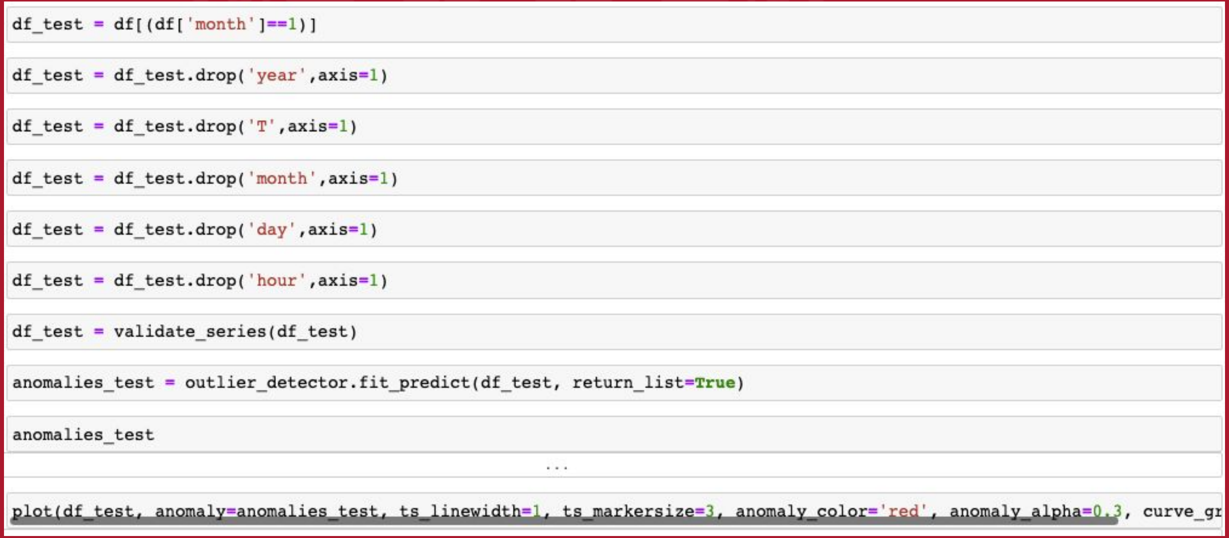
Create an outlier detection method using scikit-learn's Local Outlier Factor module and fit that model to the training data in order to detect anomalies in the training set



```
outlier_detector = OutlierDetector(LocalOutlierFactor(contamination=0.05))
```

```
anomalies_train = outlier_detector.fit_detect(df_train, return_list=True)
```

Create the test set and clean it up, use the predict() function to take the previous local outlier factor model and predict testing values on the new data



```
df_test = df[(df['month']==1)]
```

```
df_test = df_test.drop('year',axis=1)
```

```
df_test = df_test.drop('T',axis=1)
```

```
df_test = df_test.drop('month',axis=1)
```

```
df_test = df_test.drop('day',axis=1)
```

```
df_test = df_test.drop('hour',axis=1)
```

```
df_test = validate_series(df_test)
```

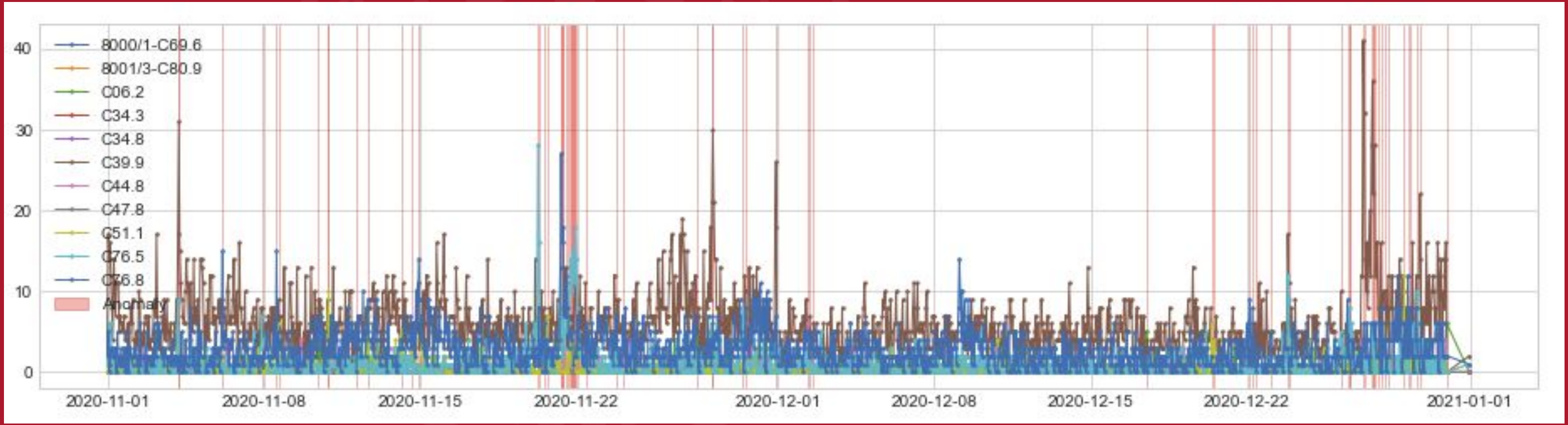
```
anomalies_test = outlier_detector.fit_predict(df_test, return_list=True)
```

```
anomalies_test
```

...

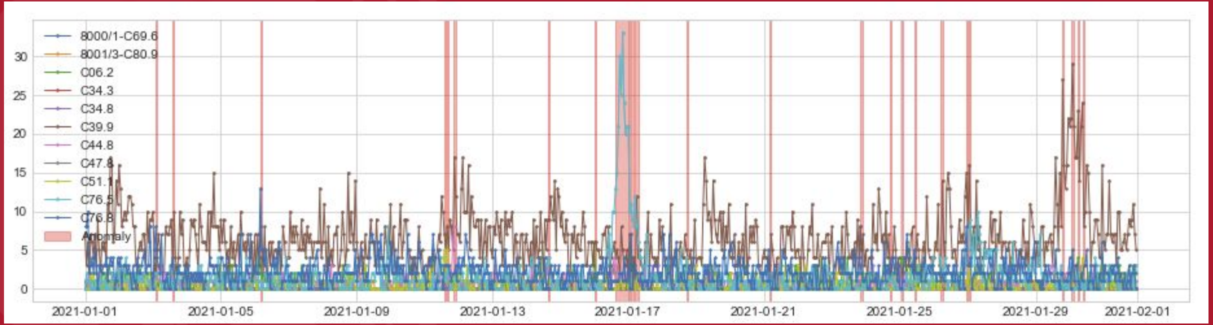
```
plot(df_test, anomaly=anomalies_test, ts_linewidth=1, ts_markersize=3, anomaly_color='red', anomaly_alpha=0.3, curve_gr
[Timestamp('2020-11-04 03:00:00'),
 Timestamp('2020-11-04 04:00:00'),
 Timestamp('2020-11-06 03:00:00'),
 Timestamp('2020-11-07 21:00:00'),
 Timestamp('2020-11-08 13:00:00'),
 Timestamp('2020-11-08 15:00:00'),
 Timestamp('2020-11-10 10:00:00'),
 Timestamp('2020-11-10 20:00:00'),
 Timestamp('2020-11-10 21:00:00'),
 Timestamp('2020-11-12 03:00:00'),
 Timestamp('2020-11-12 15:00:00'),
 Timestamp('2020-11-14 03:00:00'),
 Timestamp('2020-11-14 14:00:00'),
 Timestamp('2020-11-14 22:00:00'),
 Timestamp('2020-11-20 06:00:00'),
 Timestamp('2020-11-20 07:00:00'),
 Timestamp('2020-11-20 13:00:00'),
 Timestamp('2020-11-20 16:00:00'),
 Timestamp('2020-11-21 06:00:00'),
 Timestamp('2020-11-21 07:00:00'),
 Timestamp('2020-11-21 08:00:00'),
 Timestamp('2020-11-21 09:00:00'),
 Timestamp('2020-11-21 13:00:00'),
 Timestamp('2020-11-21 14:00:00'),
 Timestamp('2020-11-21 16:00:00'),
 Timestamp('2020-11-21 17:00:00'),
 Timestamp('2020-11-21 18:00:00'),
 Timestamp('2020-11-21 19:00:00'),
 Timestamp('2020-11-21 20:00:00'),
 Timestamp('2020-11-21 21:00:00'),
 Timestamp('2020-11-21 22:00:00'),
 Timestamp('2020-11-21 23:00:00'),
 Timestamp('2020-11-22 10:00:00'),
 Timestamp('2020-11-23 18:00:00'),
 Timestamp('2020-11-24 01:00:00'),
 Timestamp('2020-11-27 09:00:00'),
 Timestamp('2020-11-28 01:00:00'),
 Timestamp('2020-11-28 02:00:00'),
 Timestamp('2020-11-29 10:00:00'),
 Timestamp('2020-11-29 13:00:00'),
 Timestamp('2020-11-30 22:00:00'),
 Timestamp('2020-12-02 08:00:00'),
 Timestamp('2020-12-02 09:00:00'),
 Timestamp('2020-12-02 14:00:00'),
 Timestamp('2020-12-17 12:00:00'),
 Timestamp('2020-12-20 10:00:00'),
 Timestamp('2020-12-20 13:00:00'),
 Timestamp('2020-12-22 03:00:00'),
 Timestamp('2020-12-22 06:00:00'),
 Timestamp('2020-12-22 10:00:00'),
 Timestamp('2020-12-23 02:00:00'),
 Timestamp('2020-12-23 20:00:00'),
 Timestamp('2020-12-23 21:00:00'),
 Timestamp('2020-12-26 05:00:00'),
 Timestamp('2020-12-26 12:00:00'),
 Timestamp('2020-12-26 14:00:00'),
 Timestamp('2020-12-26 15:00:00'),
 Timestamp('2020-12-27 05:00:00'),
 Timestamp('2020-12-27 06:00:00'),
 Timestamp('2020-12-27 14:00:00'),
 Timestamp('2020-12-27 15:00:00'),
 Timestamp('2020-12-27 16:00:00'),
 Timestamp('2020-12-27 17:00:00'),
 Timestamp('2020-12-27 22:00:00'),
 Timestamp('2020-12-28 01:00:00'),
 Timestamp('2020-12-28 05:00:00'),
 Timestamp('2020-12-28 09:00:00'),
 Timestamp('2020-12-29 01:00:00'),
 Timestamp('2020-12-29 05:00:00'),
 Timestamp('2020-12-29 07:00:00'),
 Timestamp('2020-12-29 15:00:00'),
 Timestamp('2020-12-29 18:00:00'),
 Timestamp('2020-12-30 23:00:00')]
```

Timestamps of the anomalies detected in training (November and December)



Predicted anomalies for January

```
[(Timestamp('2021-01-03 02:00:00', freq='H'),
  Timestamp('2021-01-03 02:59:59.999999999', freq='H')),
 (Timestamp('2021-01-03 14:00:00', freq='H'),
  Timestamp('2021-01-03 14:59:59.999999999', freq='H')),
 (Timestamp('2021-01-06 04:00:00', freq='H'),
  Timestamp('2021-01-06 04:59:59.999999999', freq='H')),
 (Timestamp('2021-01-11 14:00:00', freq='H'),
  Timestamp('2021-01-11 14:59:59.999999999', freq='H')),
 (Timestamp('2021-01-11 16:00:00', freq='H'),
  Timestamp('2021-01-11 16:59:59.999999999', freq='H')),
 (Timestamp('2021-01-11 20:00:00', freq='H'),
  Timestamp('2021-01-11 21:59:59.999999999', freq='H')),
 (Timestamp('2021-01-14 15:00:00', freq='H'),
  Timestamp('2021-01-14 15:59:59.999999999', freq='H')),
 (Timestamp('2021-01-16 00:00:00', freq='H'),
  Timestamp('2021-01-16 00:59:59.999999999', freq='H')),
 (Timestamp('2021-01-16 15:00:00', freq='H'),
  Timestamp('2021-01-17 00:59:59.999999999', freq='H')),
 (Timestamp('2021-01-17 02:00:00', freq='H'),
  Timestamp('2021-01-17 03:59:59.999999999', freq='H')),
 (Timestamp('2021-01-17 05:00:00', freq='H'),
  Timestamp('2021-01-17 06:59:59.999999999', freq='H')),
 (Timestamp('2021-01-18 17:00:00', freq='H'),
  Timestamp('2021-01-18 17:59:59.999999999', freq='H')),
 (Timestamp('2021-01-21 04:00:00', freq='H'),
  Timestamp('2021-01-21 04:59:59.999999999', freq='H')),
 (Timestamp('2021-01-23 20:00:00', freq='H'),
  Timestamp('2021-01-23 21:59:59.999999999', freq='H')),
 (Timestamp('2021-01-24 17:00:00', freq='H'),
  Timestamp('2021-01-24 17:59:59.999999999', freq='H')),
 (Timestamp('2021-01-25 01:00:00', freq='H'),
  Timestamp('2021-01-25 01:59:59.999999999', freq='H')),
 (Timestamp('2021-01-25 10:00:00', freq='H'),
  Timestamp('2021-01-25 10:59:59.999999999', freq='H')),
 (Timestamp('2021-01-26 05:00:00', freq='H'),
  Timestamp('2021-01-26 05:59:59.999999999', freq='H')),
 (Timestamp('2021-01-26 23:00:00', freq='H'),
  Timestamp('2021-01-26 23:59:59.999999999', freq='H')),
 (Timestamp('2021-01-27 01:00:00', freq='H'),
  Timestamp('2021-01-27 01:59:59.999999999', freq='H')),
 (Timestamp('2021-01-29 19:00:00', freq='H'),
  Timestamp('2021-01-29 19:59:59.999999999', freq='H')),
 (Timestamp('2021-01-30 01:00:00', freq='H'),
  Timestamp('2021-01-30 02:59:59.999999999', freq='H')),
 (Timestamp('2021-01-30 06:00:00', freq='H'),
  Timestamp('2021-01-30 06:59:59.999999999', freq='H')),
 (Timestamp('2021-01-30 09:00:00', freq='H'),
  Timestamp('2021-01-30 09:59:59.999999999', freq='H'))]
```



Potential “gaps” of LOF

When you have a small k-value (only looking at nearby points), the model is more erroneous when the data has a lot of noise

When you have a large k-value, you can often miss local outliers

Conclusion- the software does its best to pick the “perfect k-value” but there will be disadvantages either way

LSTM

LSTM

- LSTM models can process multivariate time-series data without the need for dimensionality reduction
- Dynamic Error Thresholds
  - Since environmental factors are currently changing, we need a fast, general, and unsupervised approach.
- Gaussian assumptions of past smoothed error allowed us to compare between new and prior errors
  - However, this approach can be problematic if a parametric assumption was violated
-

AutoEncoder

- An unsupervised Artificial Neural Network
- Attempts to encode the data by compressing it into the lower dimensions
- Decodes the data to reconstruct the original input
- The bottleneck layer (or code) holds the compressed representation of the input data
- The number of hidden units in the code is called code size

LSTM Autoencoder

This algorithm follows a similar structure to PCA because reconstruction errors are used to compute anomalies, but this model seems to be more accurate because the model learns the data representation through a more extensive training process. LSTM models utilize a cell state to learn the long term relationships of the data, if data is worth remembering it will pass through a sigmoid function and return a number closer to 1. But if it is worth forgetting, the sigmoid function will return a number closer to zero. The LSTM autoencoder takes LSTMs to another by using their layers as encoders/decoders. Typical autoencoders use encoder functions to decompress the data into a lower dimension and then use decoding functions to learn/recompute the decompressed data into the data's original dimension. So in an LSTM autoencoder, the first layer uses n neurons, the second layer uses n/2 neurons to decompress the input data, the third layer than repeats the vector to pass it on to the fourth layer and then the last layer recompresses the data into the original data dimension. In our results we observed PCA and the LSTM Autoencoder were the strongest models as it was able to learn the representation of the multivariate data as a whole. But the LSTM autoencoder uses more complex activation functions such as RELU compared to PCA which uses a linear function, which at the end had a greater impact to detect more anomaly time points.

As the LSTM autoencoder was our best performing model, the advantages from the other baseline algorithms were incorporated into the LSTM autoencoder to create an enhanced version which we will be showcased over the coming sections.

VAR

Vector Autoregression (Prediction)

- Each variable (time series) is modeled as a function of the past values
  - The predictors are nothing but the lags (time delated value) of the series
- Bi-directional
  - The variables influence one another
- Each variable is modeled as a linear combination of past values of itself and the past values of other variables in the system
- Advantages- quickly tests every possible combination of time series and their effects
- Disadvantages- does not work well if variables do not affect one another

How do we detect anomalies with VAR?

1. Use the AUgmented-Dickey Fuller test to test whether each column of the data is stationary or not
   a. If the p-value is less than the significance level than the data is stationary
   b. If not, convert the non-stationary data into stationary data using the “differencing” technique
2. Fit a time series model to model the relationships between the data using the Vector Auto-Regression model
3. Find the best lag for the data
   a. Whatever gives the minimum AIC value for the model
   b. In our case, it was lag = 3
4. The VAR model uses the lags of every column of the data as features and the columns in the provided data as targets
5. The VAR model is going to fit the generated features and fit the least-squares or linear regression by using every column of the data as targets separately
   a. the reason behind using this kind of method is the presence of autocorrelation in the data
6. Now by using the selected lag, fit the VAR model and find the squared errors of the data
7. The squared errors are then used to find the threshold, **above which the observations are considered to be anomalies**
8. This is the method to find the threshold- threshold = mean(squared_errors) + z * standard_deviation(squared_errors)

VAR in Python

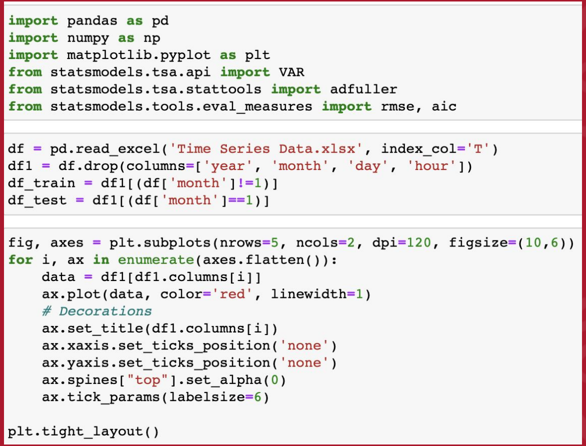
1. Analyze the time series characteristics
2. Test for causation amongst the time series
3. Test for stationarity
4. Transform the series to make it stationary, if needed
5. Find optimal order (p)
6. Prepare training and test datasets
7. Train the model
8. Roll back the transformations, if any.
9. Evaluate the model using test set
10. Forecast to future

System of equations for VAR with two time series

$$
\begin{aligned}
Y_{1,t} &= \alpha_1 + \beta_{11,1} Y_{1,t-1} + \beta_{12,1} Y_{2,t-1} + \epsilon_{1,t} \\
Y_{2,t} &= \alpha_2 + \beta_{21,1} Y_{1,t-1} + \beta_{22,1} Y_{2,t-1} + \epsilon_{2,t}
\end{aligned}
$$

How to detect anomalies with VAR

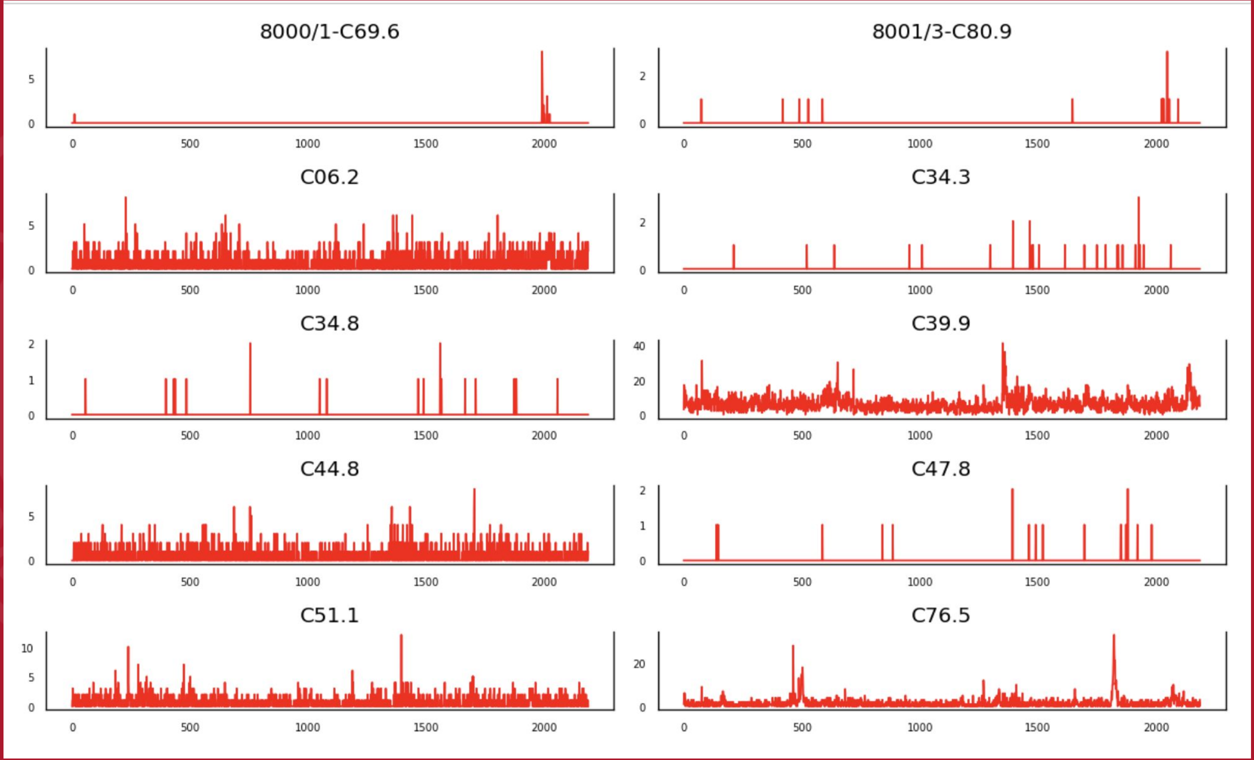
- Conduct an ADF test to check whether the data is stationary or not. If the data is not stationary convert the data into stationary data.
- After converting the data into stationary data, fit a time-series model to model the relationship between the data.
- Find the squared residual errors for each observation and find a threshold for those squared errors.
- Any observation’s squared error exceeding the threshold can be marked as an anomaly.

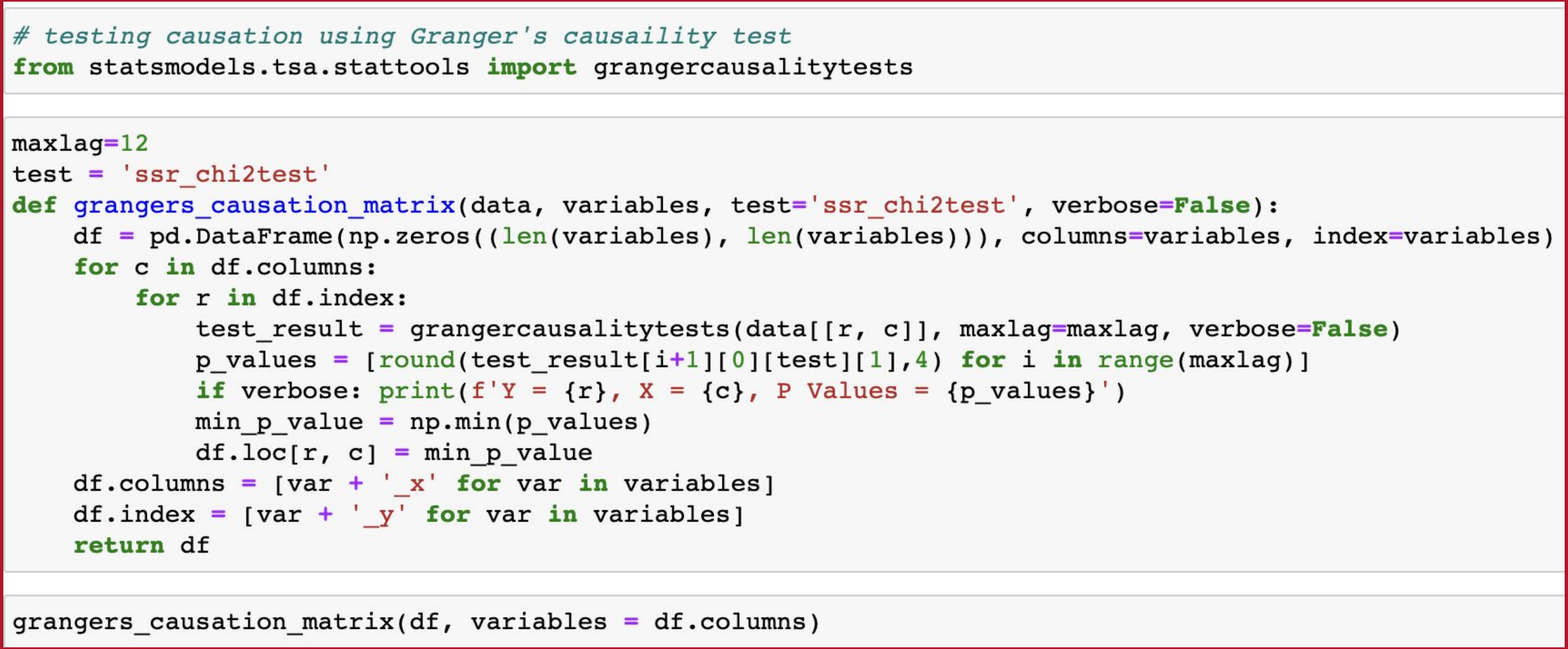


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.api import VAR
from statsmodels.tsa.stattools import adfuller
from statsmodels.tools.eval_measures import rmse, aic
```

```
df = pd.read_excel('Time Series Data.xlsx', index_col='T')
df1 = df.drop(columns=['year', 'month', 'day', 'hour'])
df_train = df1[(df['month']!=1)]
df_test = df1[(df['month']==1)]
```

```
fig, axes = plt.subplots(nrows=5, ncols=2, dpi=120, figsize=(10,6))
for i, ax in enumerate(axes.flatten()):
    data = df1[df1.columns[i]]
    ax.plot(data, color='red', linewidth=1)
    # Decorations
    ax.set_title(df1.columns[i])
    ax.xaxis.set_ticks_position('none')
    ax.yaxis.set_ticks_position('none')
    ax.spines["top"].set_alpha(0)
    ax.tick_params(labelsize=6)

plt.tight_layout()
```



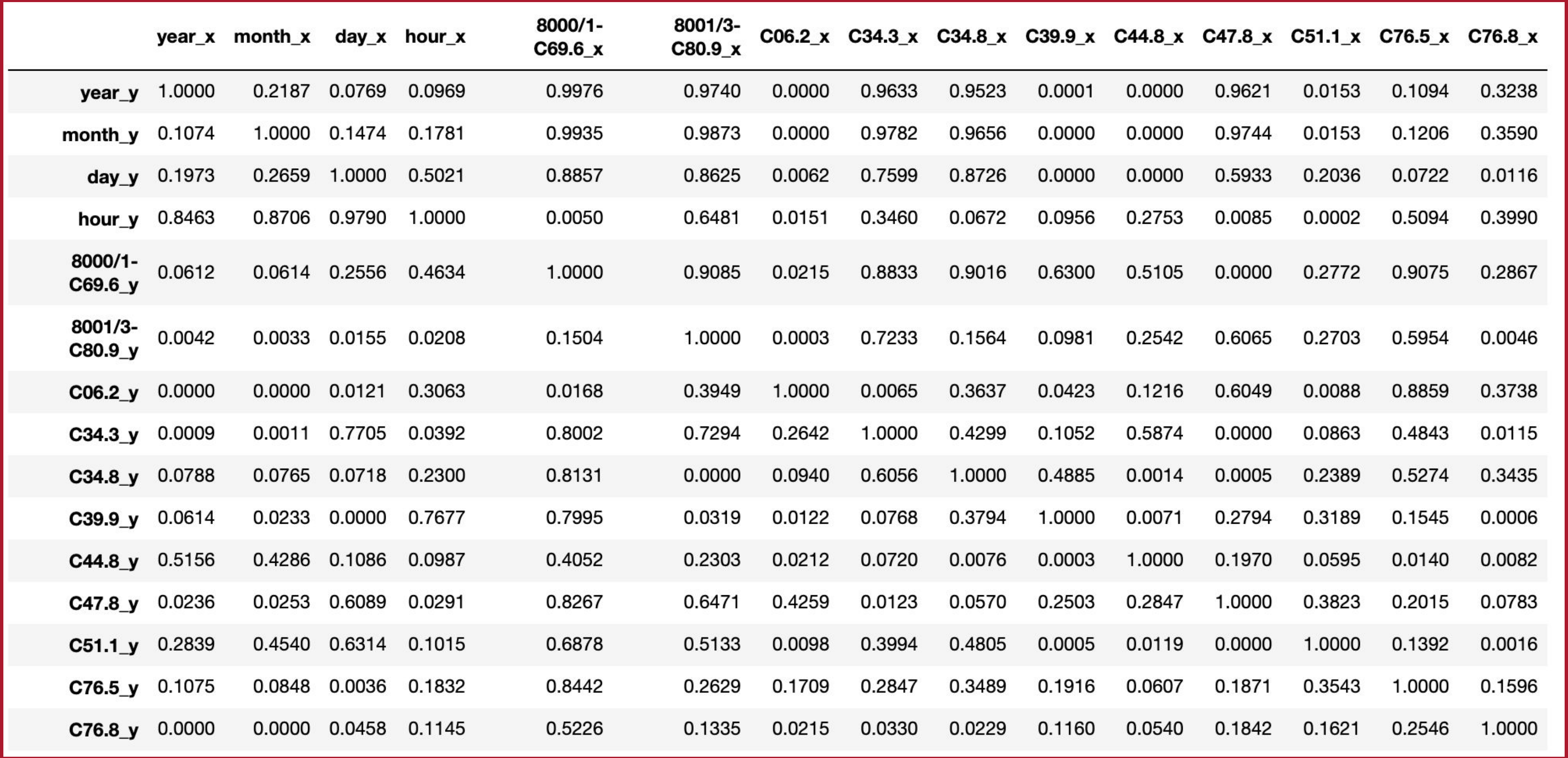


```
# testing causation using Granger's causaility test
from statsmodels.tsa.stattools import grangercausalitytests
```

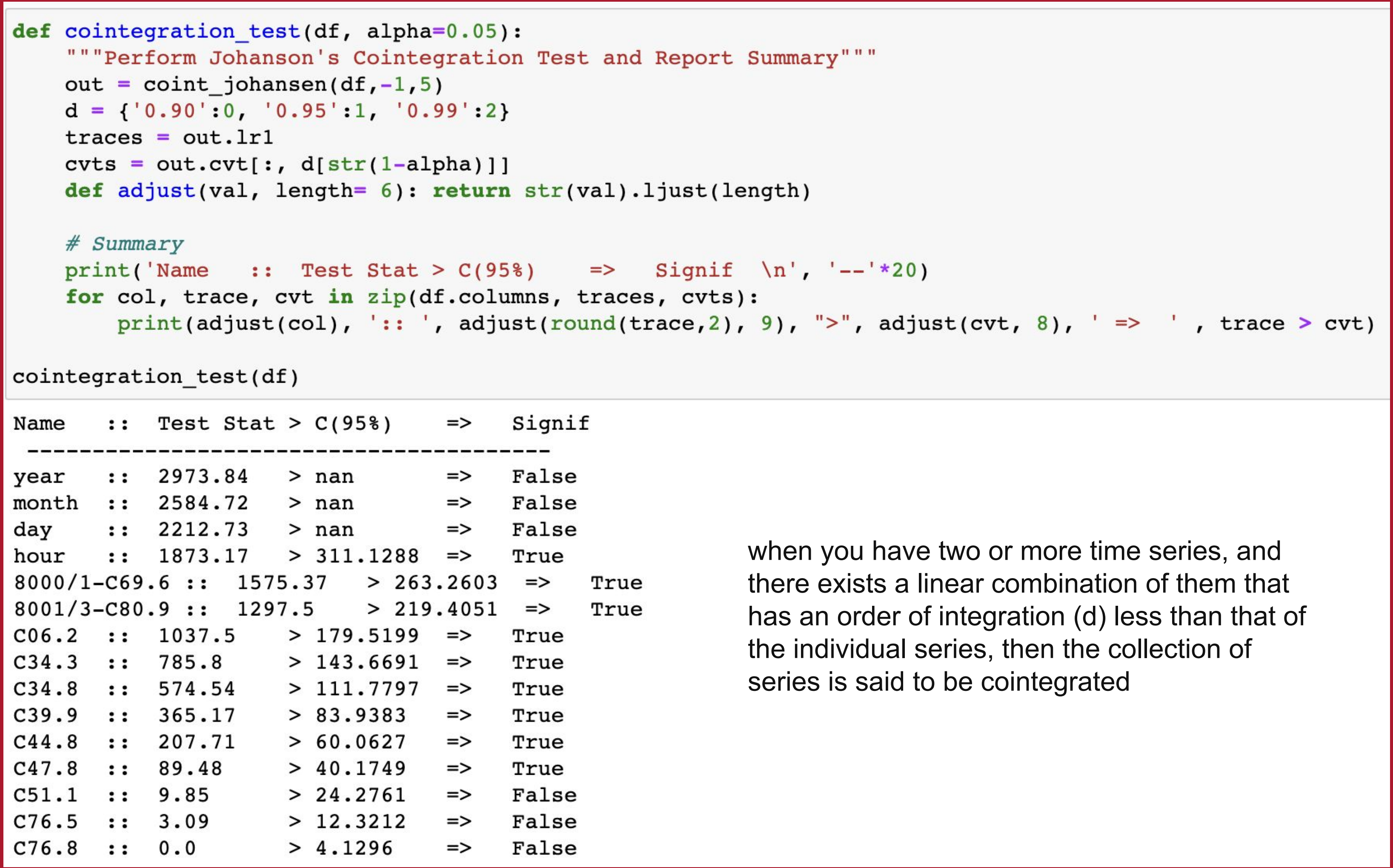
```
maxlag=12
test = 'ssr_chi2test'
def grangers_causation_matrix(data, variables, test='ssr_chi2test', verbose=False):
    df = pd.DataFrame(np.zeros((len(variables), len(variables))), columns=variables, index=variables)
    for c in df.columns:
        for r in df.index:
            test_result = grangercausalitytests(data[[r, c]], maxlag=maxlag, verbose=False)
            p_values = [round(test_result[i+1][0][test][1],4) for i in range(maxlag)]
            if verbose: print(f'Y = {r}, X = {c}, P Values = {p_values}')
            min_p_value = np.min(p_values)
            df.loc[r, c] = min_p_value
    df.columns = [var + '_x' for var in variables]
    df.index = [var + '_y' for var in variables]
    return df
```

```
grangers_causation_matrix(df, variables = df.columns)
```

Causality Test Results



	year_x	month_x	day_x	hour_x	8000/1-C69.6_x	8001/3-C80.9_x	C06.2_x	C34.3_x	C34.8_x	C39.9_x	C44.8_x	C47.8_x	C51.1_x	C76.5_x	C76.8_x
year_y	1.0000	0.2187	0.0769	0.0969	0.9976	0.9740	0.0000	0.9633	0.9523	0.0001	0.0000	0.9621	0.0153	0.1094	0.3238
month_y	0.1074	1.0000	0.1474	0.1781	0.9935	0.9873	0.0000	0.9782	0.9656	0.0000	0.0000	0.9744	0.0153	0.1206	0.3590
day_y	0.1973	0.2659	1.0000	0.5021	0.8857	0.8625	0.0062	0.7599	0.8726	0.0000	0.0000	0.5933	0.2036	0.0722	0.0116
hour_y	0.8463	0.8706	0.9790	1.0000	0.0050	0.6481	0.0151	0.3460	0.0672	0.0956	0.2753	0.0085	0.0002	0.5094	0.3990
8000/1-C69.6_y	0.0612	0.0614	0.2556	0.4634	1.0000	0.9085	0.0215	0.8833	0.9016	0.6300	0.5105	0.0000	0.2772	0.9075	0.2867
8001/3-C80.9_y	0.0042	0.0033	0.0155	0.0208	0.1504	1.0000	0.0003	0.7233	0.1564	0.0981	0.2542	0.6065	0.2703	0.5954	0.0046
C06.2_y	0.0000	0.0000	0.0121	0.3063	0.0168	0.3949	1.0000	0.0065	0.3637	0.0423	0.1216	0.6049	0.0088	0.8859	0.3738
C34.3_y	0.0009	0.0011	0.7705	0.0392	0.8002	0.7294	0.2642	1.0000	0.4299	0.1052	0.5874	0.0000	0.0863	0.4843	0.0115
C34.8_y	0.0788	0.0765	0.0718	0.2300	0.8131	0.0000	0.0940	0.6056	1.0000	0.4885	0.0014	0.0005	0.2389	0.5274	0.3435
C39.9_y	0.0614	0.0233	0.0000	0.7677	0.7995	0.0319	0.0122	0.0768	0.3794	1.0000	0.0071	0.2794	0.3189	0.1545	0.0006
C44.8_y	0.5156	0.4286	0.1086	0.0987	0.4052	0.2303	0.0212	0.0720	0.0076	0.0003	1.0000	0.1970	0.0595	0.0140	0.0082
C47.8_y	0.0236	0.0253	0.6089	0.0291	0.8267	0.6471	0.4259	0.0123	0.0570	0.2503	0.2847	1.0000	0.3823	0.2015	0.0783
C51.1_y	0.2839	0.4540	0.6314	0.1015	0.6878	0.5133	0.0098	0.3994	0.4805	0.0005	0.0119	0.0000	1.0000	0.1392	0.0016
C76.5_y	0.1075	0.0848	0.0036	0.1832	0.8442	0.2629	0.1709	0.2847	0.3489	0.1916	0.0607	0.1871	0.3543	1.0000	0.1596
C76.8_y	0.0000	0.0000	0.0458	0.1145	0.5226	0.1335	0.0215	0.0330	0.0229	0.1160	0.0540	0.1842	0.1621	0.2546	1.0000

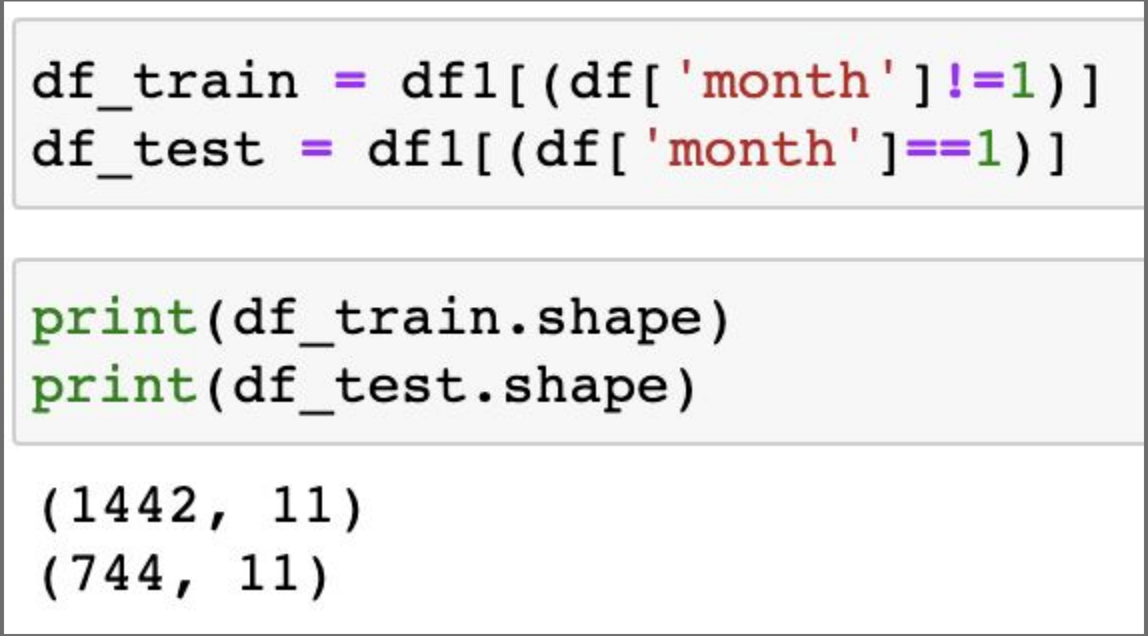


```python
def cointegration_test(df, alpha=0.05):
    """Perform Johanson's Cointegration Test and Report Summary"""
    out = coint_johansen(df,-1,5)
    d = {'0.90':0, '0.95':1, '0.99':2}
    traces = out.lr1
    cvts = out.cvt[:, d[str(1-alpha)]]
    def adjust(val, length= 6): return str(val).ljust(length)

    # Summary
    print('Name   ::  Test Stat > C(95%)    =>   Signif  \n', '--'*20)
    for col, trace, cvt in zip(df.columns, traces, cvts):
        print(adjust(col), ':: ', adjust(round(trace,2), 9), ">", adjust(cvt, 8), ' =>  ' , trace > cvt)

cointegration_test(df)
```

```
Name   ::  Test Stat > C(95%)    =>   Signif  
 ----------------------------------------
year   ::  2973.84   > nan       =>   False
month  ::  2584.72   > nan       =>   False
day    ::  2212.73   > nan       =>   False
hour   ::  1873.17   > 311.1288  =>   True
8000/1-C69.6 ::  1575.37   > 263.2603  =>   True
8001/3-C80.9 ::  1297.5    > 219.4051  =>   True
C06.2  ::  1037.5    > 179.5199  =>   True
C34.3  ::  785.8     > 143.6691  =>   True
C34.8  ::  574.54    > 111.7797  =>   True
C39.9  ::  365.17    > 83.9383   =>   True
C44.8  ::  207.71    > 60.0627   =>   True
C47.8  ::  89.48     > 40.1749   =>   True
C51.1  ::  9.85      > 24.2761   =>   False
C76.5  ::  3.09      > 12.3212   =>   False
C76.8  ::  0.0       > 4.1296    =>   False
```

when you have two or more time series, and there exists a linear combination of them that has an order of integration (d) less than that of the individual series, then the collection of series is said to be cointegrated

Split the data into training and testing data



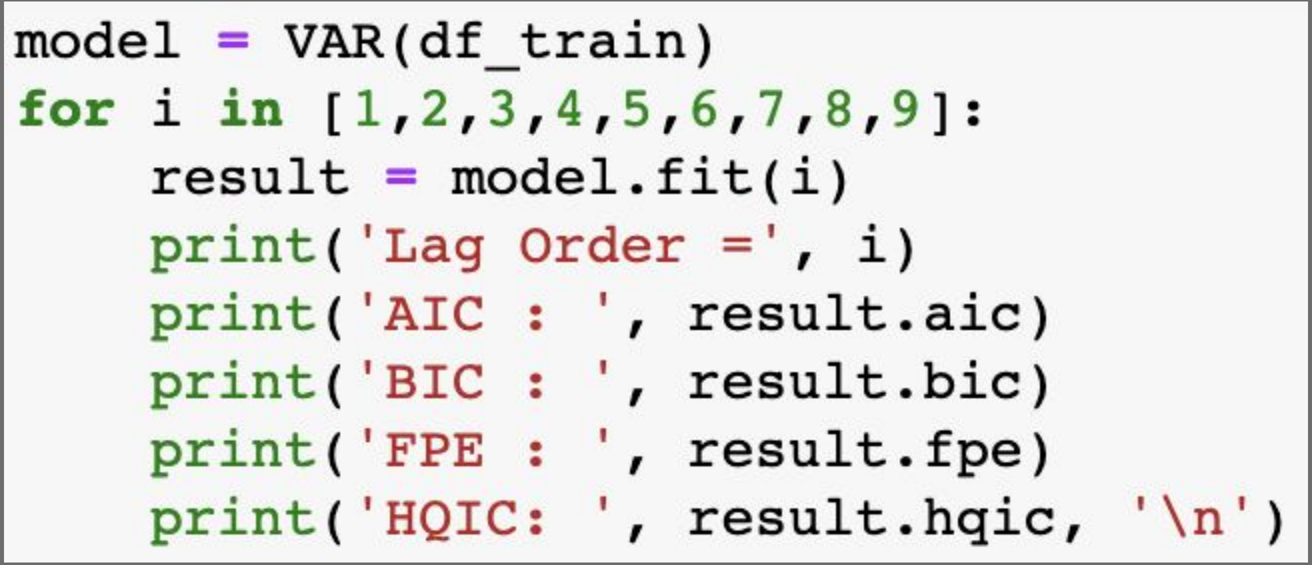
```
df_train = df1[(df['month']!=1)]
df_test = df1[(df['month']==1)]
```

```
print(df_train.shape)
print(df_test.shape)
```

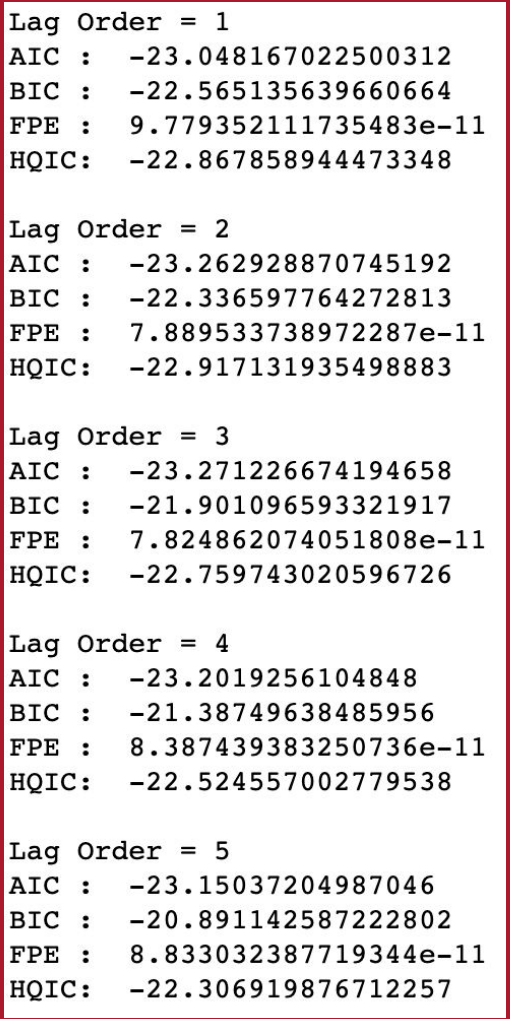
```
(1442, 11)
(744, 11)
```

Select the Order (P) of VAR Model

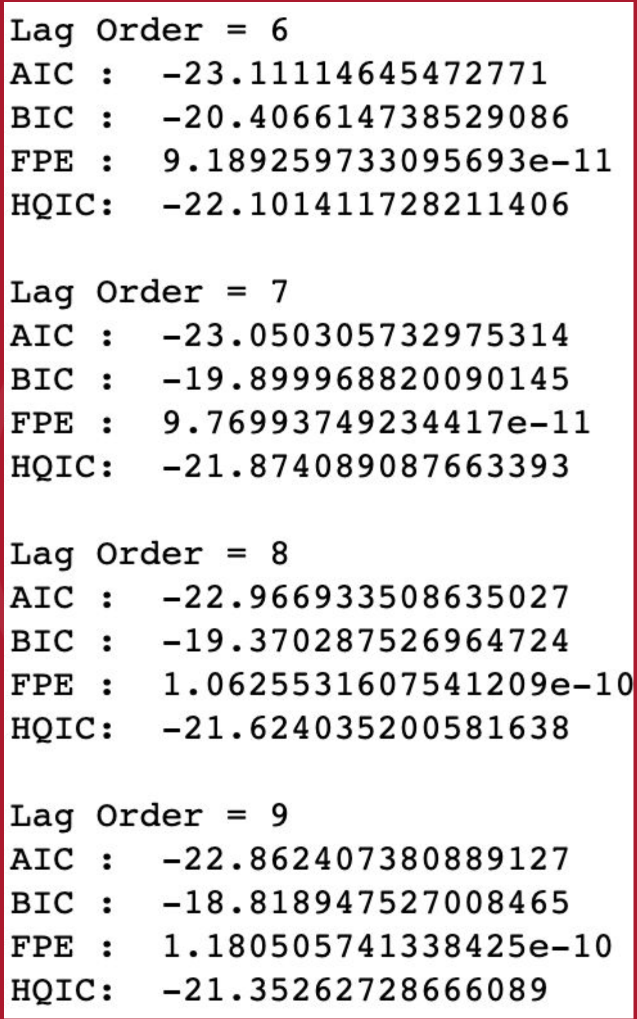
iteratively fit increasing orders of VAR model and pick the order that gives a model with least AIC



```python
model = VAR(df_train)
for i in [1,2,3,4,5,6,7,8,9]:
    result = model.fit(i)
    print('Lag Order =', i)
    print('AIC : ', result.aic)
    print('BIC : ', result.bic)
    print('FPE : ', result.fpe)
    print('HQIC: ', result.hqic, '\n')
```



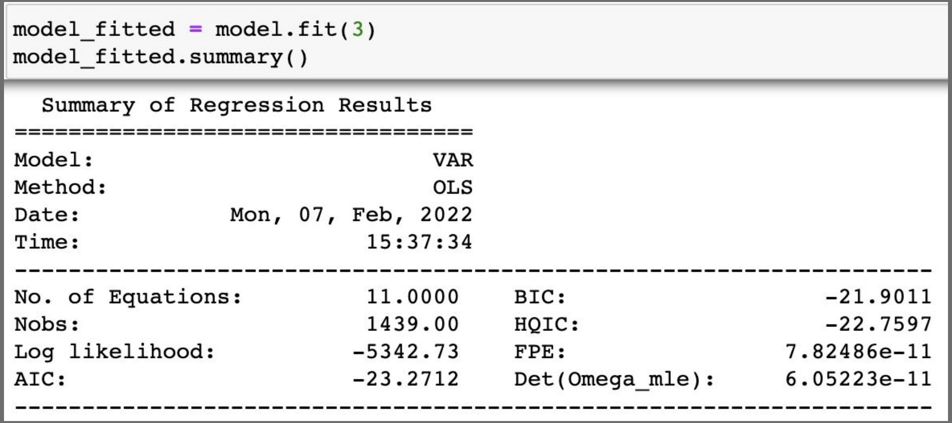
```
Lag Order = 1
AIC :  -23.048167022500312
BIC :  -22.565135639660664
FPE :  9.779352111735483e-11
HQIC:  -22.867858944473348

Lag Order = 2
AIC :  -23.262928870745192
BIC :  -22.336597764272813
FPE :  7.889533738972287e-11
HQIC:  -22.917131935498883

Lag Order = 3
AIC :  -23.271226674194658
BIC :  -21.901096593321917
FPE :  7.824862074051808e-11
HQIC:  -22.759743020596726

Lag Order = 4
AIC :  -23.2019256104848
BIC :  -21.38749638485956
FPE :  8.387439383250736e-11
HQIC:  -22.524557002779538

Lag Order = 5
AIC :  -23.15037204987046
BIC :  -20.891142587222802
FPE :  8.833032387719344e-11
HQIC:  -22.306919876712257
```



```
Lag Order = 6
AIC :  -23.11114645472771
BIC :  -20.406614738529086
FPE :  9.189259733095693e-11
HQIC:  -22.10141172821406

Lag Order = 7
AIC :  -23.050305732975314
BIC :  -19.899968820090145
FPE :  9.769937492344417e-11
HQIC:  -21.874089087663393

Lag Order = 8
AIC :  -22.966933508635027
BIC :  -19.370287526964724
FPE :  1.0625531607541209e-10
HQIC:  -21.624035200581638

Lag Order = 9
AIC :  -22.862407380889127
BIC :  -18.818947527008465
FPE :  1.180505741338425e-10
HQIC:  -21.3526272866089
```

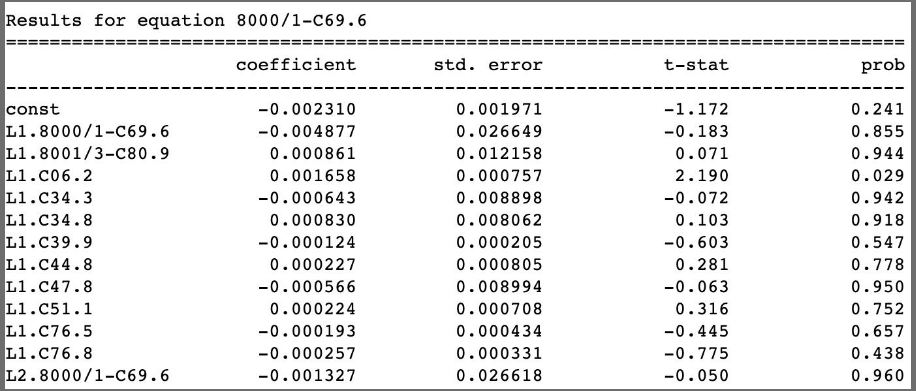
AIC drops to lowest at lag 3, then increases then drops again, so we are going to go with 3

Train the VAR Model

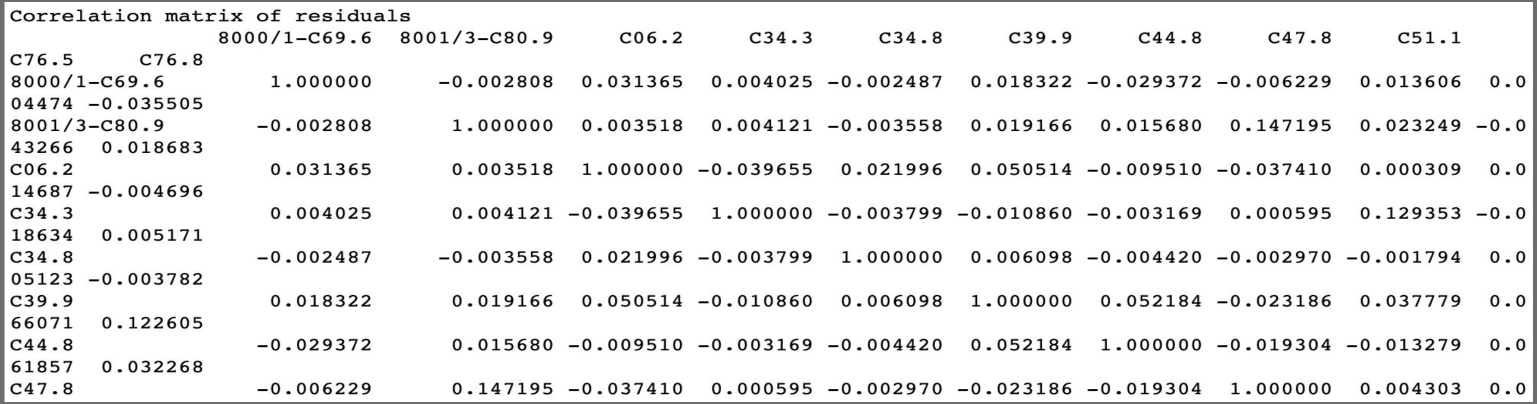


```
model_fitted = model.fit(3)
model_fitted.summary()
```

```
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Mon, 07, Feb, 2022
Time:                     15:37:34
--------------------------------------------------------------------
No. of Equations:         11.0000    BIC:                  -21.9011
Nobs:                     1439.00    HQIC:                 -22.7597
Log likelihood:          -5342.73    FPE:               7.82486e-11
AIC:                     -23.2712    Det(Omega_mle):    6.05223e-11
--------------------------------------------------------------------
```



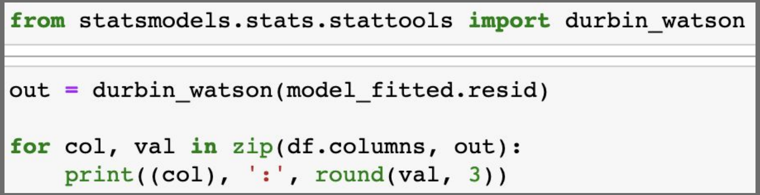
```
Results for equation 8000/1-C69.6
==============================================================================
                   coefficient       std. error           t-stat          prob
------------------------------------------------------------------------------
const                -0.002310         0.001971           -1.172         0.241
L1.8000/1-C69.6      -0.004877         0.026649           -0.183         0.855
L1.8001/3-C80.9       0.000861         0.012158            0.071         0.944
L1.C06.2              0.001658         0.000757            2.190         0.029
L1.C34.3             -0.000643         0.008898           -0.072         0.942
L1.C34.8              0.000830         0.008062            0.103         0.918
L1.C39.9             -0.000124         0.000205           -0.603         0.547
L1.C44.8              0.000227         0.000805            0.281         0.778
L1.C47.8             -0.000566         0.008994           -0.063         0.950
L1.C51.1              0.000224         0.000708            0.316         0.752
L1.C76.5             -0.000193         0.000434           -0.445         0.657
L1.C76.8             -0.000257         0.000331           -0.775         0.438
L2.8000/1-C69.6      -0.001327         0.026618           -0.050         0.960
```



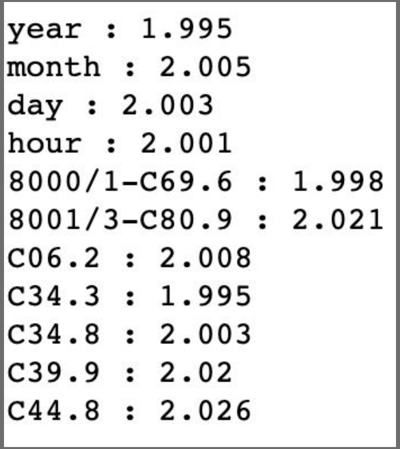
```
Correlation matrix of residuals
                8000/1-C69.6  8001/3-C80.9     C06.2     C34.3     C34.8     C39.9     C44.8     C47.8     C51.1
C76.5     C76.8
8000/1-C69.6        1.000000     -0.002808  0.031365  0.004025 -0.002487  0.018322 -0.029372 -0.006229  0.013606  0.0
04474 -0.035505
8001/3-C80.9       -0.002808      1.000000  0.003518  0.004121 -0.003558  0.019166  0.015680  0.147195  0.023249 -0.0
43266  0.018683
C06.2               0.031365      0.003518  1.000000 -0.039655  0.021996  0.050514 -0.009510 -0.037410  0.000309  0.0
14687 -0.004696
C34.3               0.004025      0.004121 -0.039655  1.000000 -0.003799 -0.010860 -0.003169  0.000595  0.129353 -0.0
18634  0.005171
C34.8              -0.002487     -0.003558  0.021996 -0.003799  1.000000  0.006098 -0.004420 -0.002970 -0.001794  0.0
05123 -0.003782
C39.9               0.018322      0.019166  0.050514 -0.010860  0.006098  1.000000  0.052184 -0.023186  0.037779  0.0
66071  0.122605
C44.8              -0.029372      0.015680 -0.009510 -0.003169 -0.004420  0.052184  1.000000 -0.019304 -0.013279  0.0
61857  0.032268
C47.8              -0.006229      0.147195 -0.037410  0.000595 -0.002970 -0.023186 -0.019304  1.000000  0.004303  0.0
```

Check for Serial Correlation of Residuals (Errors) using Durbin Watson Statistic

- Serial correlation of residuals is used to check if there is any leftover pattern in the residuals (errors).
- If there is any correlation left in the residuals, then, there is some pattern in the time series that is still left to be explained by the model.
- In that case, the typical course of action is to either increase the order of the model or induce more predictors into the system or look for a different algorithm to model the time series



```
from statsmodels.stats.stattools import durbin_watson

out = durbin_watson(model_fitted.resid)

for col, val in zip(df.columns, out):
    print((col), ':', round(val, 3))
```

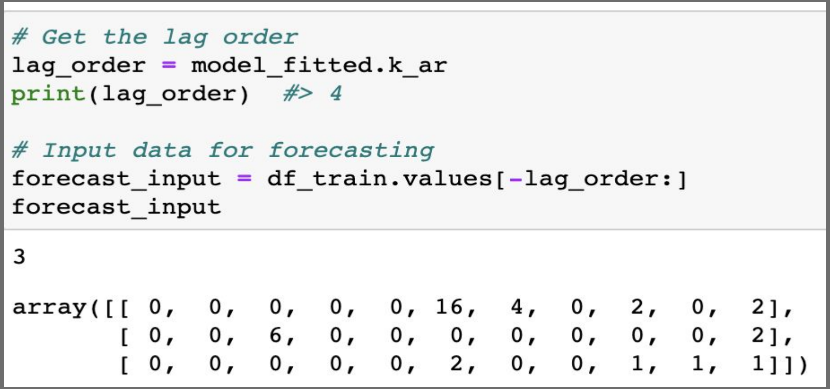


```
year : 1.995
month : 2.005
day : 2.003
hour : 2.001
8000/1-C69.6 : 1.998
8001/3-C80.9 : 2.021
C06.2 : 2.008
C34.3 : 1.995
C34.8 : 2.003
C39.9 : 2.02
C44.8 : 2.026
```

The closer the values are to 2, there is no significant serial correlation. So we can move on!

How to Forecast VAR model using statsmodels

In order to forecast, the VAR model expects up to the lag order number of observations from the past data. This is because, the terms in the VAR model are essentially the lags of the various time series in the dataset, so you need to provide it as many of the previous values as indicated by the lag order used by the model.



```
# Get the lag order
lag_order = model_fitted.k_ar
print(lag_order)  #> 4

# Input data for forecasting
forecast_input = df_train.values[-lag_order:]
forecast_input
```

```
3

array([[ 0,   0,   0,   0,   0, 16,   4,   0,   2,   0,   2],
       [ 0,   0,   6,   0,   0,   0,   0,   0,   0,   0,   2],
       [ 0,   0,   0,   0,   0,   2,   0,   0,   1,   1,   1]])
```