Noisy Panel Data Puzzle

December 5, 2018

1 BlackRock Noisy Panel Data Puzzle

In [30]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import statsmodels.api as sm
import statsmodels.tsa.stattools as ts
from pandas import Series
from matplotlib import pyplot
from statsmodels.tsa.ar_model import AR
from sklearn.metrics import mean_squared_error
warnings.filterwarnings("ignore")
from sklearn.preprocessing import MinMaxScaler
from sklearn import metrics
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
```

In [31]:
```
import os

os.chdir('/Users/zimingwang/desktop/2018/BlackRock')
data = pd.read_csv('data.csv', index_col=0)
data2 = pd.read_csv('data.csv', index_col=0)
```

In [32]:
```
# A brief summary of the data
data.describe()
```

Out[32]:
```
          sig.0001     sig.0002     sig.0003     sig.0004     sig.0005  \
count  2557.000000  2557.000000  2557.000000  2557.000000  2557.000000
mean     -0.009245     0.010331     0.023678     0.003693    -0.028997
std       1.394840     1.488754     1.386090     1.339229     1.637783
min      -5.921308    -7.034904    -5.008178    -6.354693    -7.465328
25%      -0.861943    -0.817749    -0.847725    -0.818705    -1.059241
50%      -0.020086     0.027081     0.027991     0.029008    -0.012326
```

```
75%       0.863059     0.879645     0.843743     0.823281     1.028542
max       6.122230     6.468382     6.776746     5.770354     7.195405

            sig.0006     sig.0007     sig.0008     sig.0009     sig.0010  \
count  2557.000000  2557.000000  2557.000000  2557.000000  2557.000000
mean      0.044313    -0.016797    -0.022324     0.004028     0.021681
std       1.354947     1.340255     1.257658     1.315804     1.365348
min      -6.060215    -7.657676    -4.997675    -5.452313    -6.155391
25%      -0.731308    -0.816357    -0.769304    -0.799528    -0.810633
50%       0.049240     0.038389    -0.026474    -0.001311     0.004310
75%       0.845257     0.775758     0.744726     0.779883     0.903928
max       7.140175     5.506364     6.742964     6.035447     5.757190

            ...        sig.0041     sig.0042     sig.0043     sig.0044  \
count       ...     2557.000000  2557.000000  2557.000000  2557.000000
mean        ...        0.020237     0.011441    -0.023725     0.008983
std         ...        1.399607     1.308771     1.288299     1.303261
min         ...       -7.093790    -6.546104    -6.016546    -5.397768
25%         ...       -0.785135    -0.774184    -0.799420    -0.764266
50%         ...        0.028630    -0.006124    -0.010779     0.009067
75%         ...        0.850549     0.797428     0.736082     0.772923
max         ...        6.731320     6.291640     6.732405     6.589563

            sig.0045     sig.0046     sig.0047     sig.0048     sig.0049  \
count  2557.000000  2557.000000  2557.000000  2557.000000  2557.000000
mean      0.018113     0.012419    -0.037271    -0.011683    -0.021042
std       1.337057     1.414719     1.315358     1.309446     1.415553
min      -5.353433    -6.812222    -7.024190    -7.112602    -6.431593
25%      -0.831780    -0.811434    -0.773439    -0.771524    -0.875673
50%       0.014089     0.020601    -0.022046    -0.027497    -0.034103
75%       0.830497     0.867158     0.710916     0.759456     0.823776
max       6.773119     7.712463     6.171996     5.489231     6.791841

            sig.0050
count  2557.000000
mean      0.021894
std       1.359027
min      -6.409285
25%      -0.771622
50%       0.019834
75%       0.873485
max       6.682983

[8 rows x 50 columns]
```

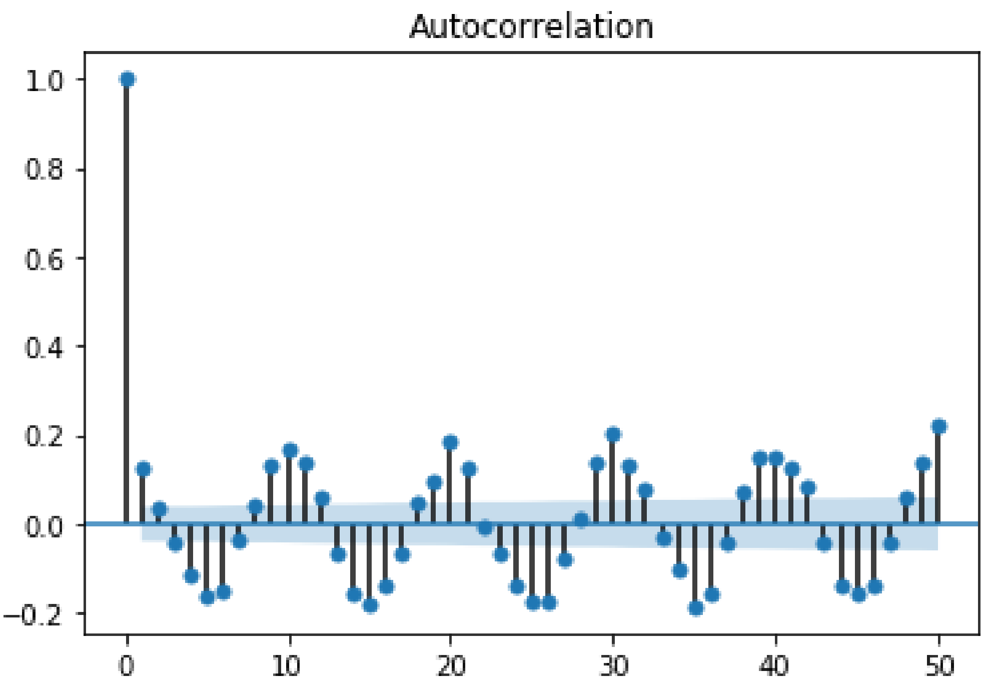
In [22]:
```
# Augmented Dickey-Fuller test
# The first number is the test statistics
# Compare it with thresholds at 1% and 5%, we can see they are much smaller
```

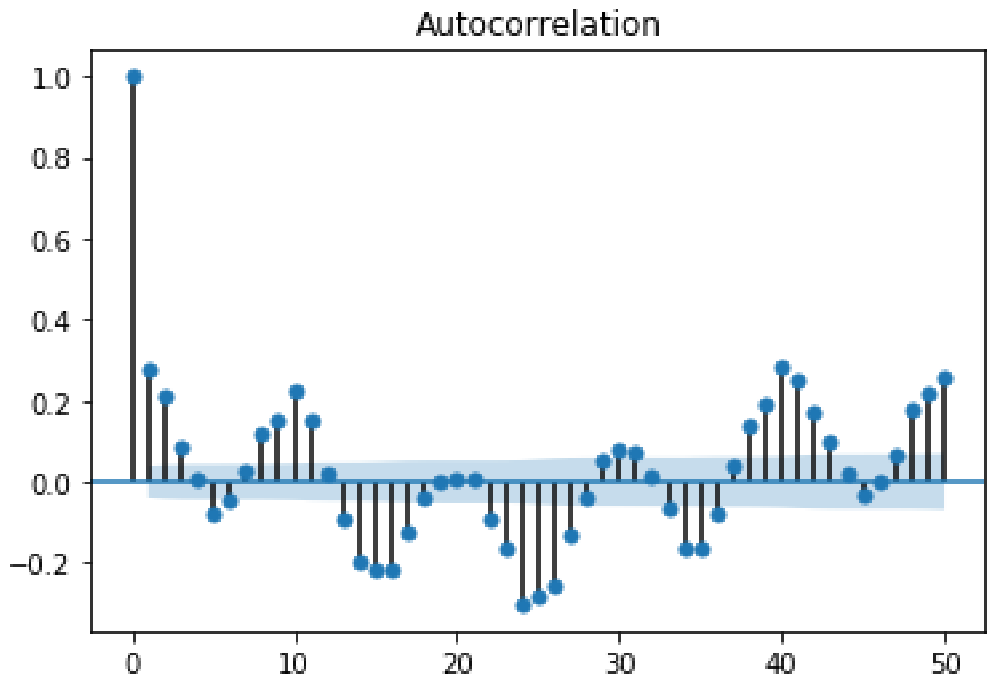
```python
for i in range (1,51):
    if i < 10:
        col_name = 'sig.000' + str(i)
    if i > 9:
        col_name = 'sig.00' + str(i)
    series = data[col_name].tolist()
    x = np.array(series)
    result = ts.adfuller(x)[:2]
    print(i,result)
```

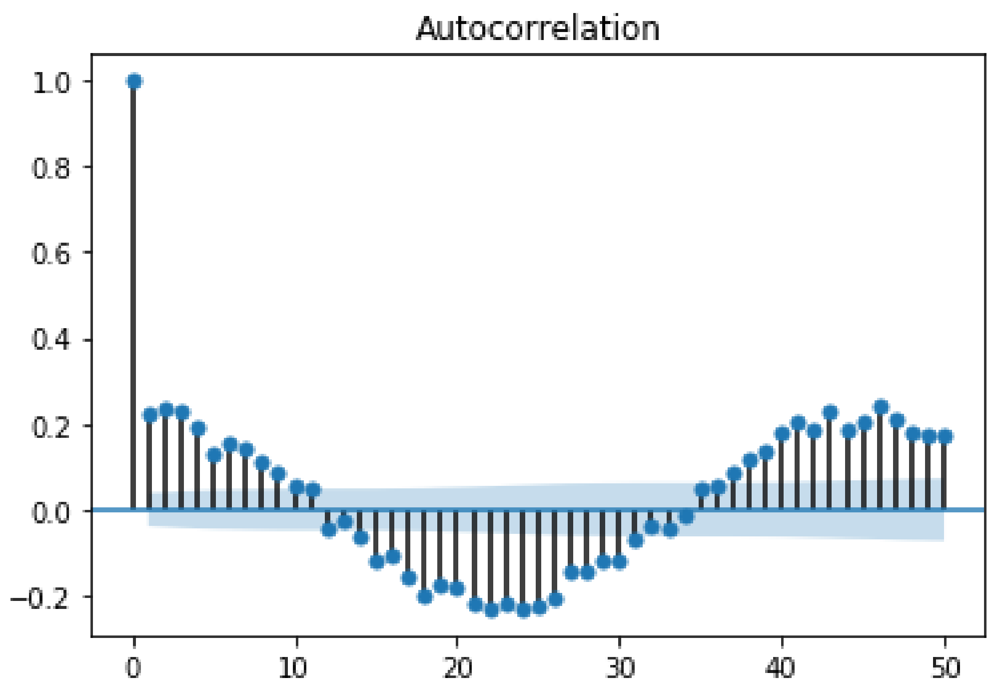
```
1 (-13.039450341076908, 2.257090011448472e-24)
2 (-17.94875240382422, 2.843571428505801e-30)
3 (-17.37353912642121, 5.1212283801555156e-30)
4 (-10.278275174898152, 3.839179158241712e-18)
5 (-18.22229849125247, 2.3782178865939708e-30)
6 (-15.225792141455138, 5.406272799537462e-28)
7 (-11.039496442891055, 5.4107759297503854e-20)
8 (-11.74982760336086, 1.2166566137401713e-21)
9 (-14.228561618302681, 1.61135337380826e-26)
10 (-17.07213779609672, 7.811956745846123e-30)
11 (-49.646905348571195, 0.0)
12 (-14.052747809924835, 3.1561516447216746e-26)
13 (-18.05909763499761, 2.6250458121883563e-30)
14 (-13.964193516742613, 4.463629020735996e-26)
15 (-11.704933835202409, 1.5369490253796701e-21)
16 (-13.09167110227606, 1.7832325034436154e-24)
17 (-12.700689785394847, 1.0815727980788214e-23)
18 (-17.45153479999921, 4.649829324221467e-30)
19 (-12.821259800335513, 6.146195575060705e-24)
20 (-10.968585246053722, 7.986473075472122e-20)
21 (-14.769141313997638, 2.33720643531162e-27)
22 (-15.467818539780527, 2.6538772478754668e-28)
23 (-16.06198965069681, 5.625877003594551e-29)
24 (-16.50602401047954, 2.1270993752458858e-29)
25 (-9.875312050672132, 3.8966361193639565e-17)
26 (-20.80401782487033, 0.0)
27 (-16.21935093688156, 3.912413565039553e-29)
28 (-48.62776718542934, 0.0)
29 (-11.483913399494895, 4.918286604496913e-21)
30 (-18.08503353318518, 2.5801510858523328e-30)
31 (-33.96766319579408, 0.0)
32 (-11.236251479272166, 1.8542244549592462e-20)
33 (-12.139478659677051, 1.663764767458479e-22)
34 (-13.021822152668976, 2.444857256868468e-24)
35 (-22.485238201715458, 0.0)
36 (-51.43548631292475, 0.0)
37 (-10.96189350080241, 8.286118414163923e-20)
38 (-15.539444831986554, 2.1688105051499216e-28)
```

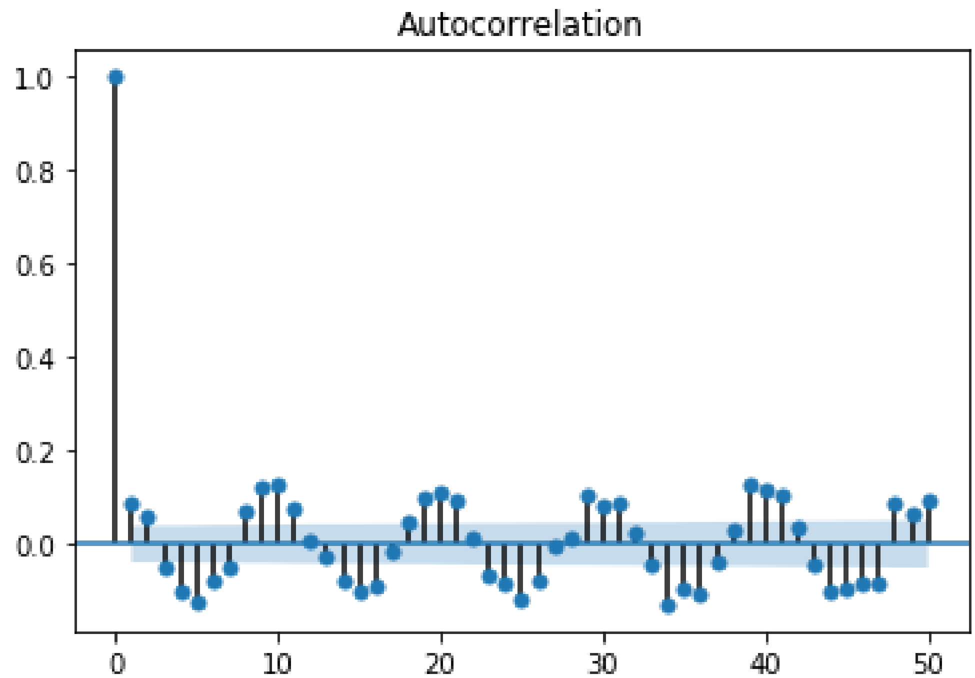
```
39 (-50.274821390447826, 0.0)
40 (-18.748078625450777, 2.0280791668308145e-30)
41 (-16.96636145429634, 9.228025048514014e-30)
42 (-11.566452335371944, 3.1777316875561197e-21)
43 (-13.022489214535412, 2.4374674365423133e-24)
44 (-10.113643315777027, 9.853437905320974e-18)
45 (-12.852982254982436, 5.304166010419043e-24)
46 (-15.490187160506641, 2.4906799905871167e-28)
47 (-14.651127250552245, 3.499102116897145e-27)
48 (-10.824239030558603, 1.7735770272710052e-19)
49 (-18.367551525681428, 2.221292590823693e-30)
50 (-15.15394501239892, 6.735975242383786e-28)
```

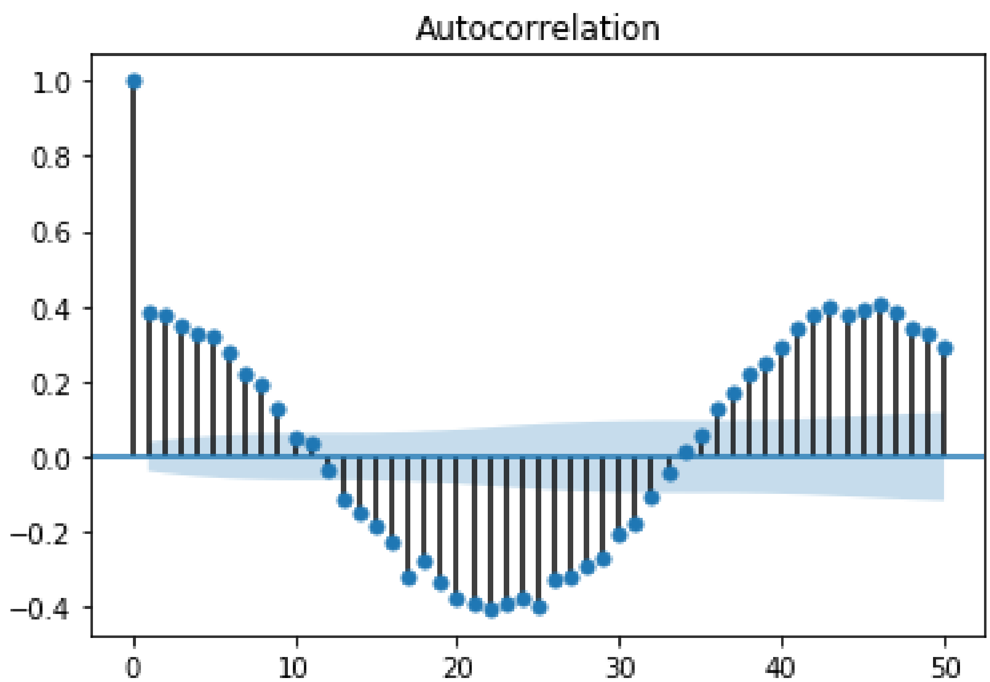
In [18]:
```
from statsmodels.graphics.tsaplots import plot_acf
for i in list(data2):
    series = data[[str(i)]]
    plot_acf(series, lags=50)
    pyplot.show()
```

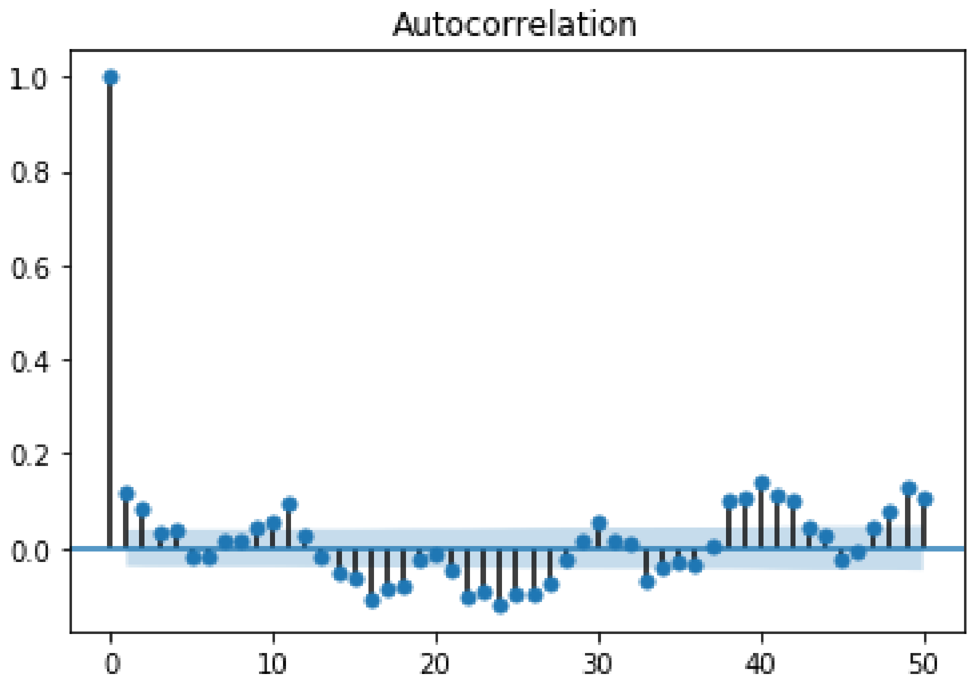



Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

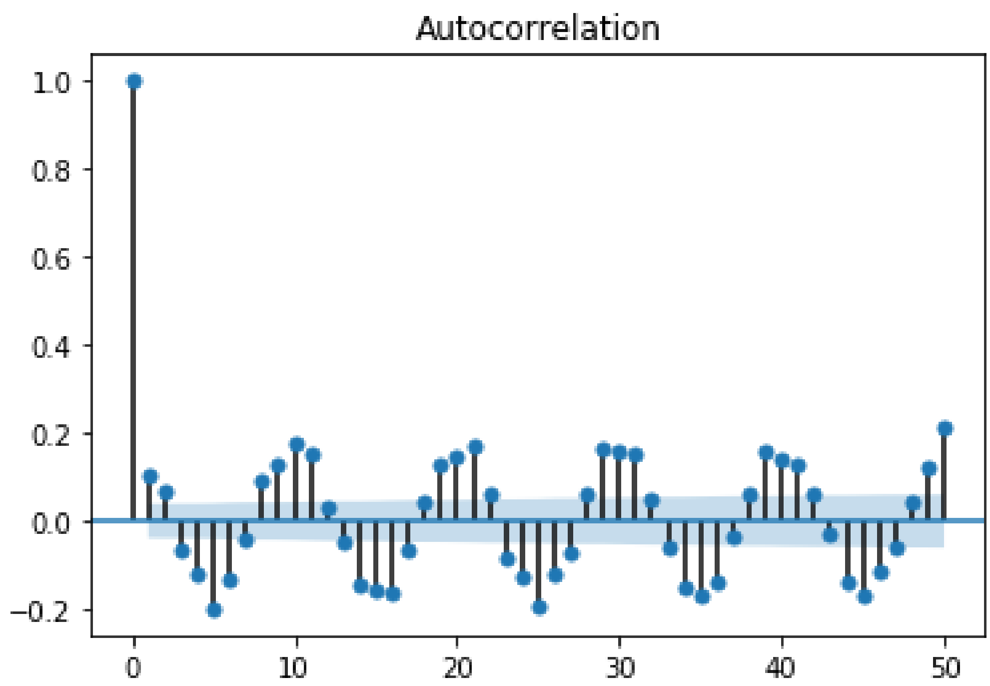

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50


Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

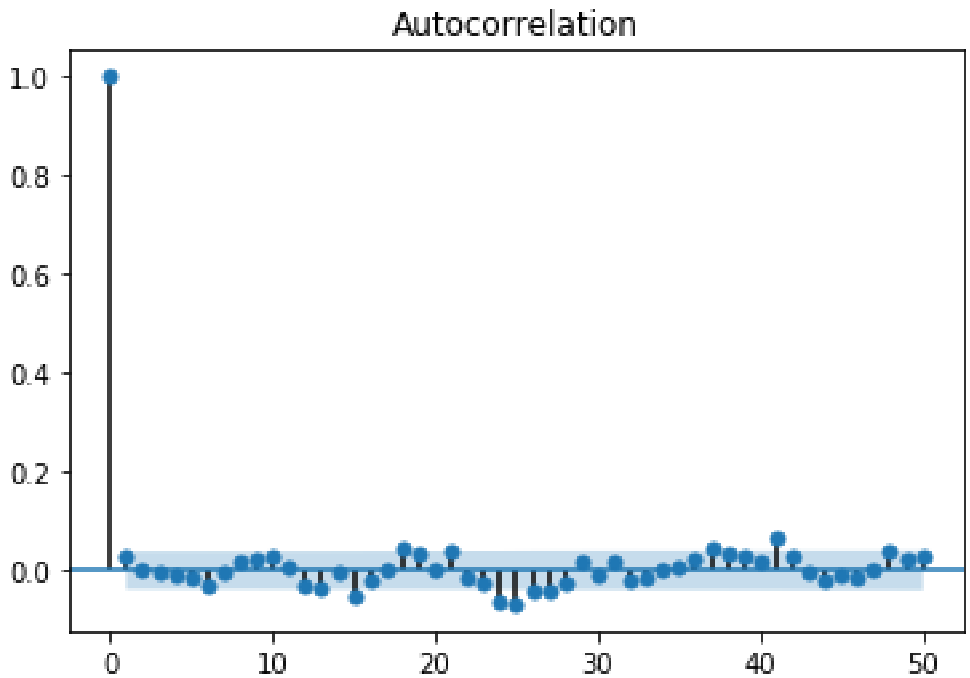

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0
10
20
30
40
50

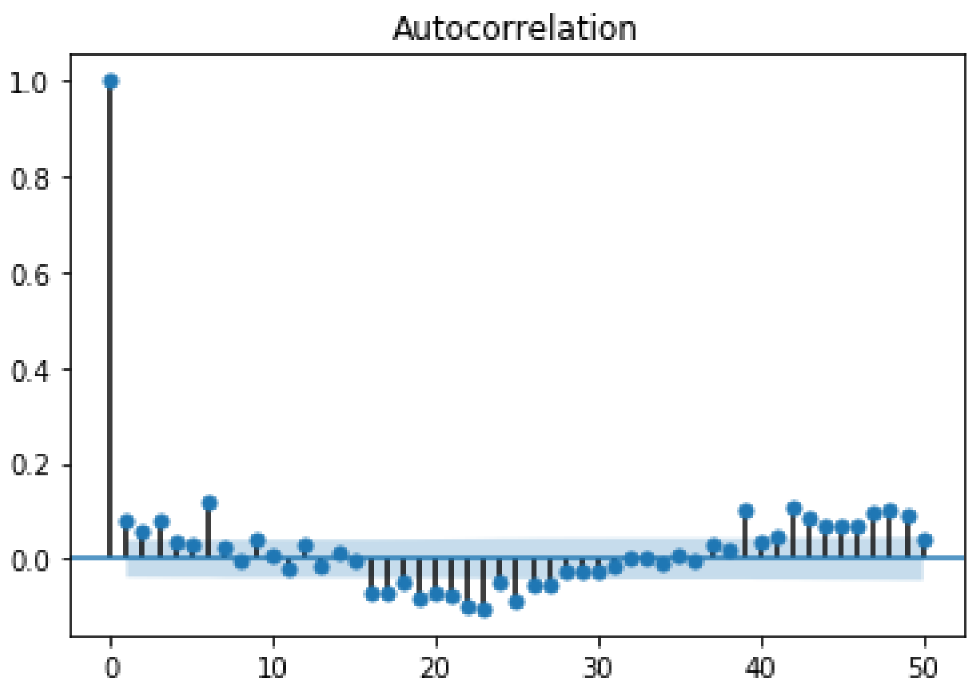


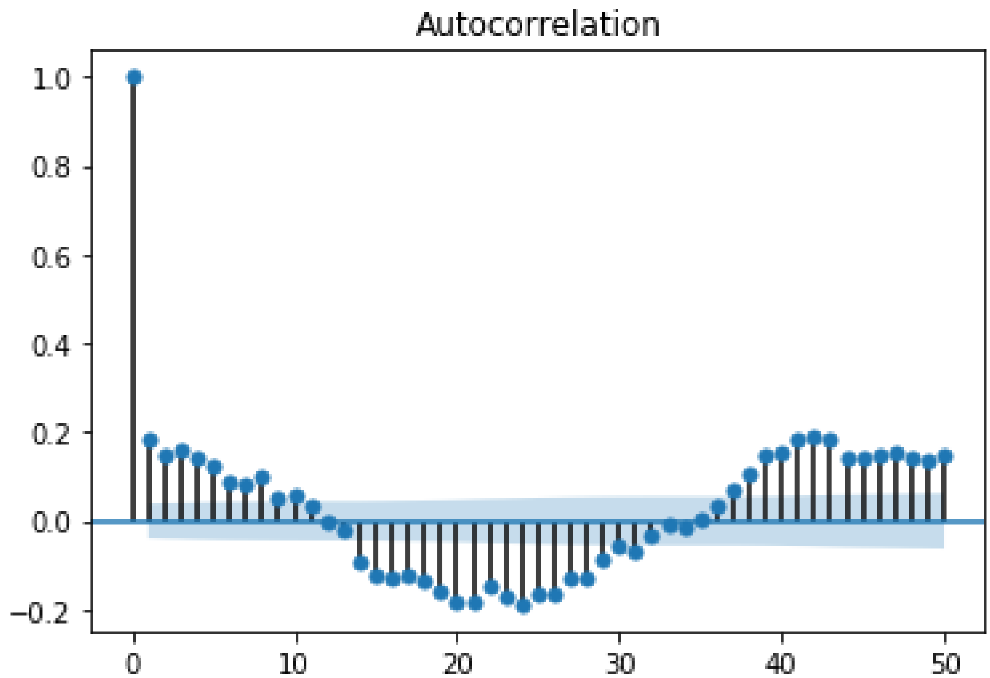
Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

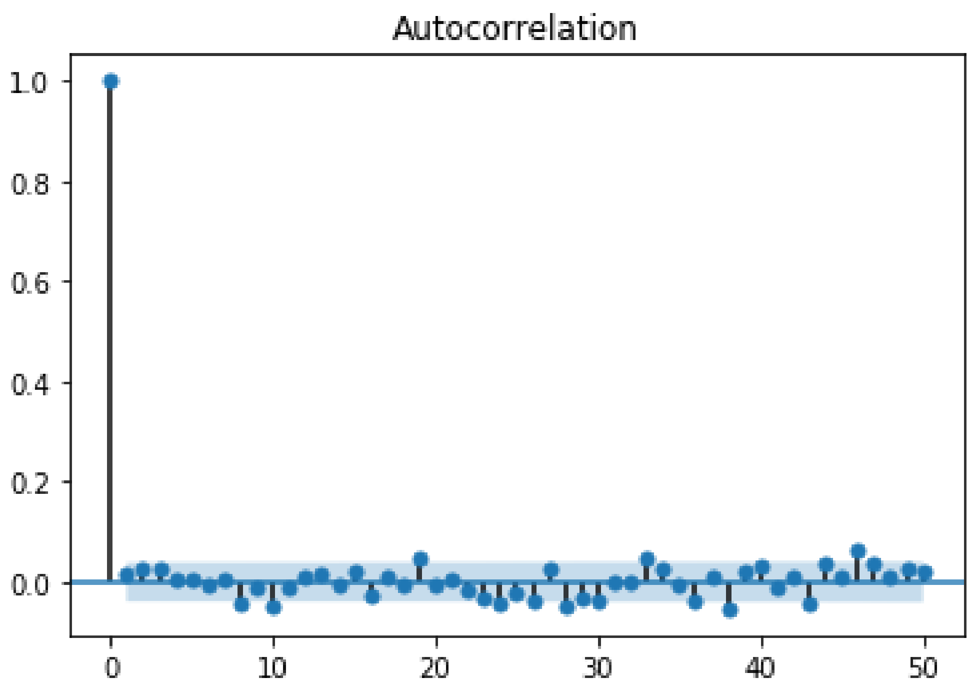


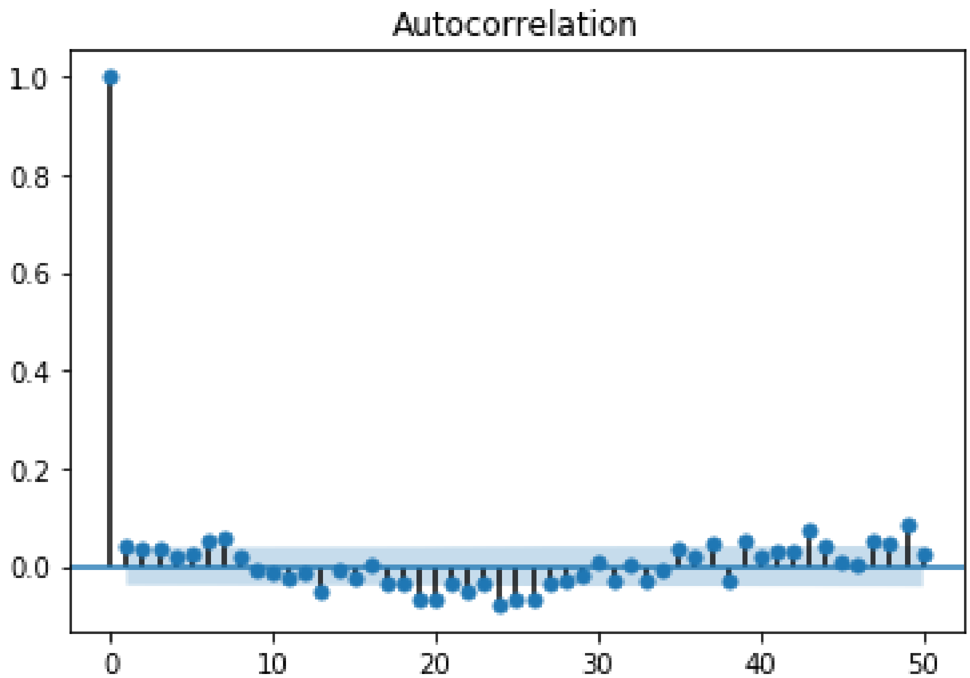
Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

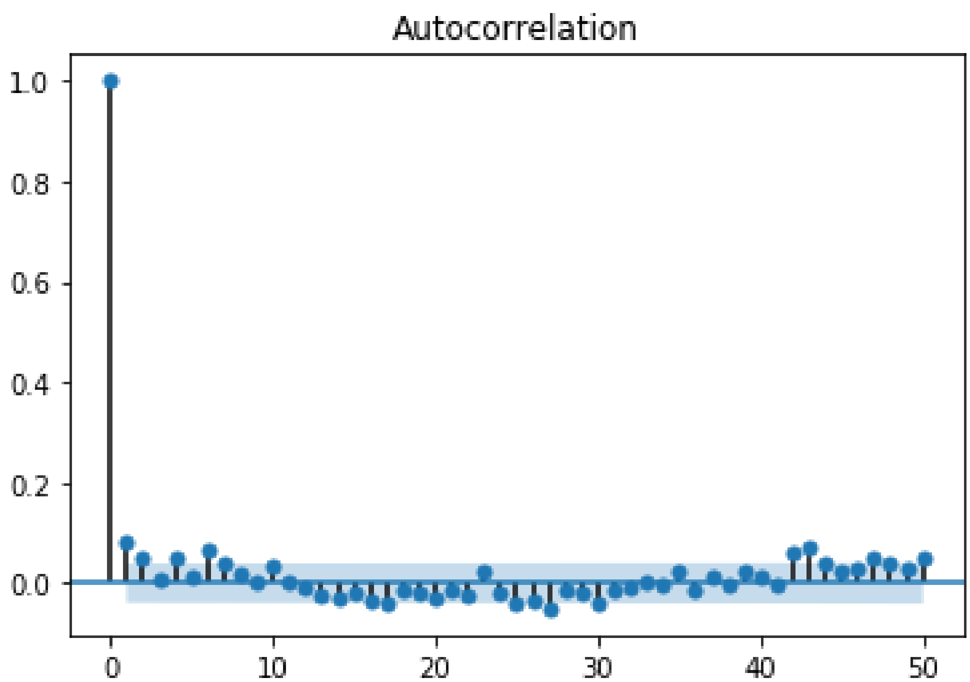

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

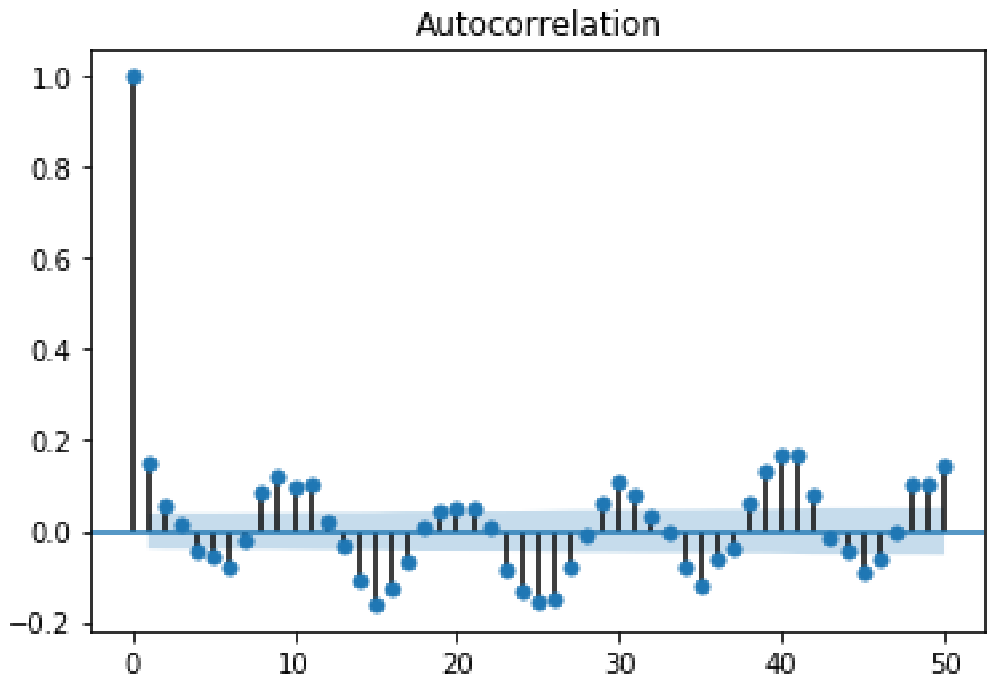

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

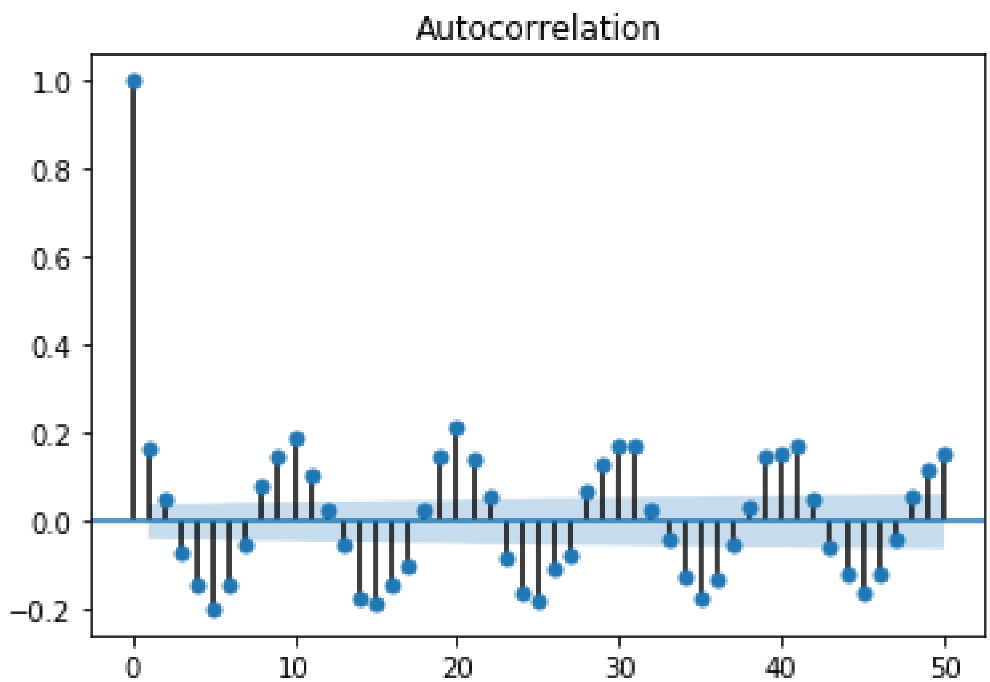

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

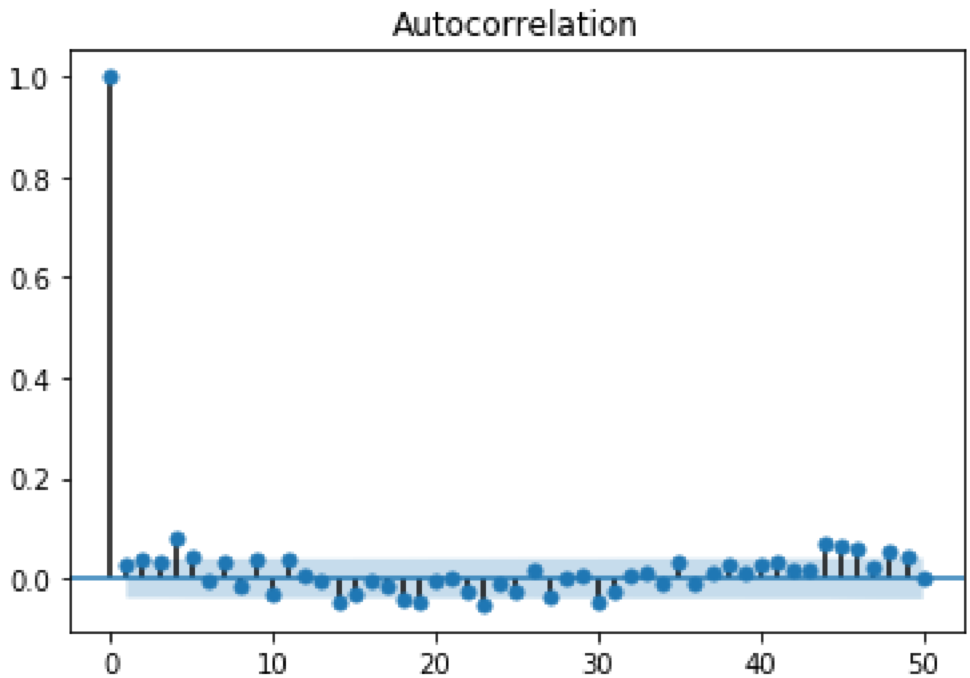

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

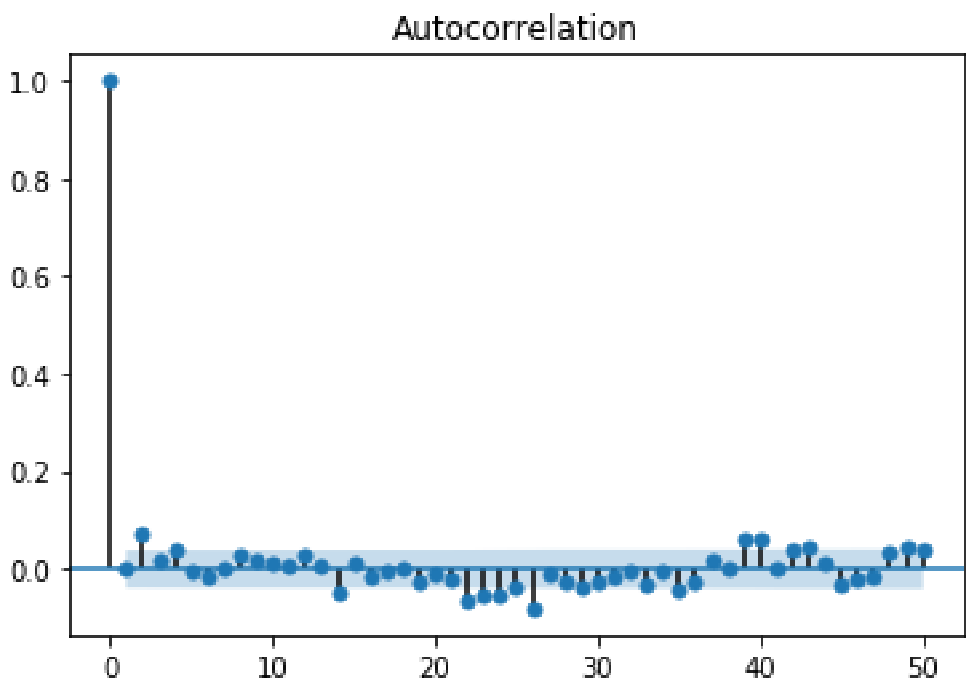

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

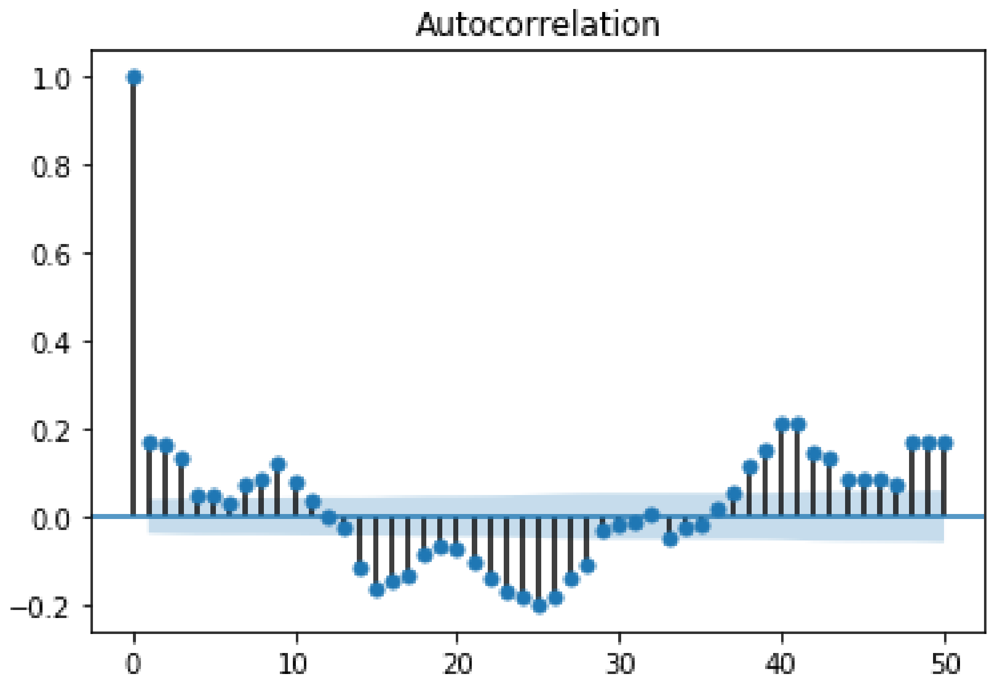

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

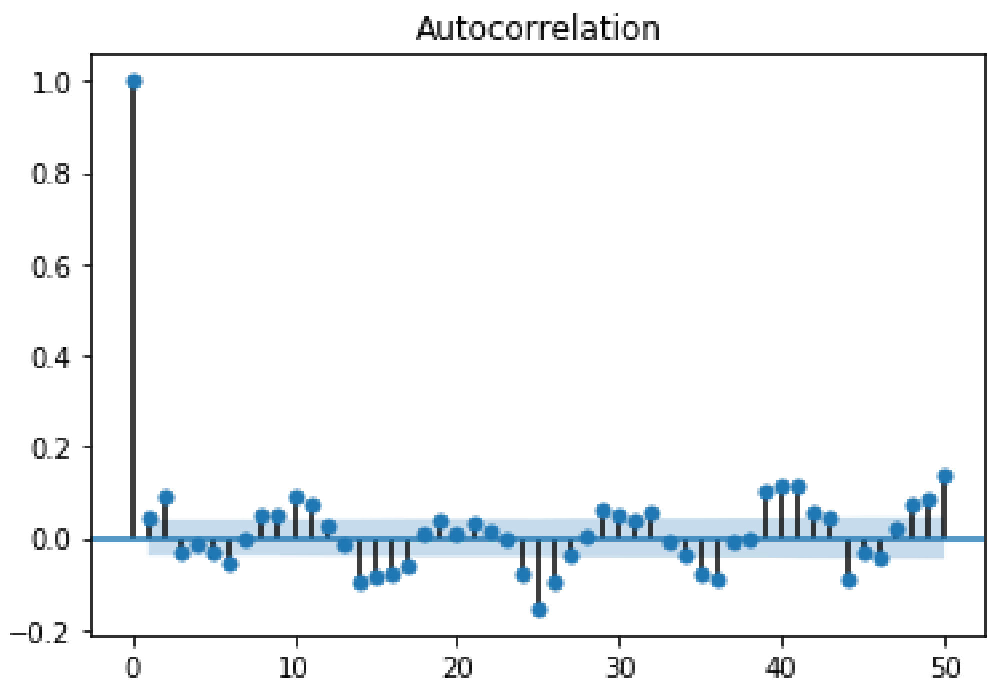

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

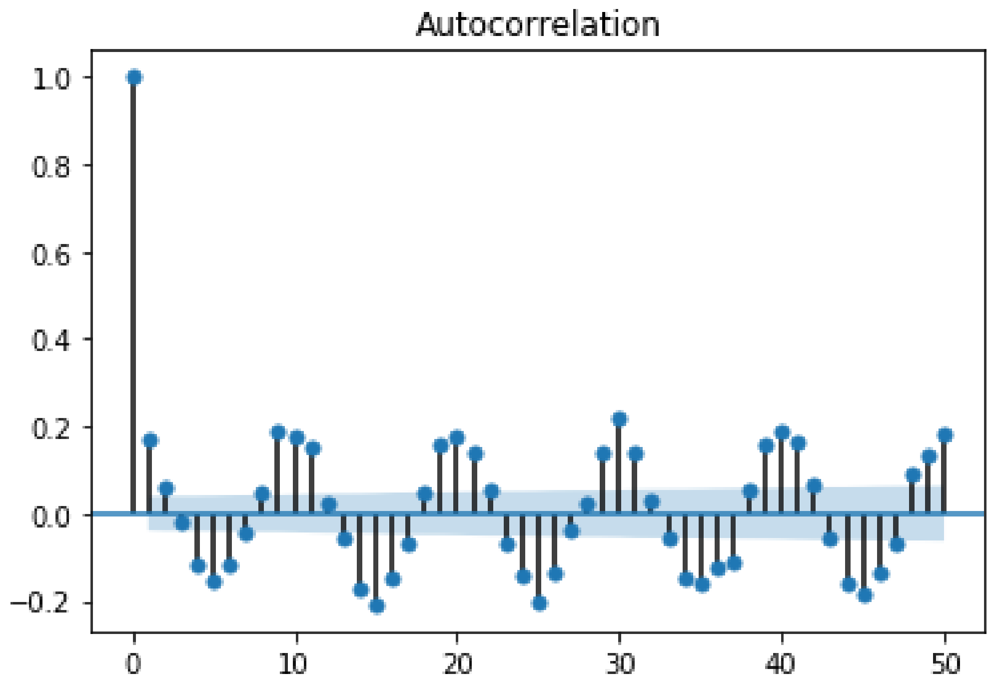

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

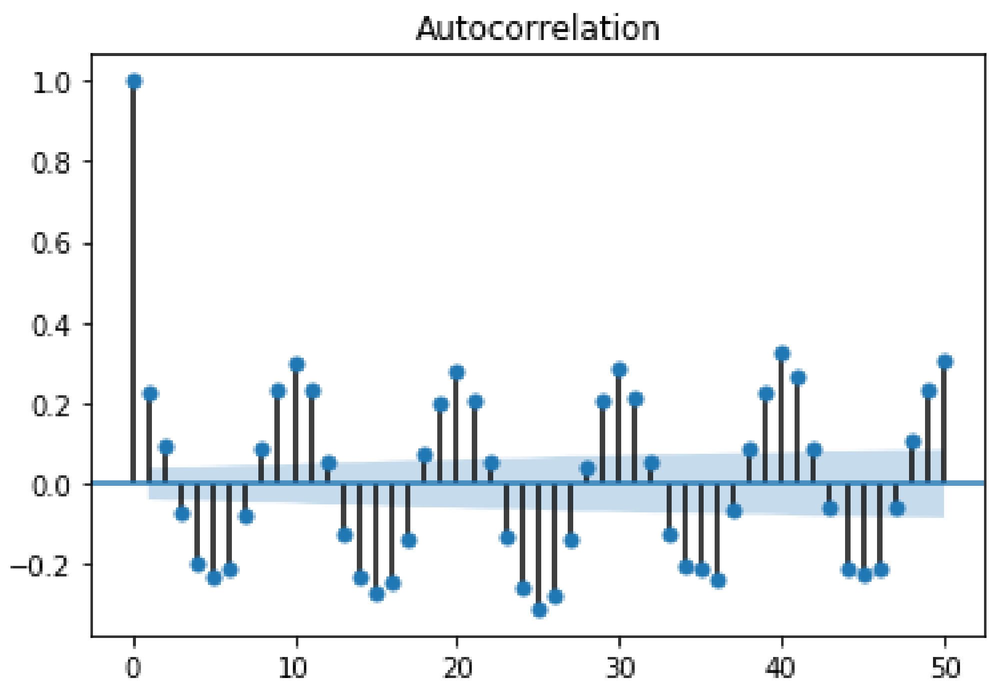

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

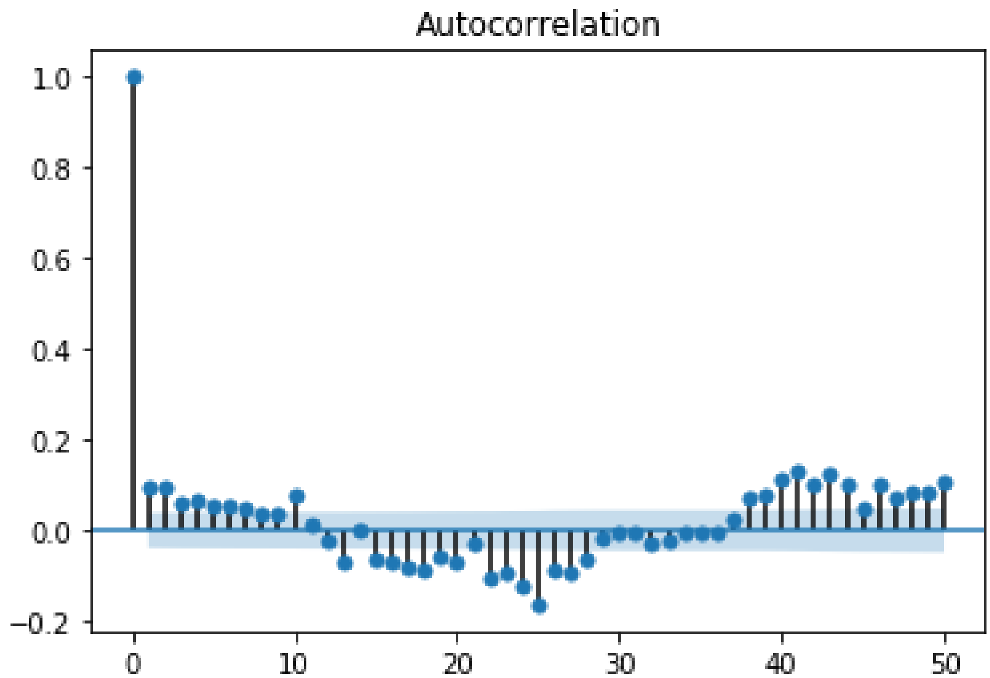

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

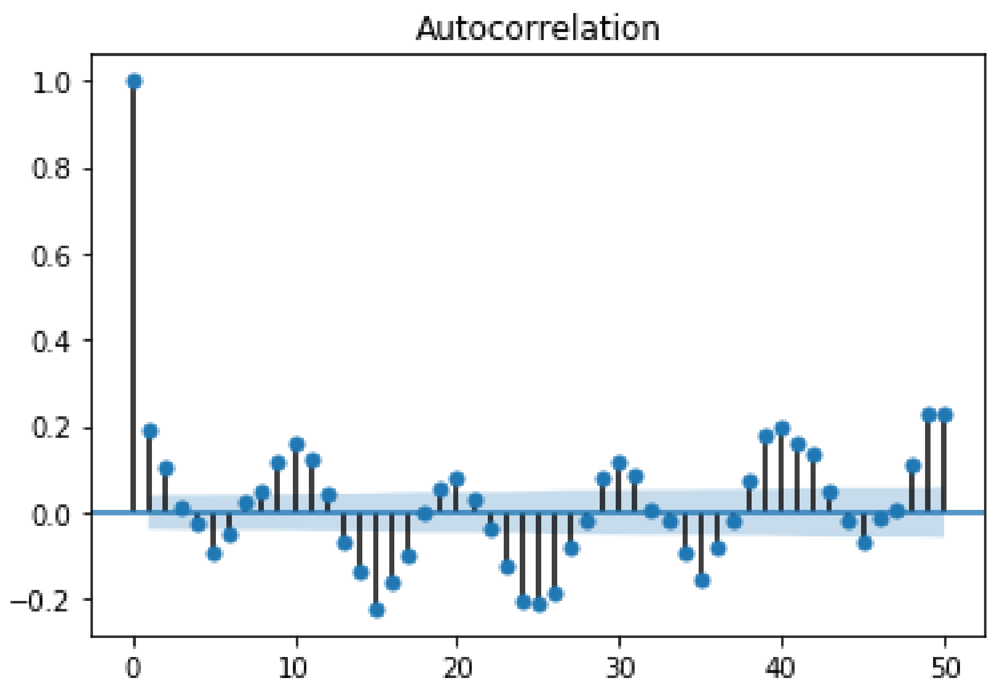

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

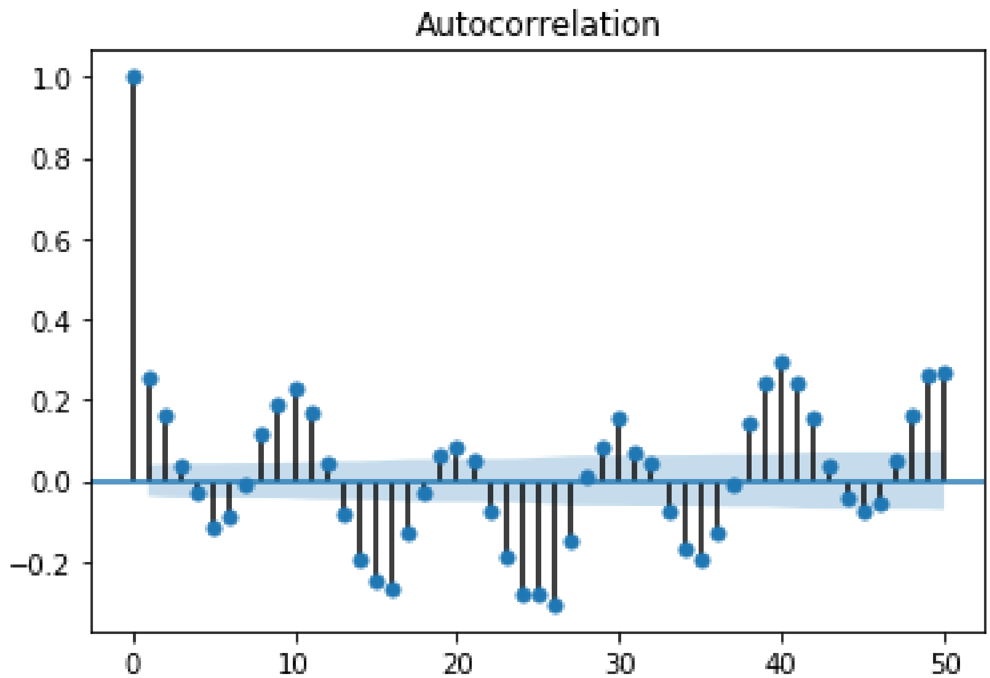

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

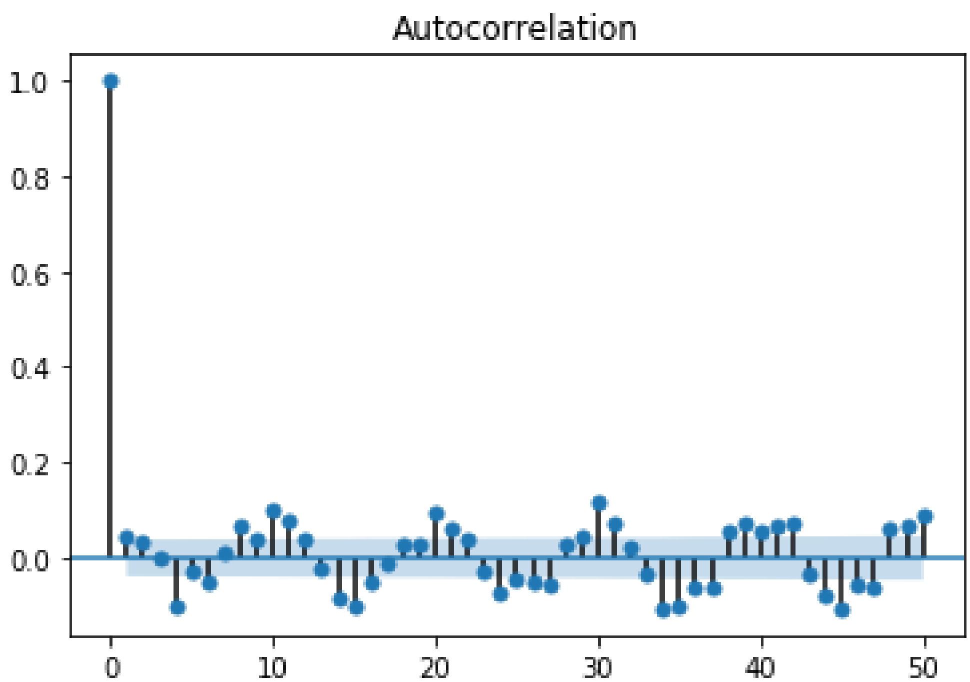

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

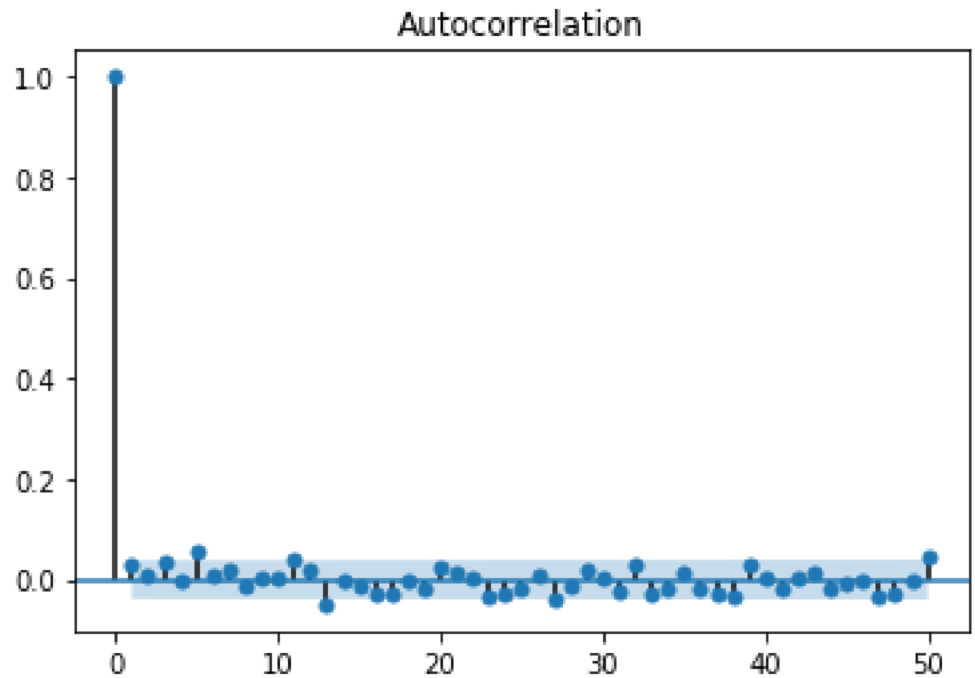

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

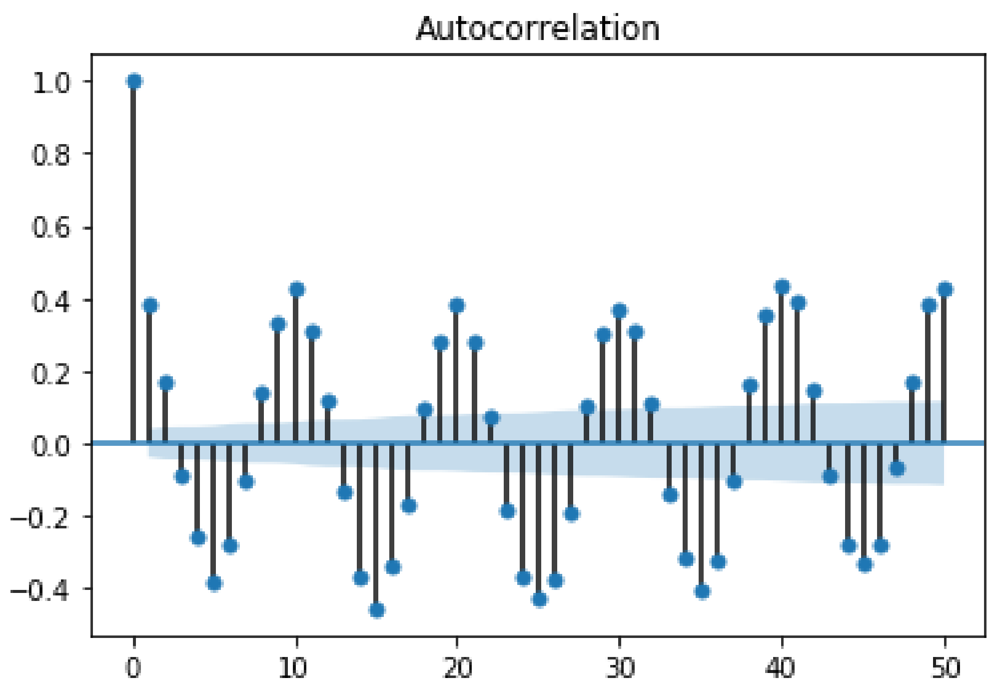

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

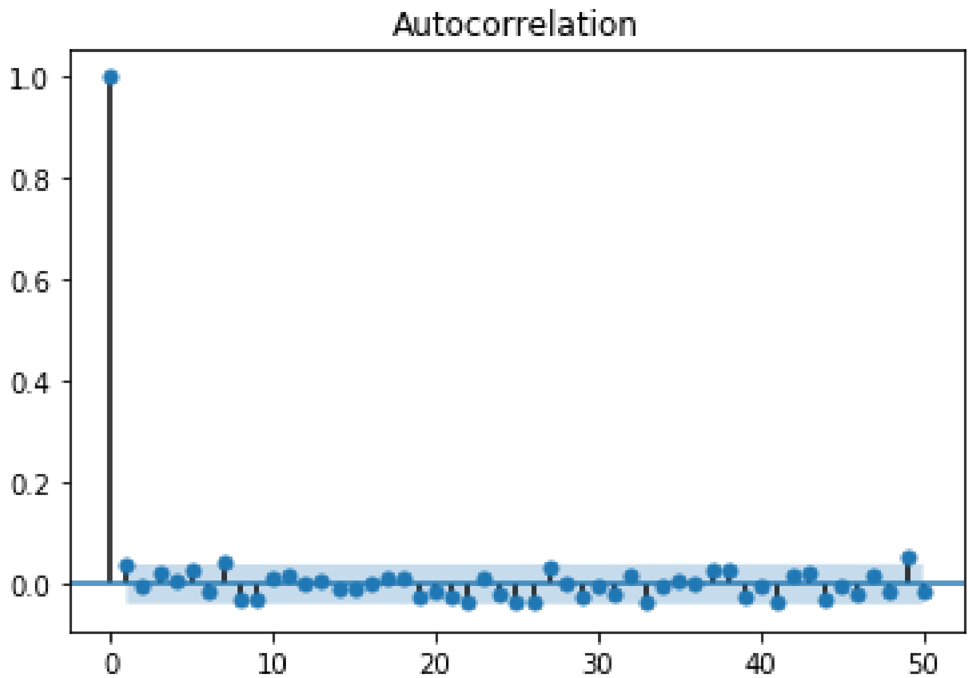

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

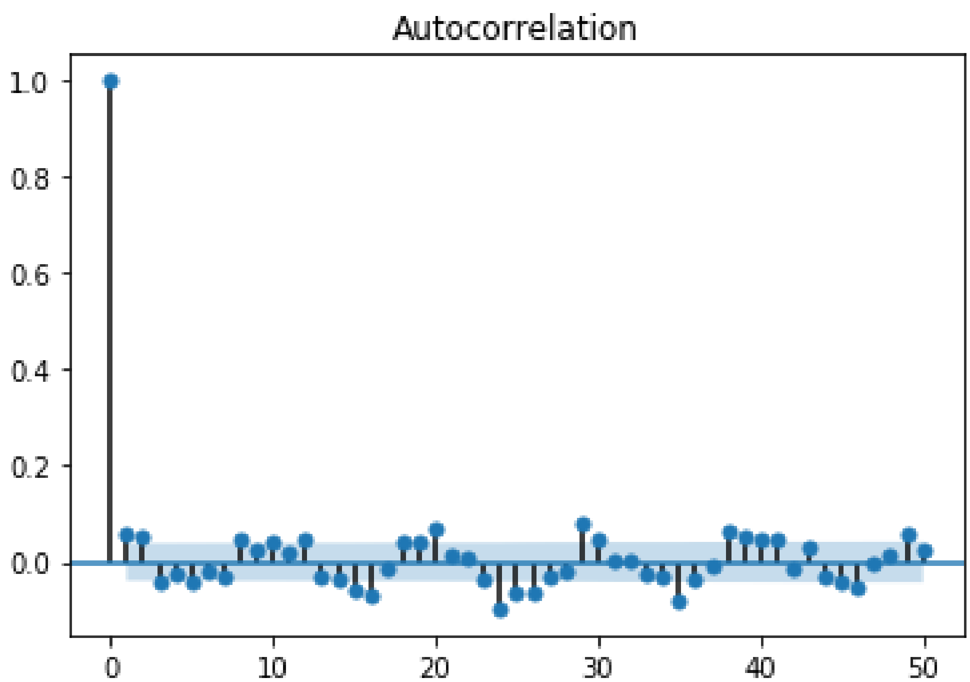

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

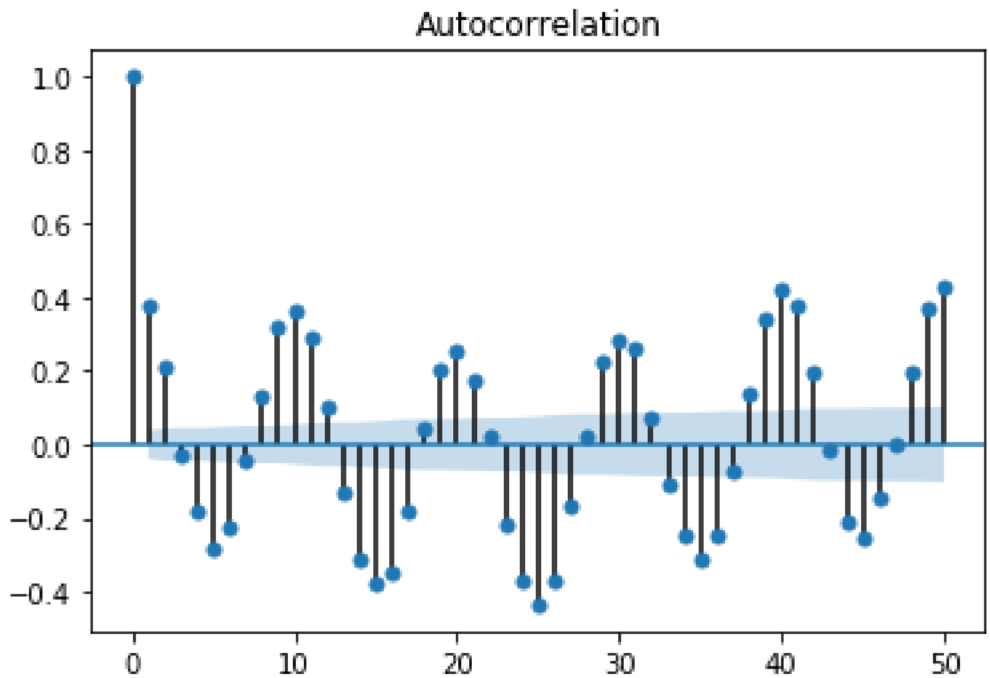

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

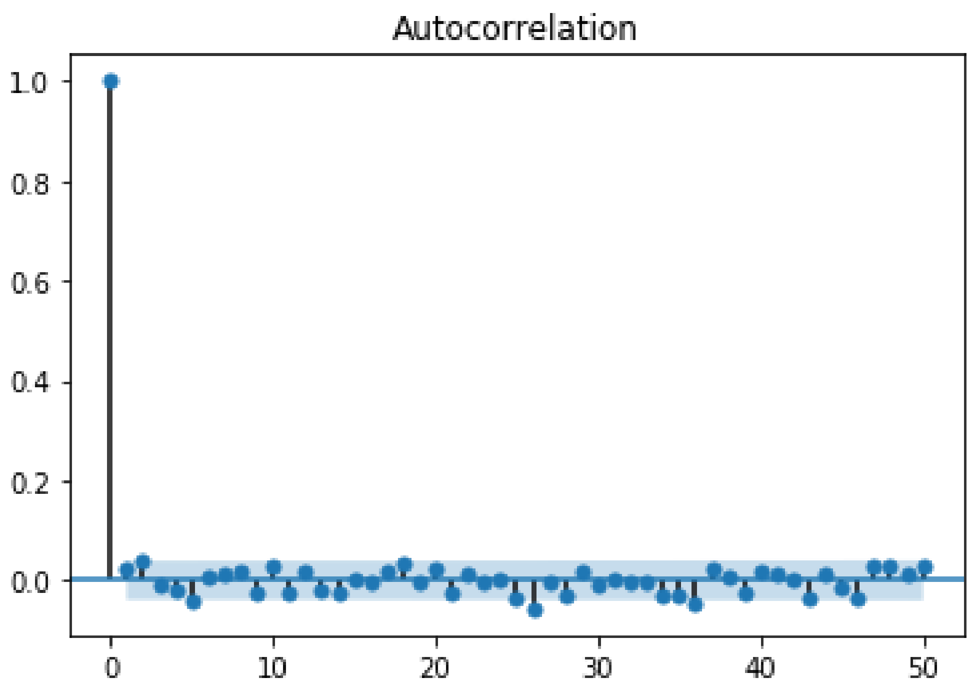

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

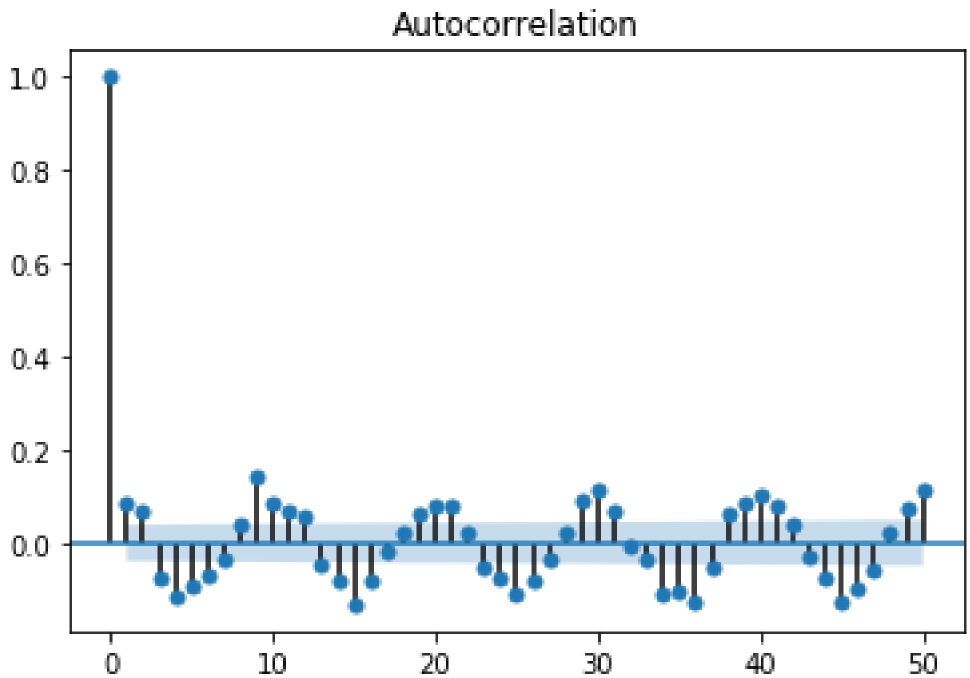

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0
10
20
30
40
50

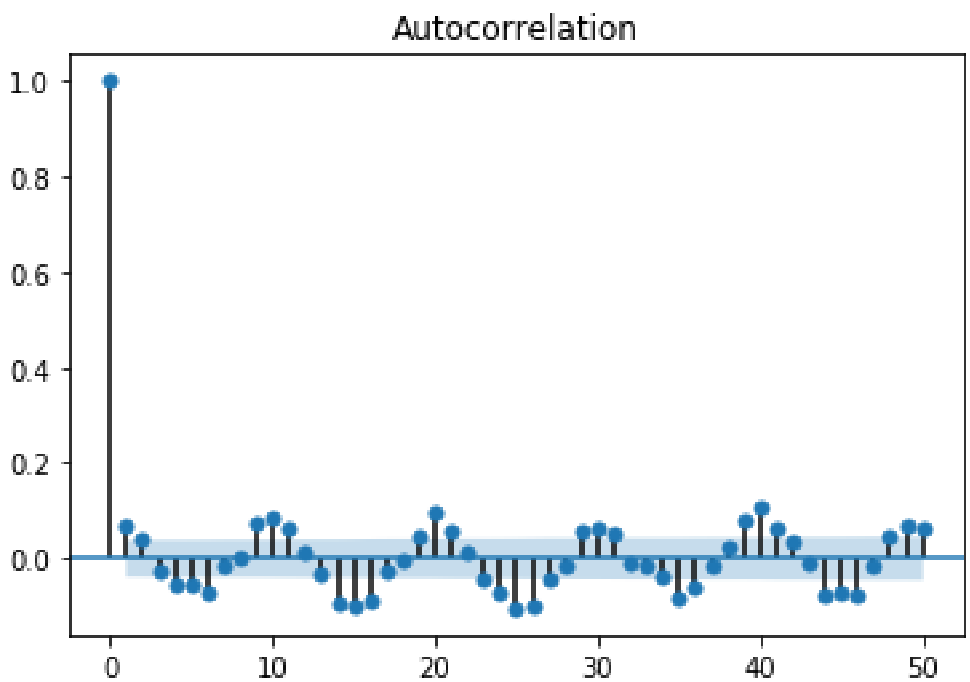

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

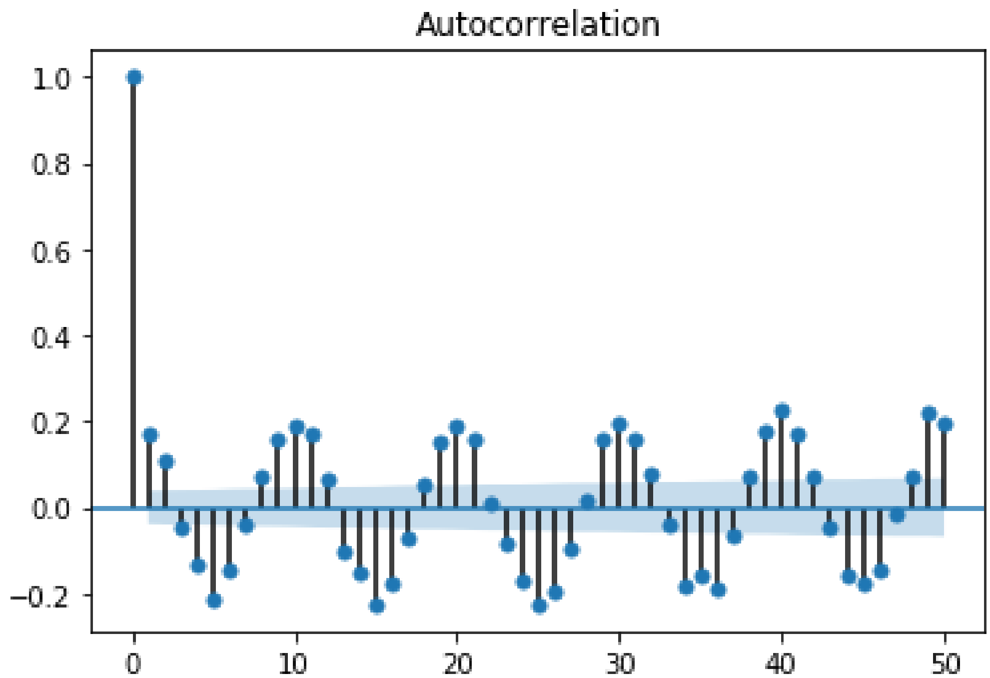

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

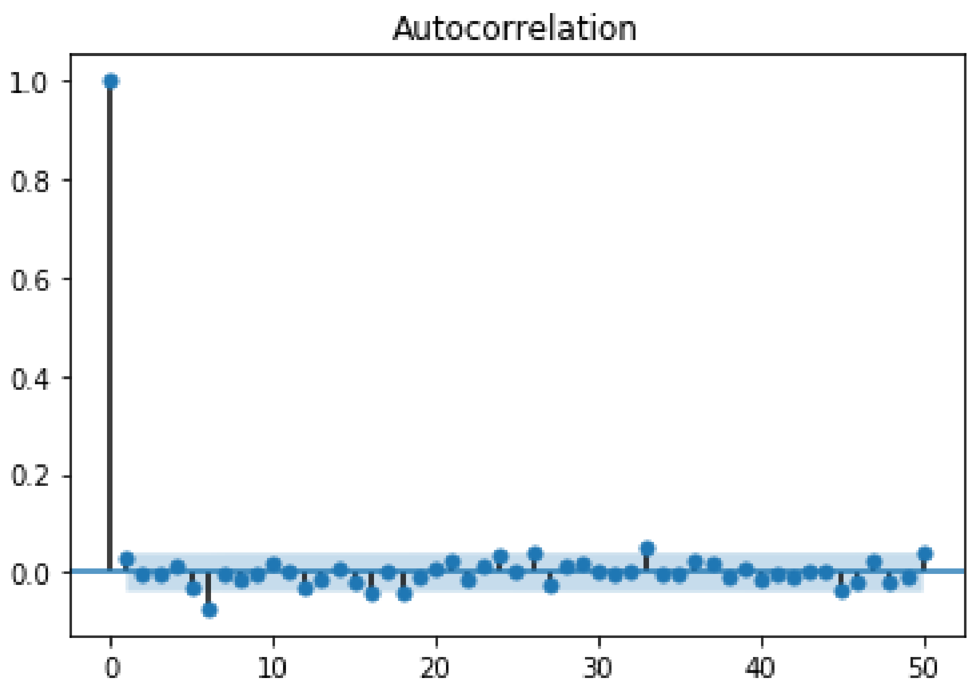

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0
10
20
30
40
50

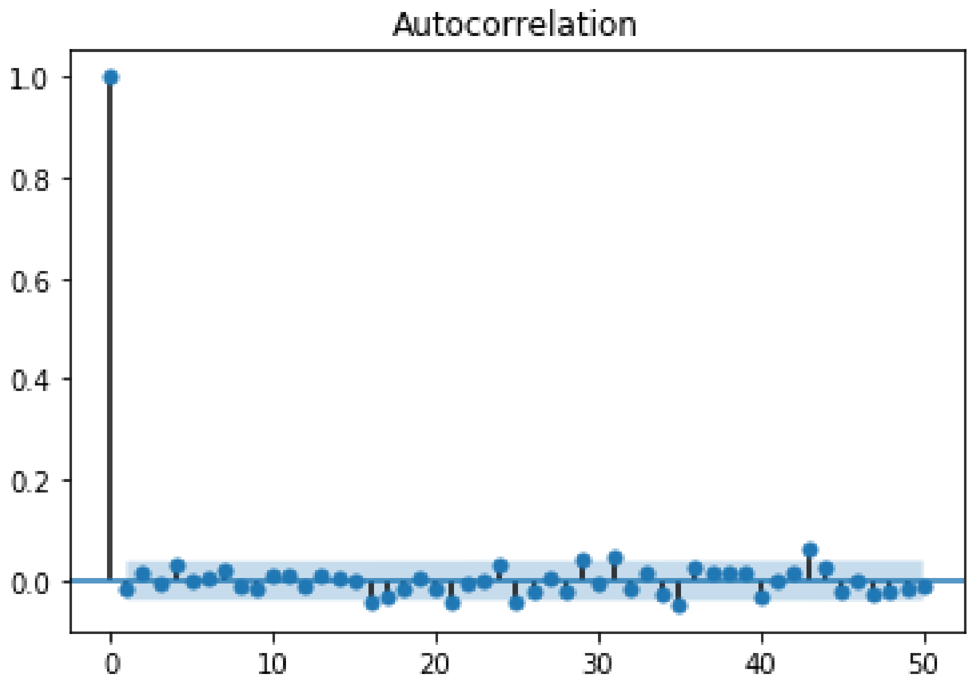

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

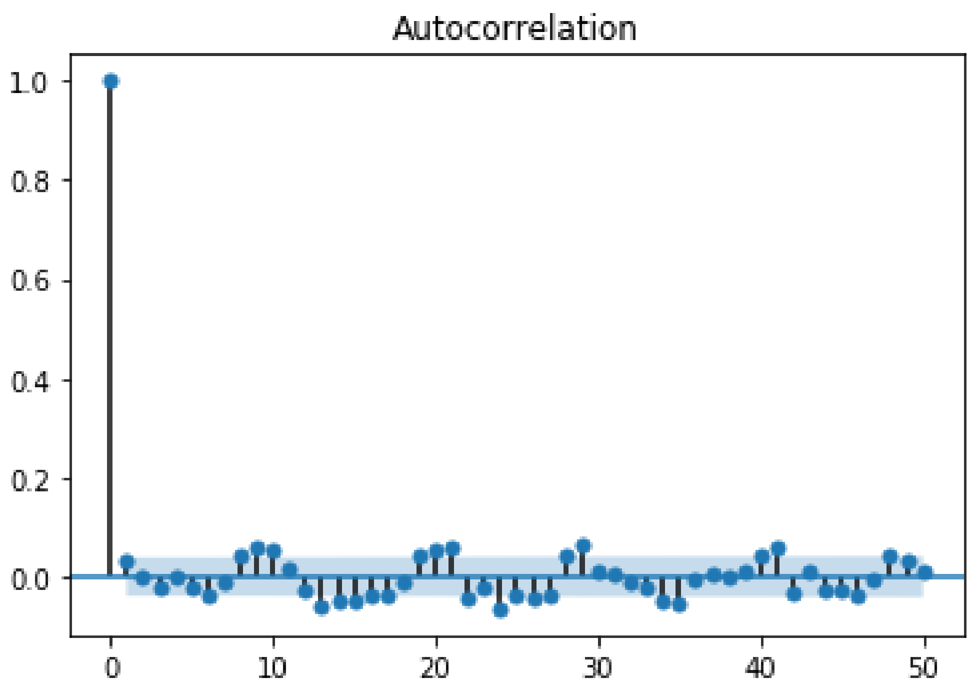

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

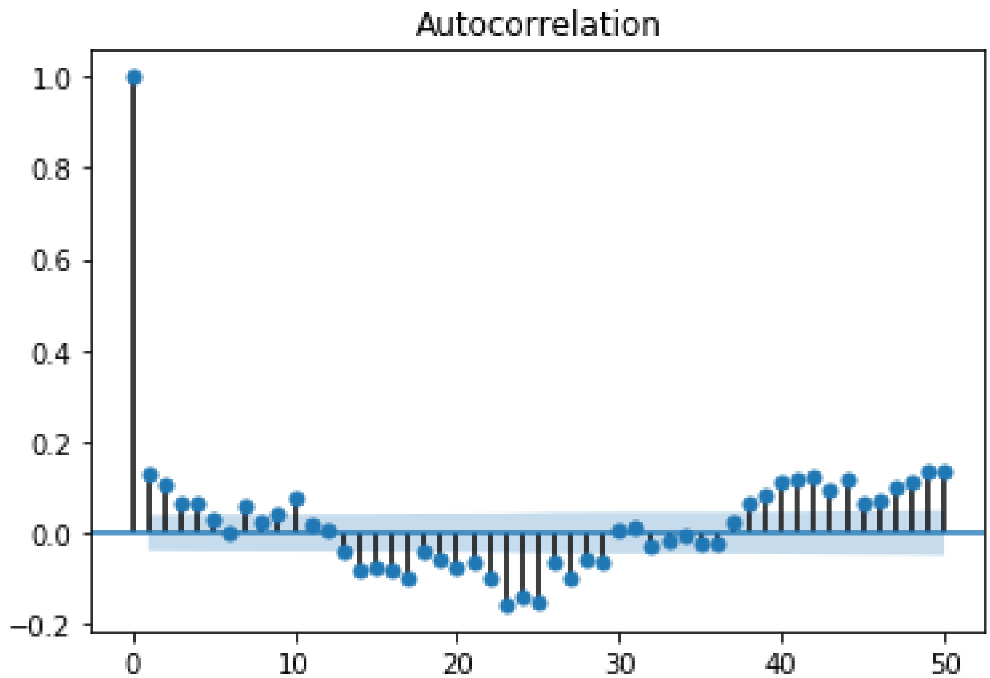

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

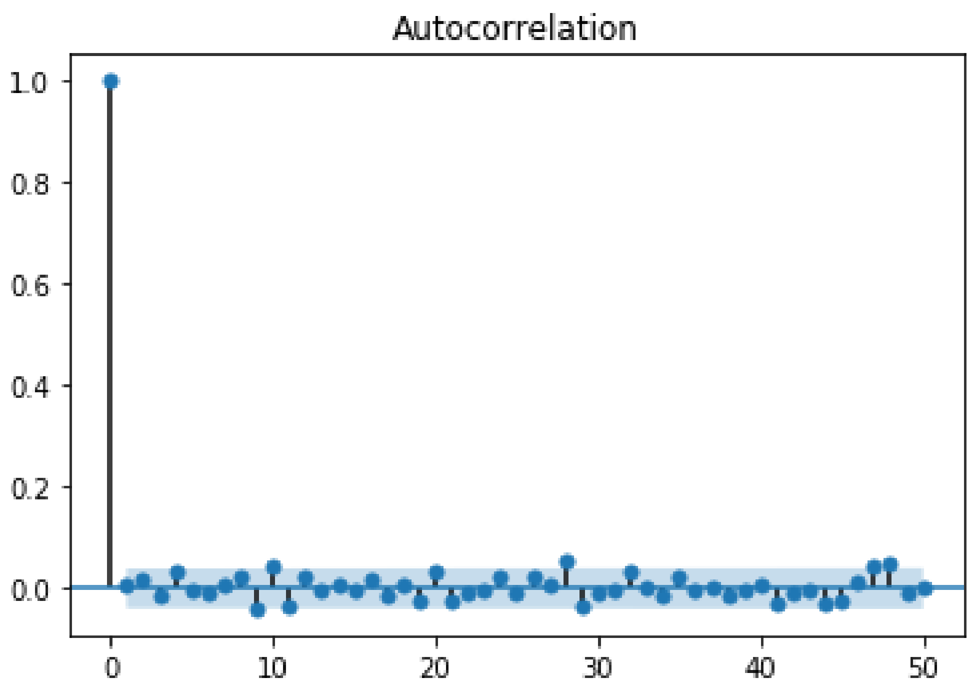

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

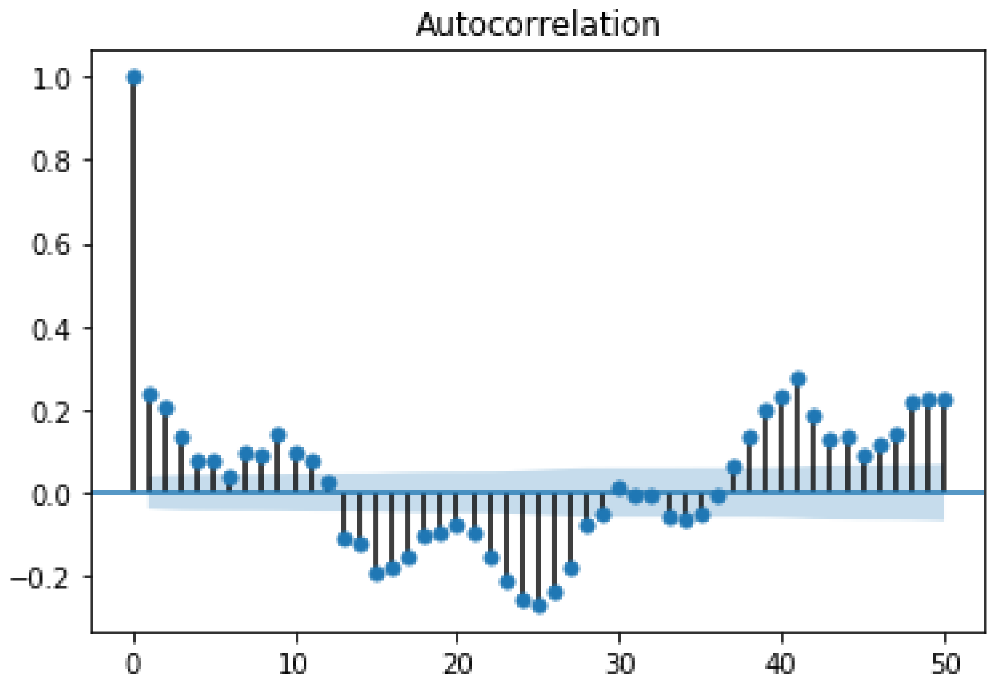

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50


Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

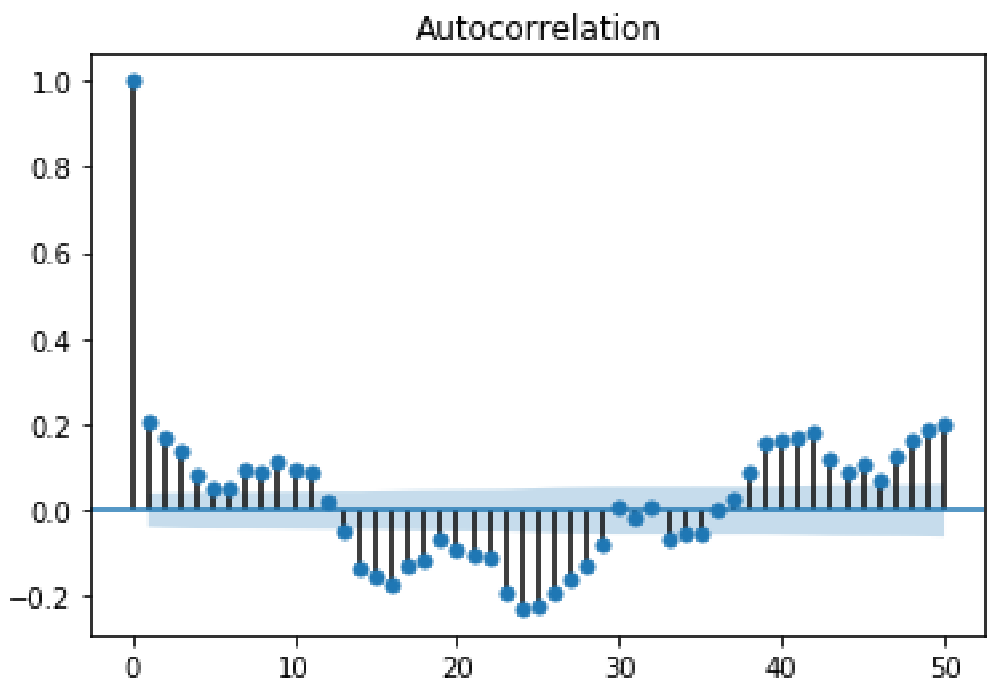

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50


Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

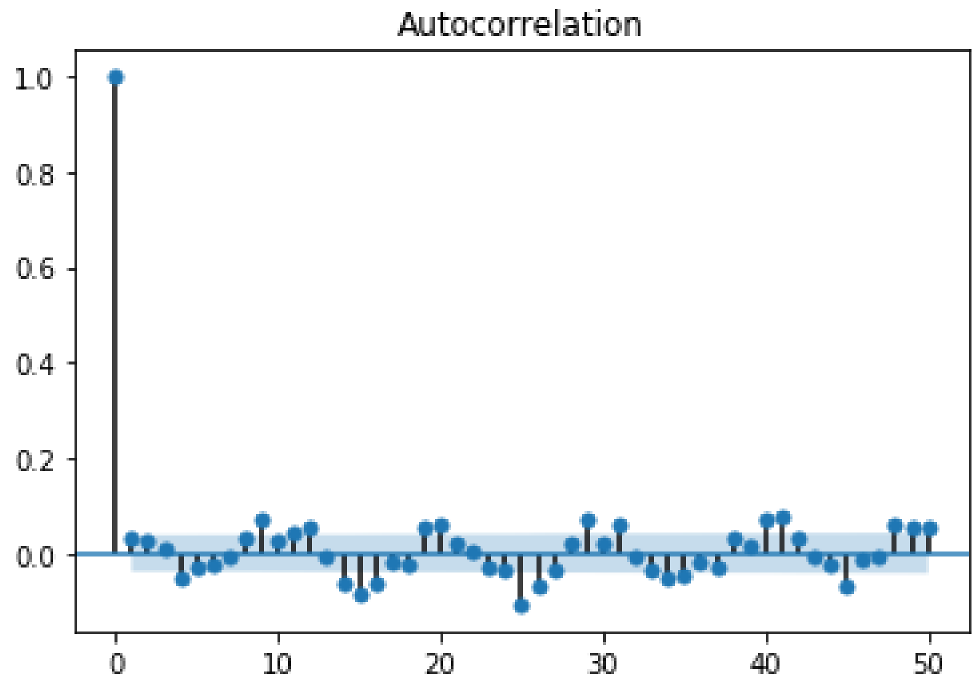

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50



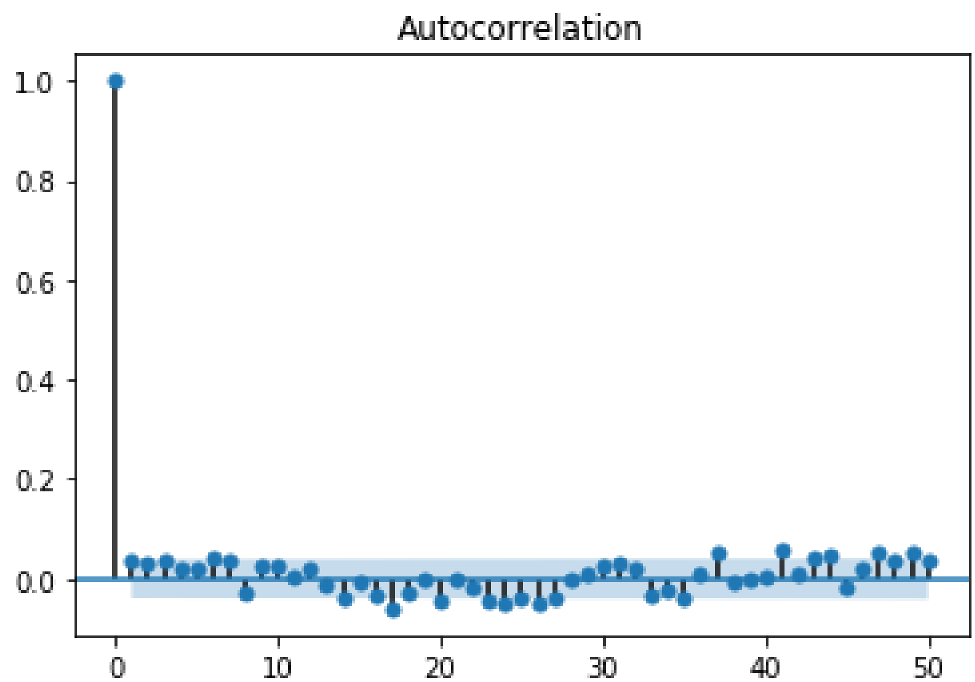
Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50



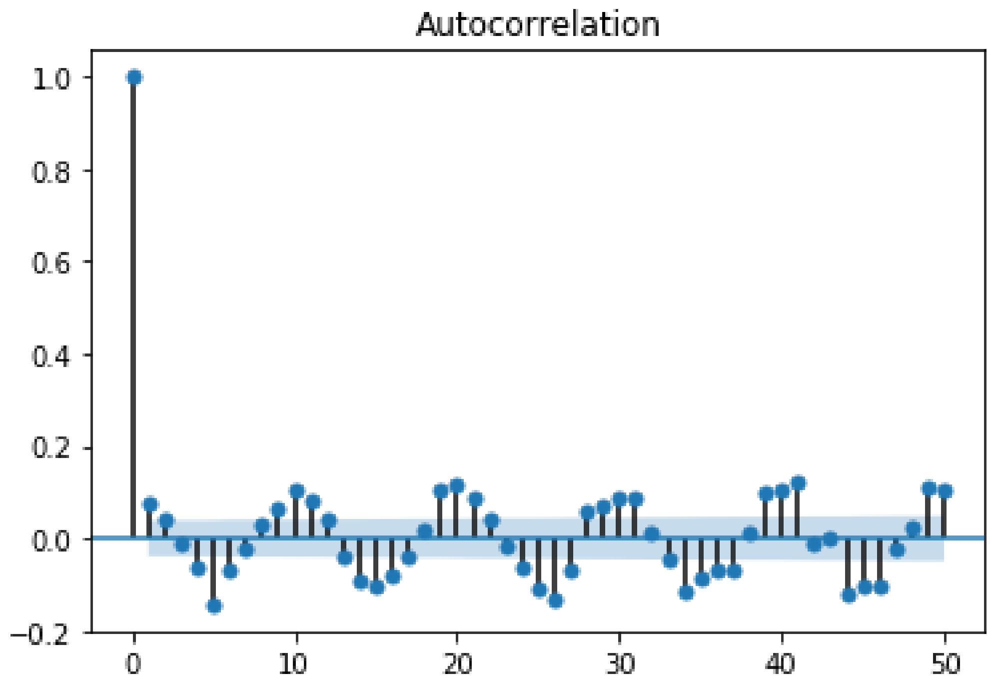
Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

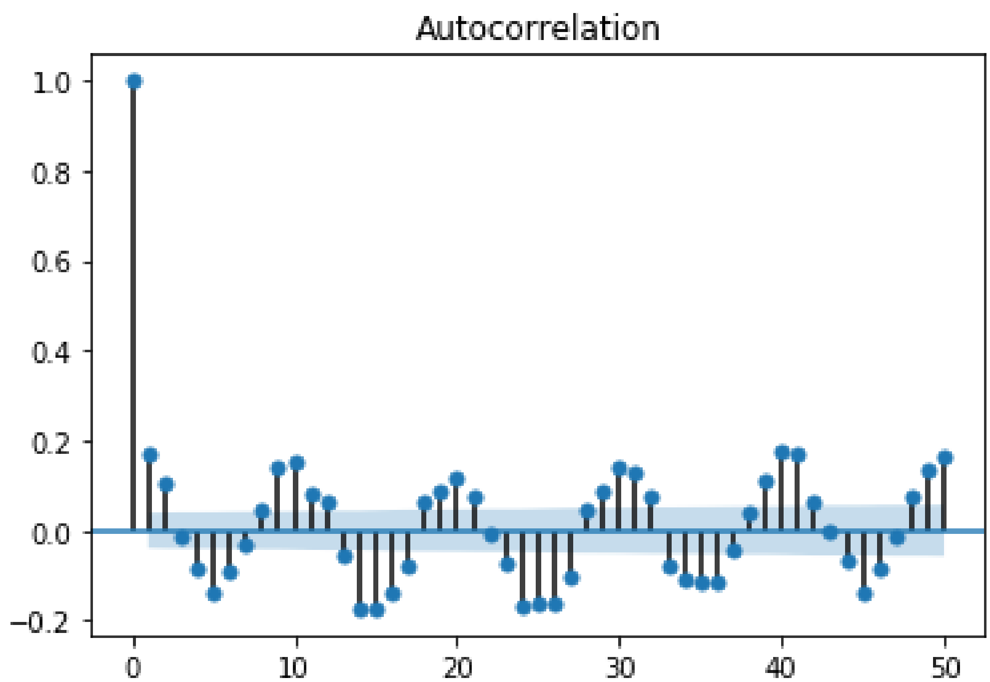


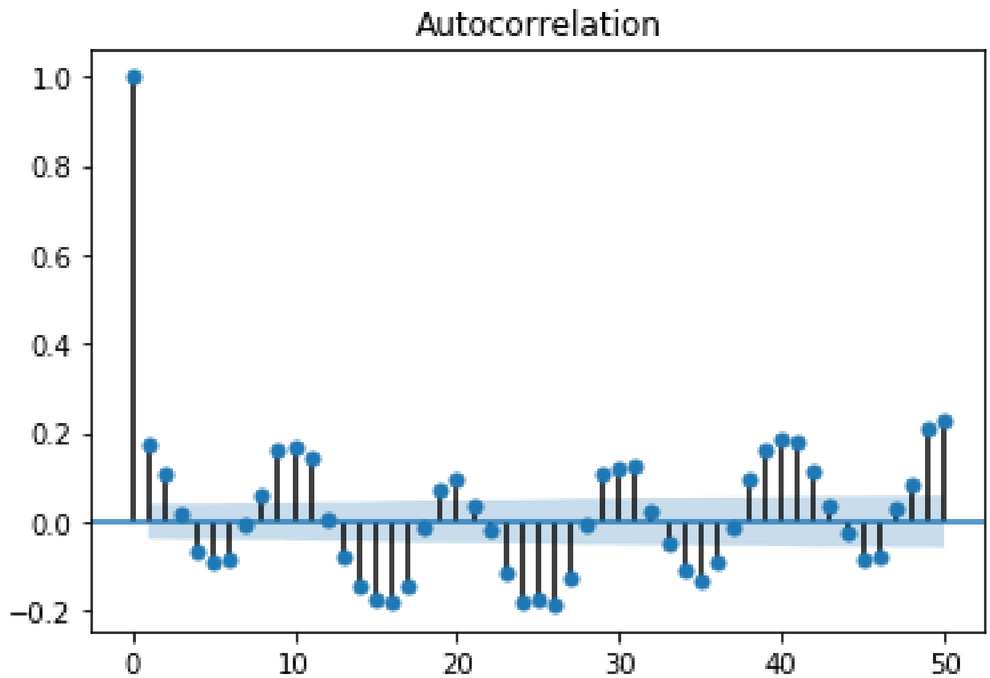
Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

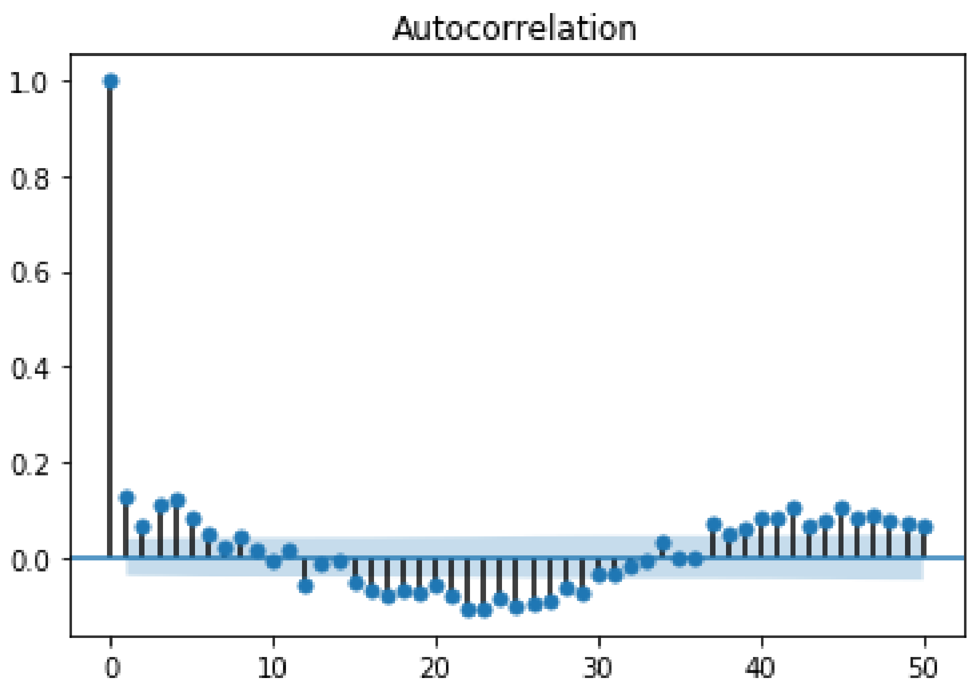


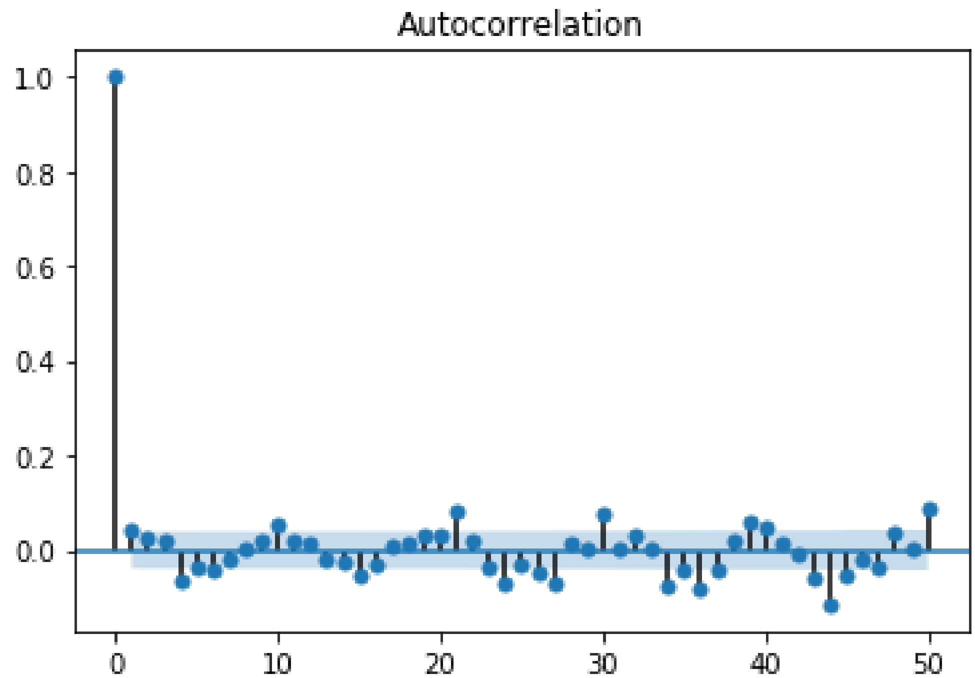
Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

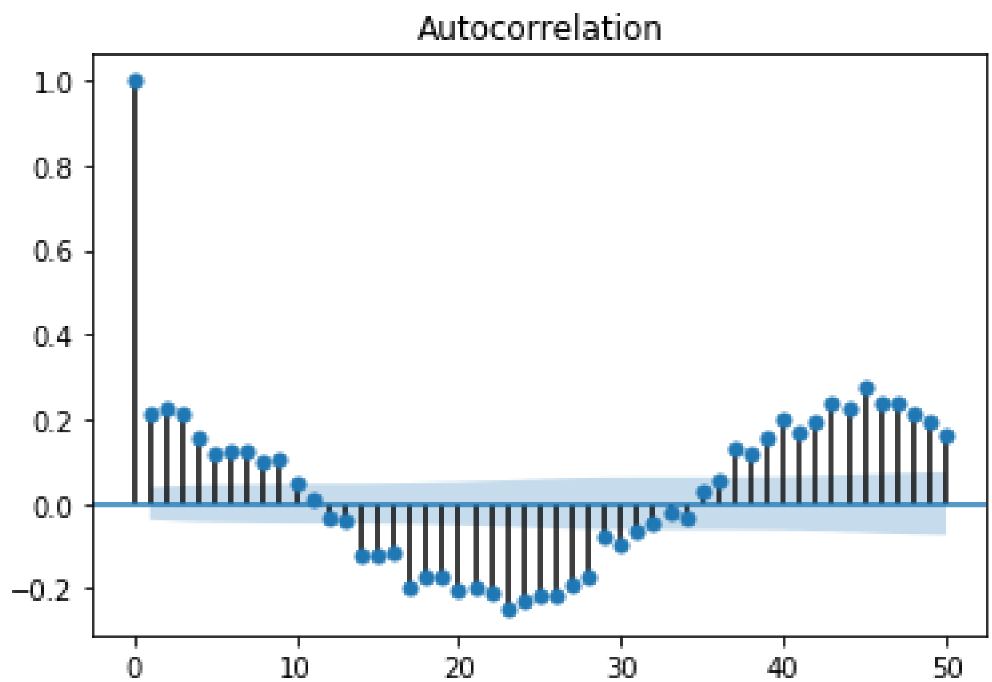

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

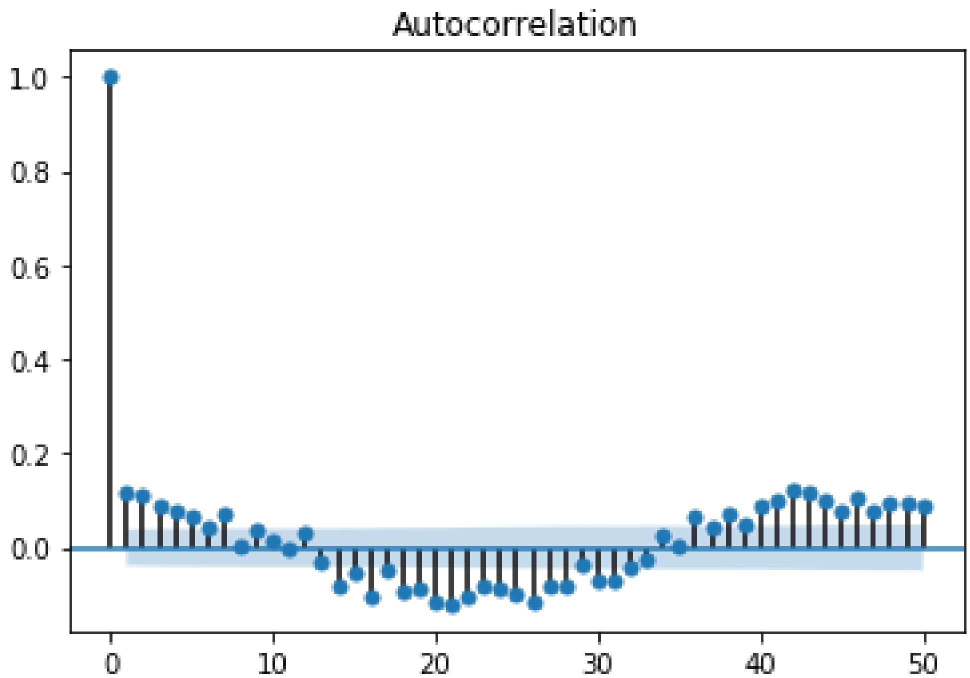

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50


Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50


Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

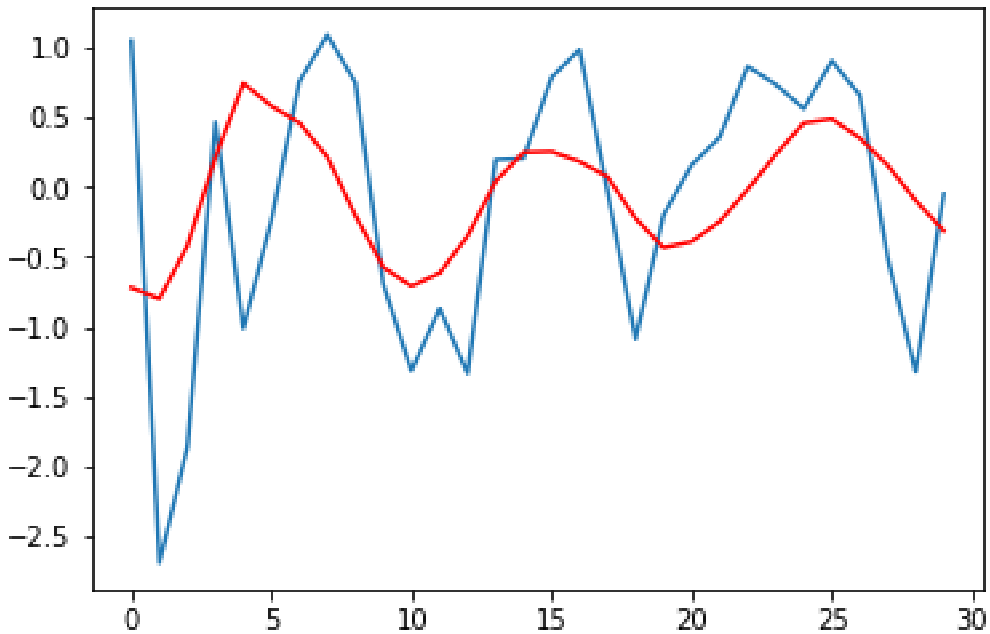

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50


Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

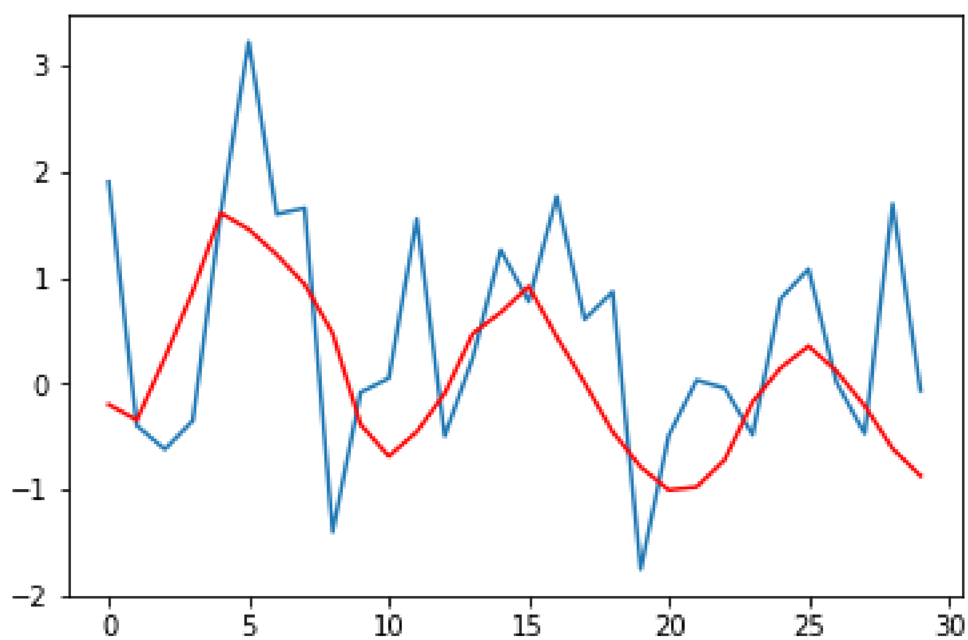


In [6]:
```
# Fit an Autoregressive model
# It will return the number of lags that model chooses to use and the rmse on one 30-d
for i in list(data2):
    series = data2[[str(i)]]
    # split dataset
    X = series.values
    train, test = X[:len(X)-30], X[len(X)-30:]
    # train autoregression
    model = AR(train)
    model_fit = model.fit()
    print('Lag: %s' % model_fit.k_ar)
    # make predictions
    predictions = model_fit.predict(start=len(train), end=len(train)+len(test)-1, dynam
    m = mean_squared_error(test, predictions)
    print('RMSE: %.3f' % np.sqrt(m))
    # plot results
    pyplot.plot(test)
    pyplot.plot(predictions, color='red')
    pyplot.show()
```

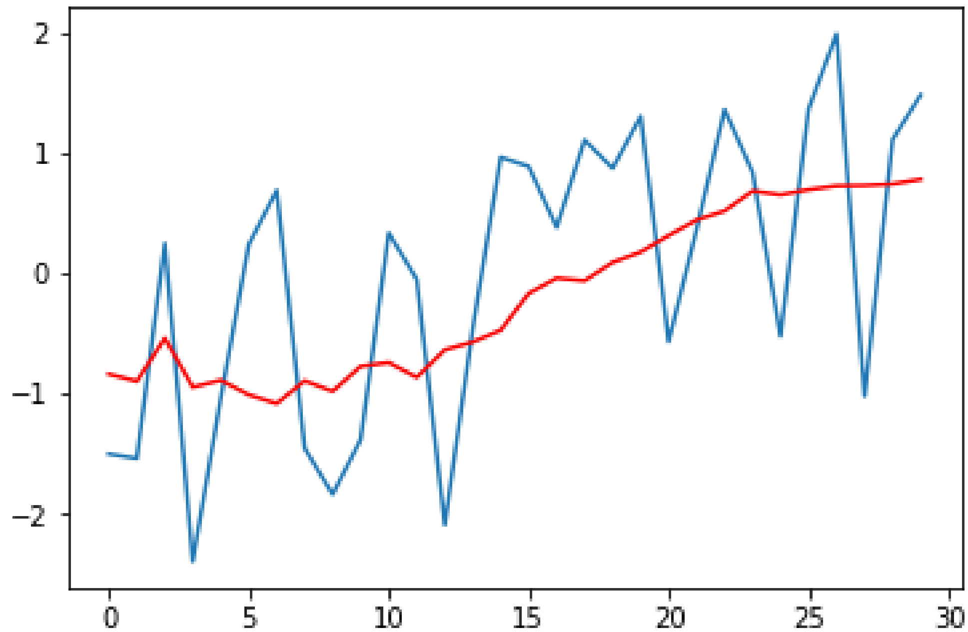
```
Lag: 27
RMSE: 0.847
```



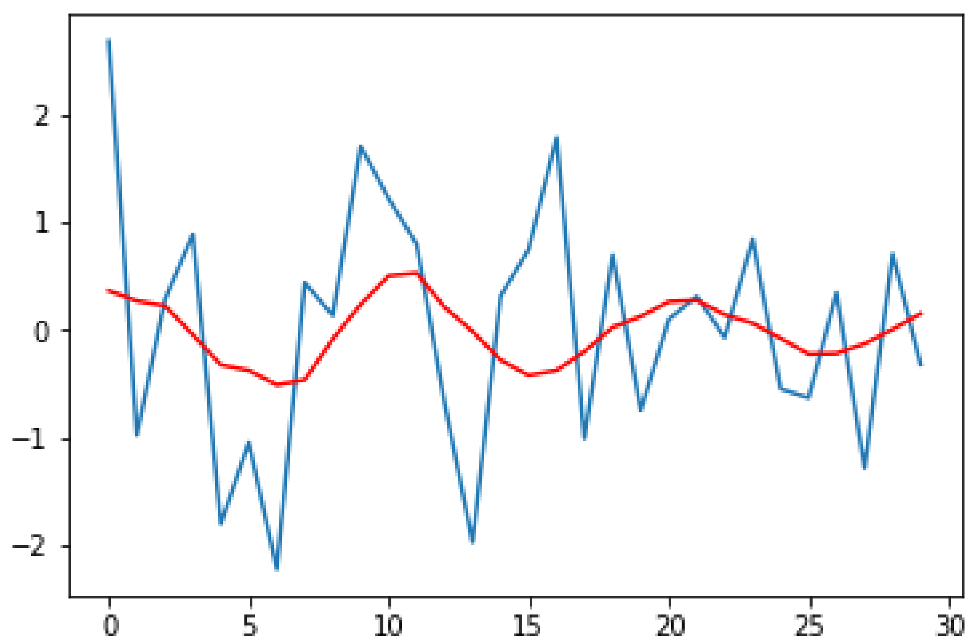

```
Lag: 27
RMSE: 1.047
```



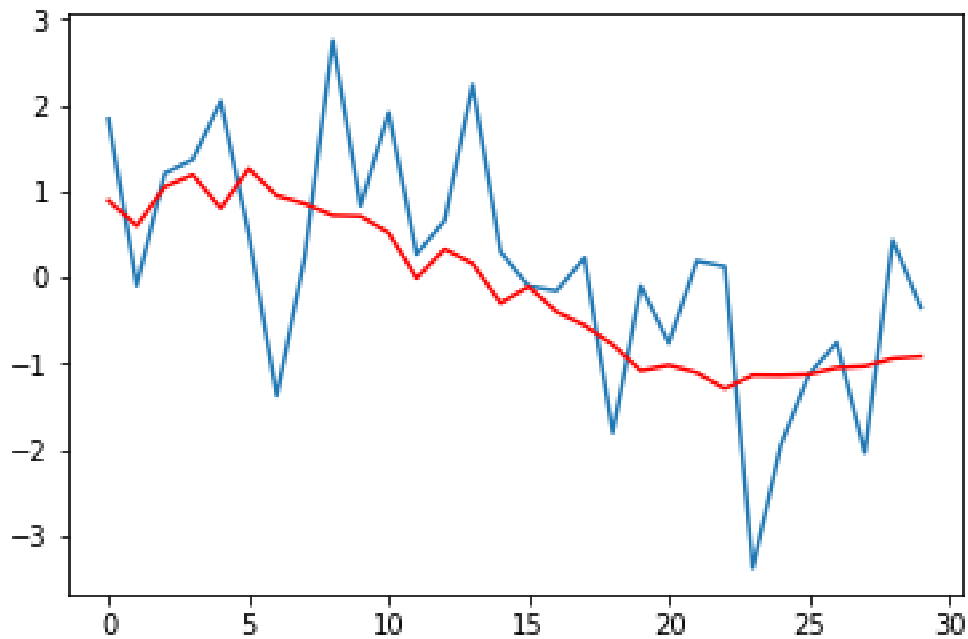
```
Lag: 27
RMSE: 0.981
```



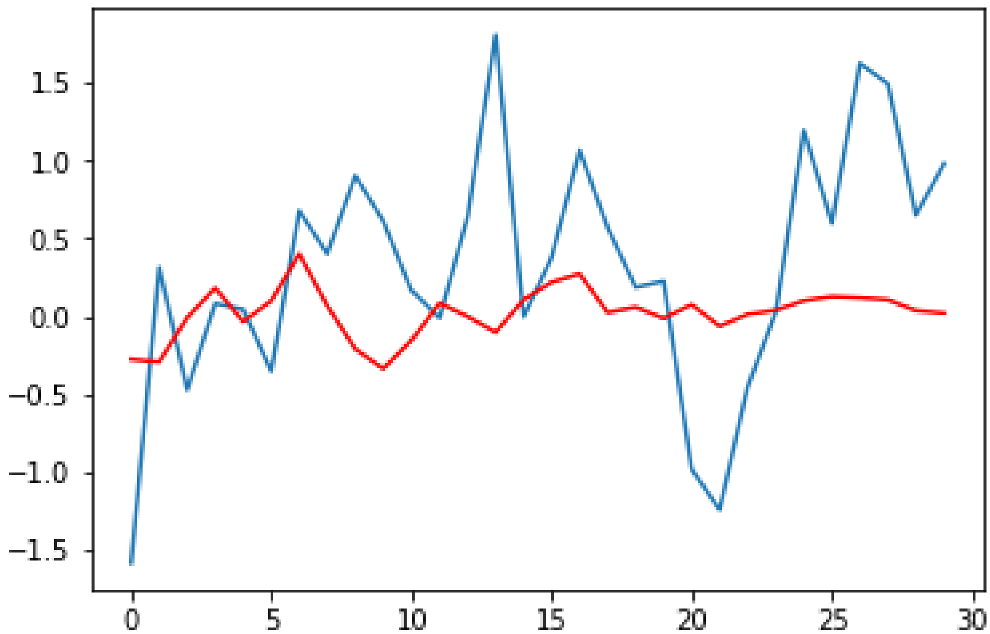

```
Lag: 27
RMSE: 1.054
```



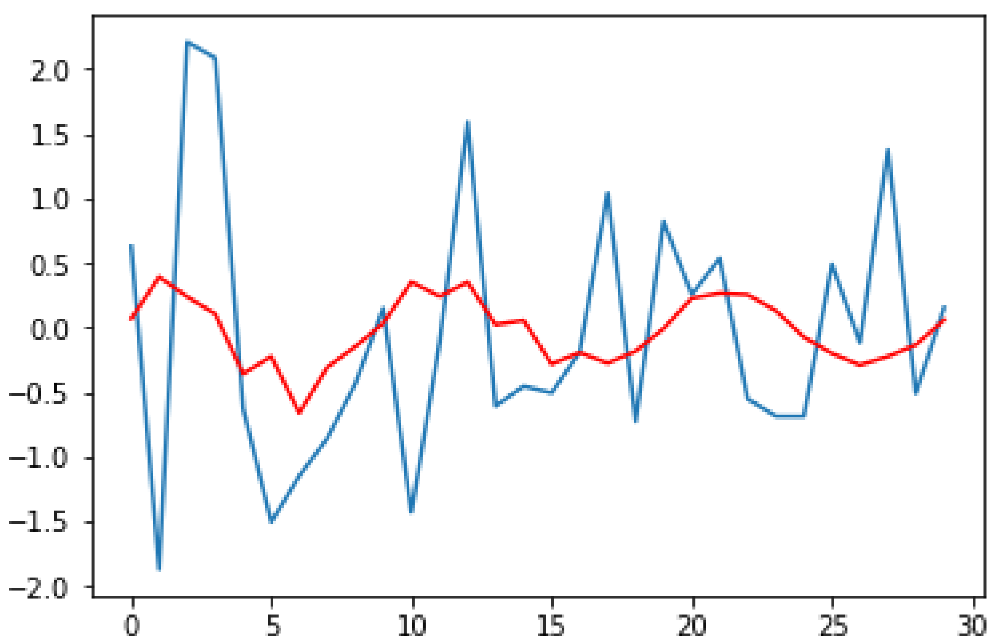
Lag: 27
RMSE: 1.091



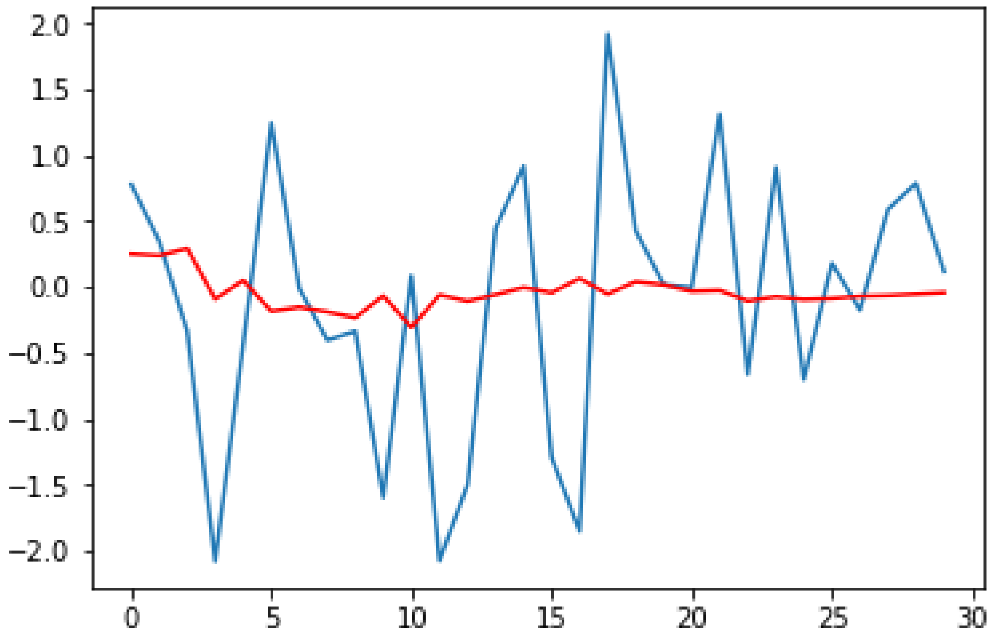
Lag: 27
RMSE: 0.809



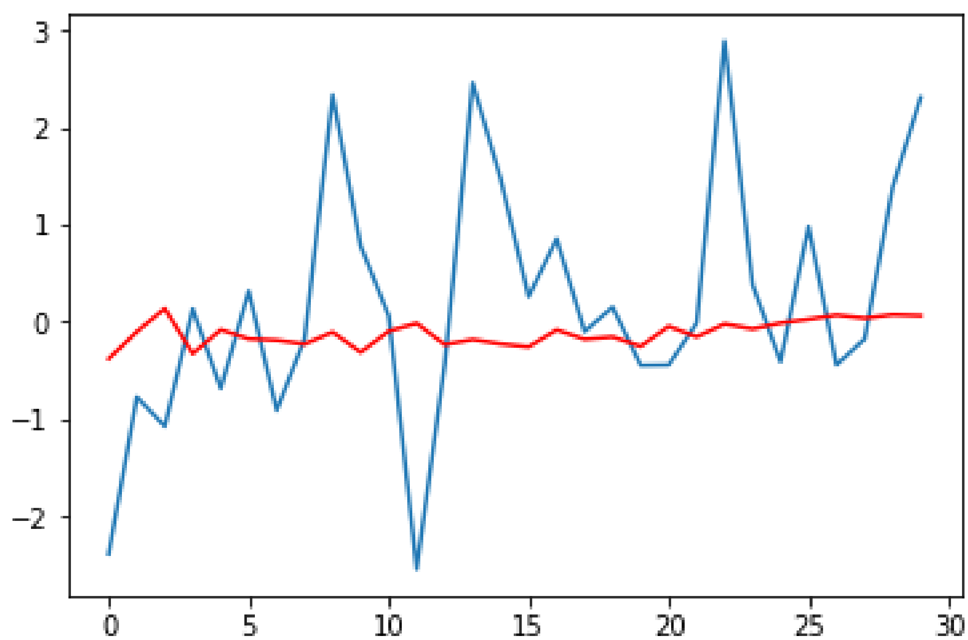

```
Lag: 27
RMSE: 0.981
```



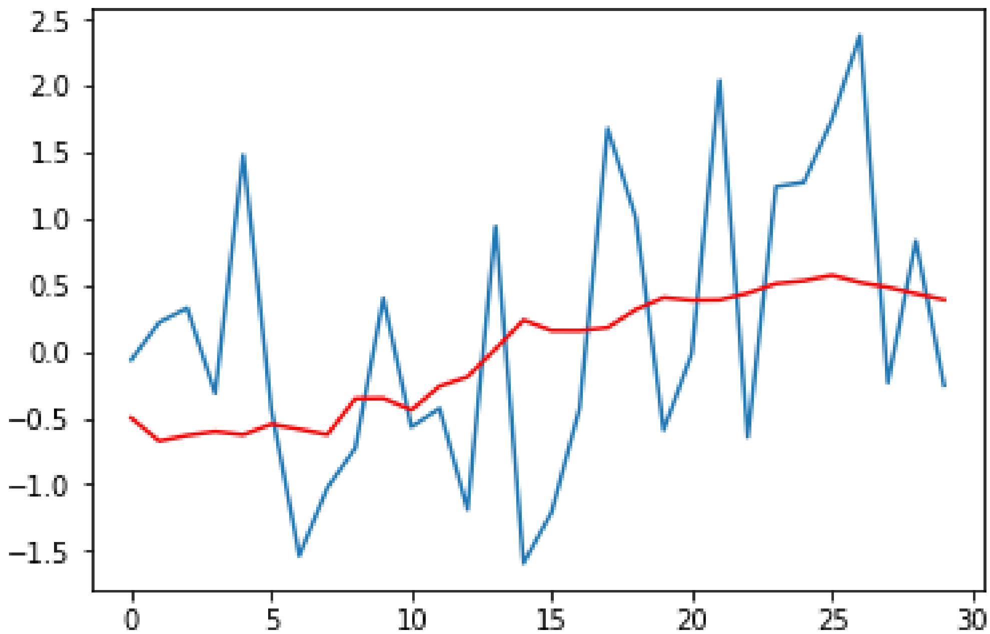
Lag: 27
RMSE: 1.010



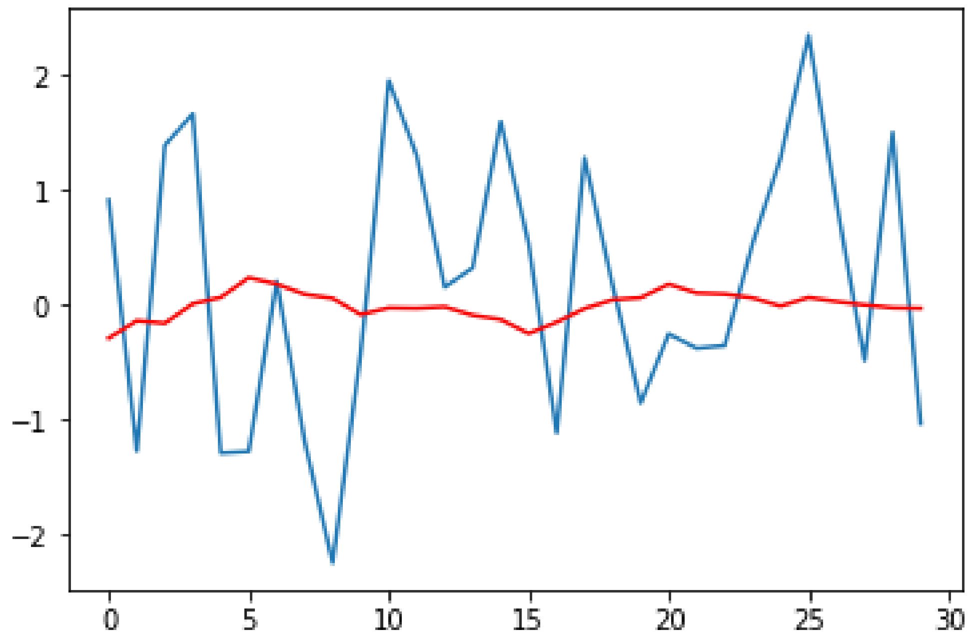
Lag: 27
RMSE: 1.277



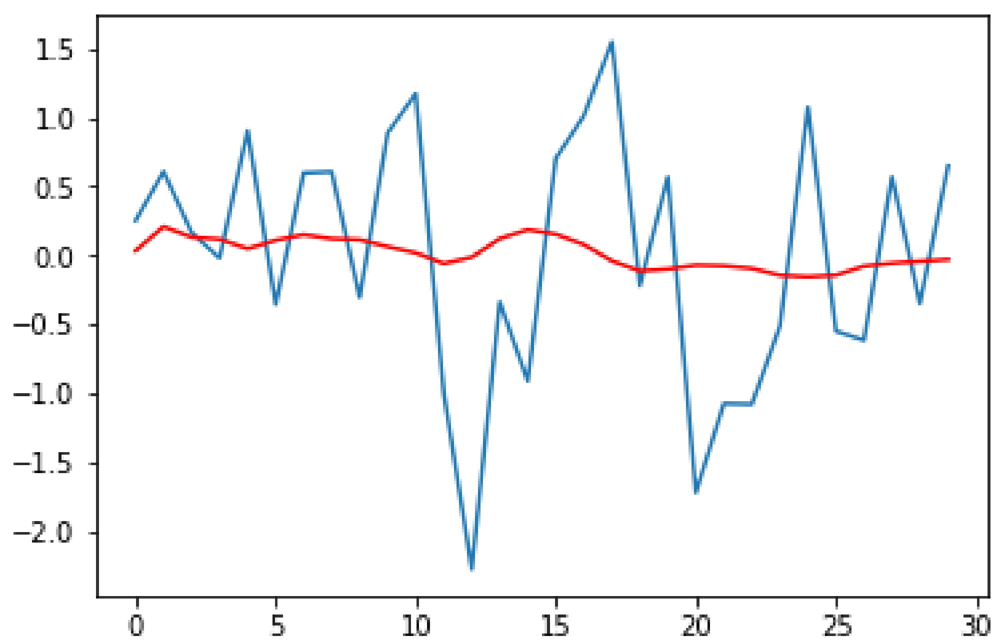
```
Lag: 27
RMSE: 1.007
```



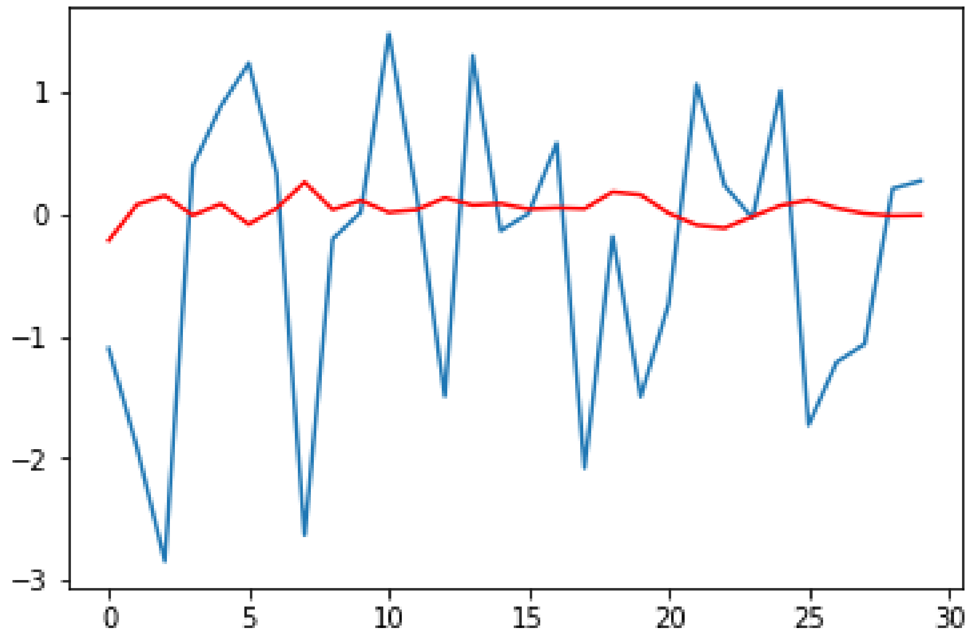

```
Lag: 27
RMSE: 1.212
```



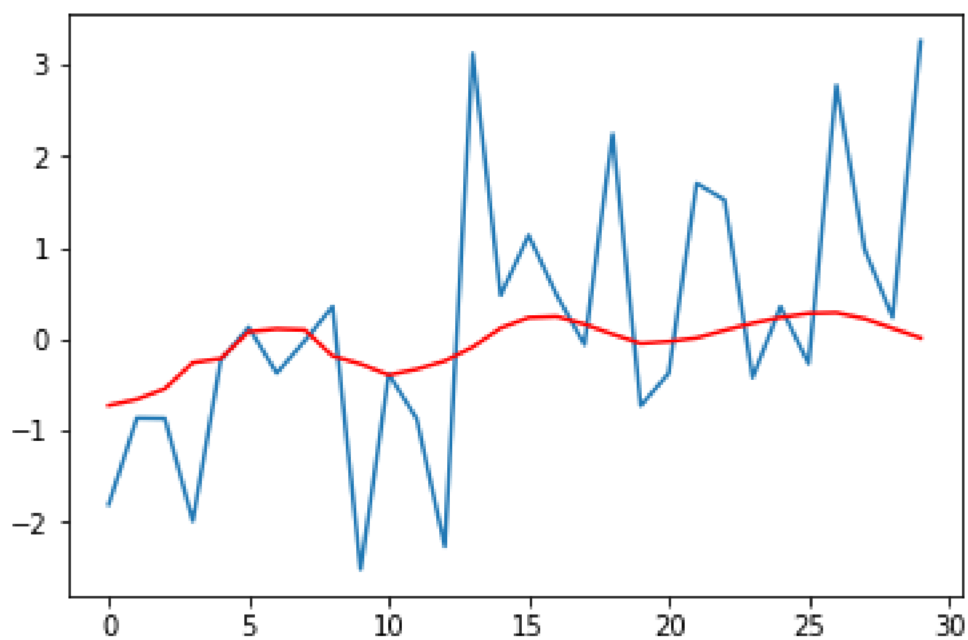

```
Lag: 27
RMSE: 0.878
```



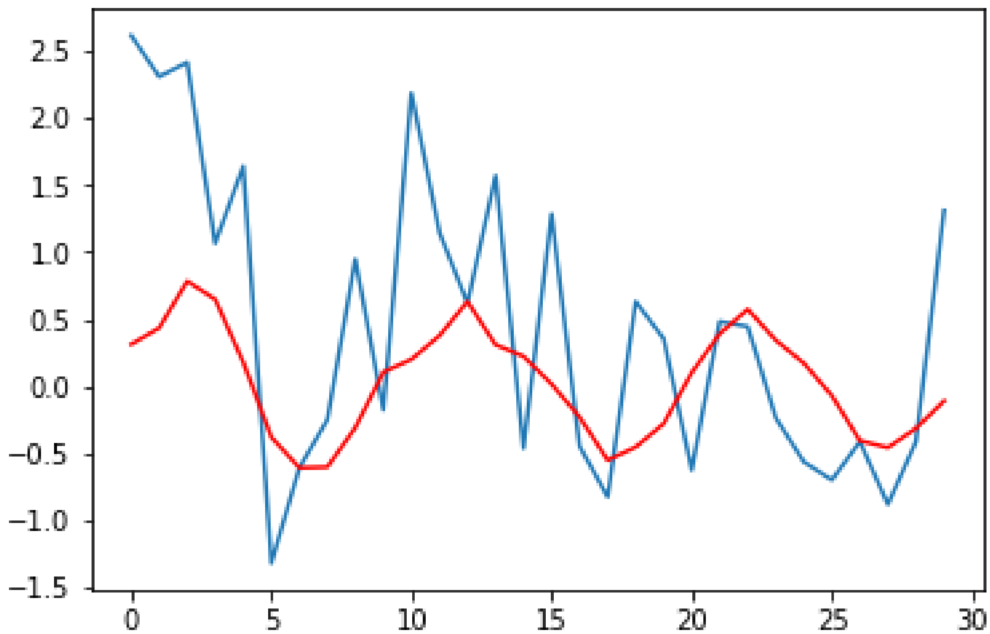
```
Lag: 27
RMSE: 1.264
```



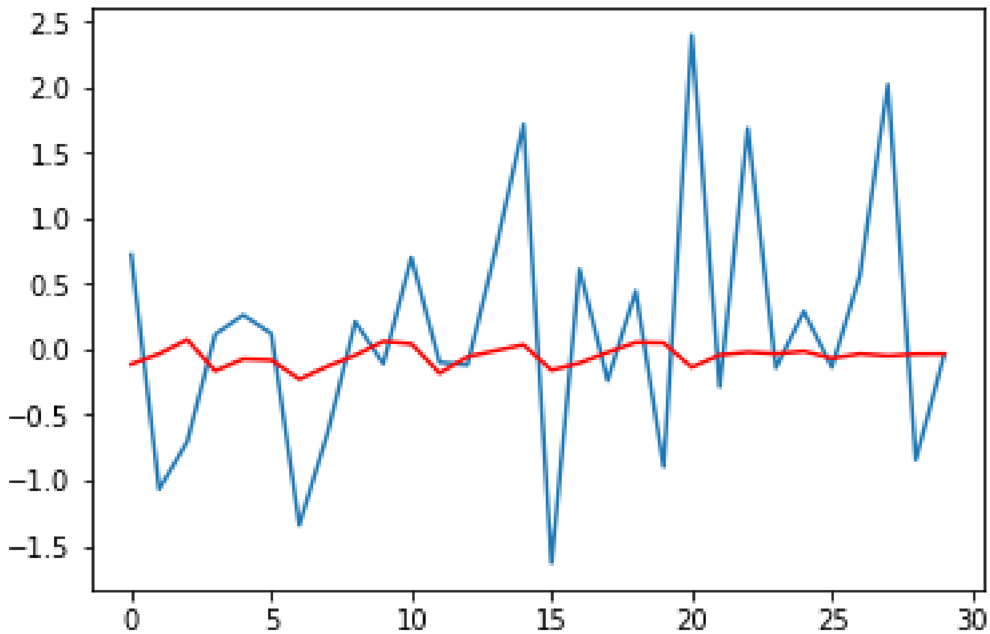

```
Lag: 27
RMSE: 1.339
```



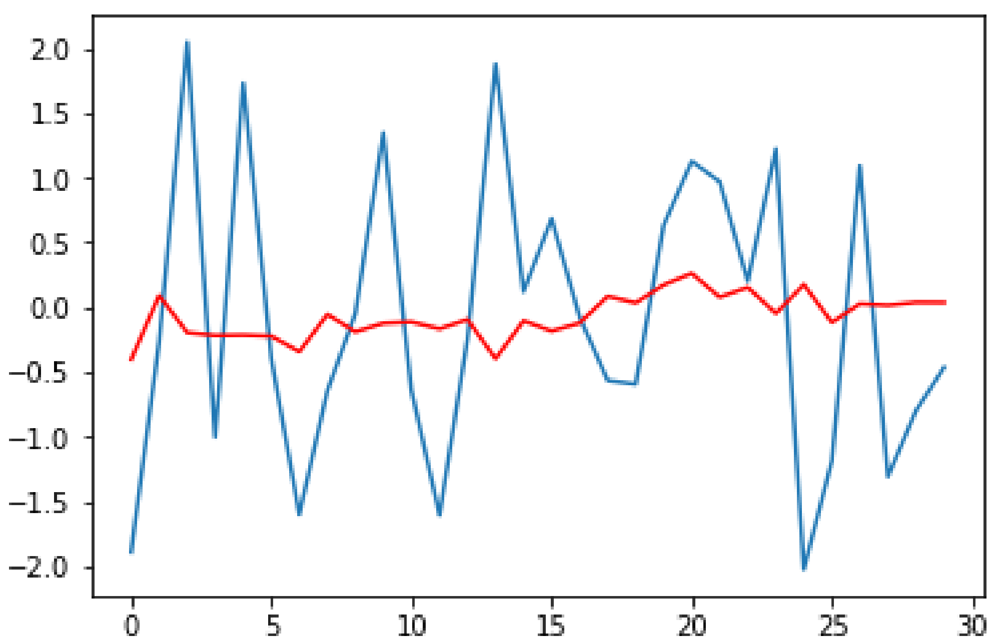
```
Lag: 27
RMSE: 1.005
```



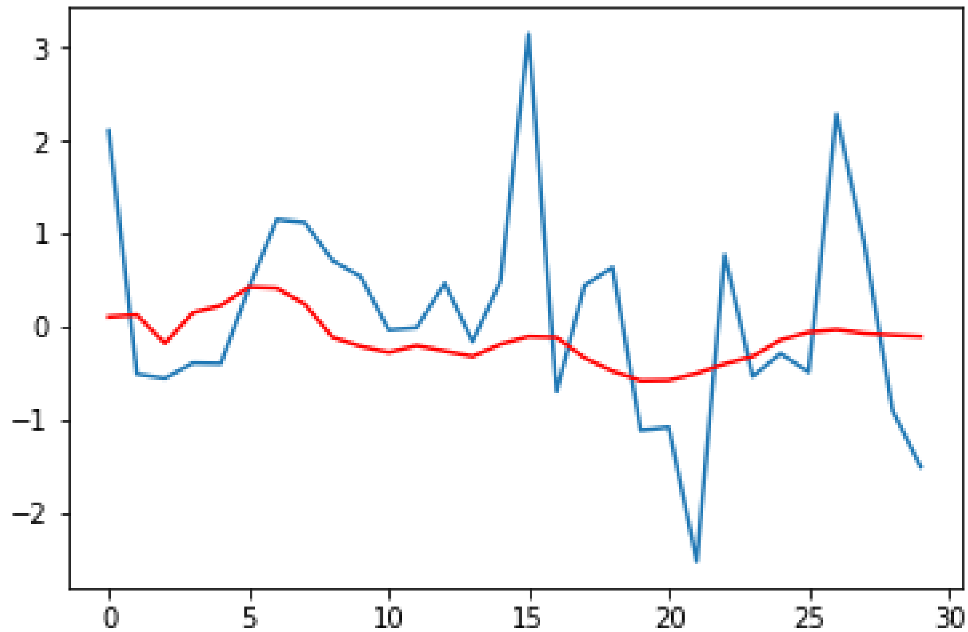

```
Lag: 27
RMSE: 0.942
```



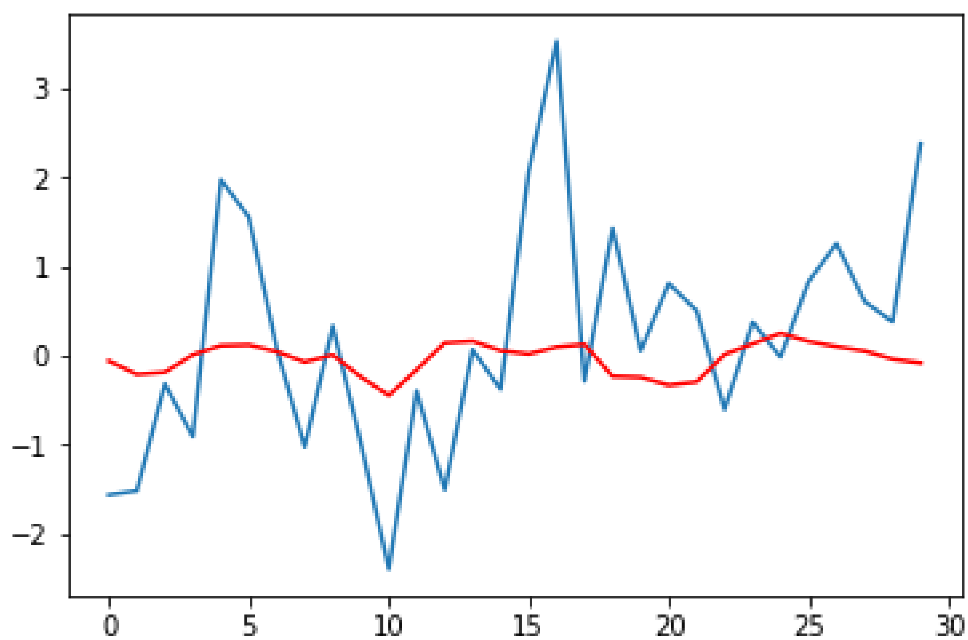

```
Lag: 27
RMSE: 1.128
```



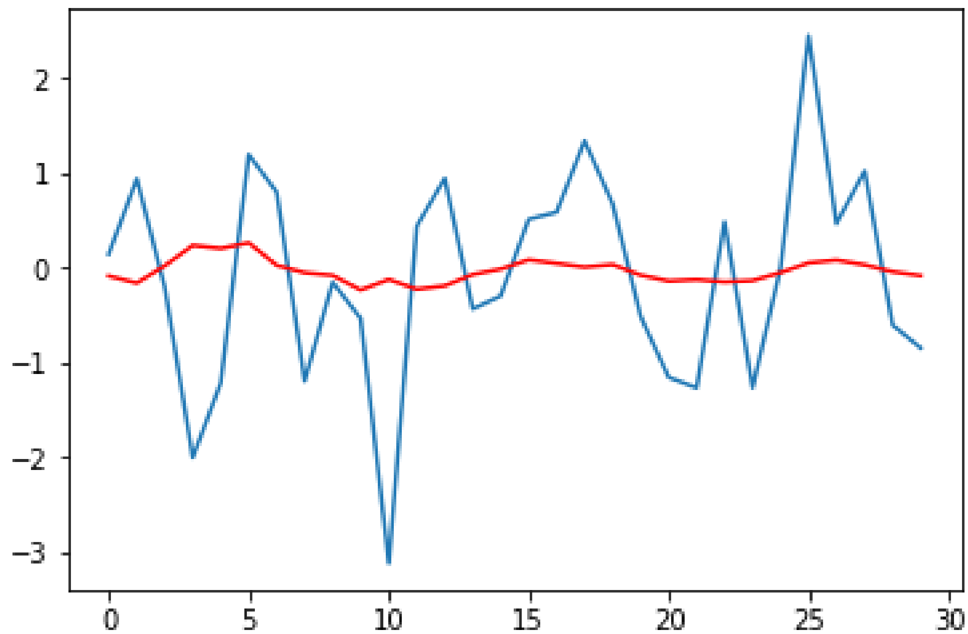
```
Lag: 27
RMSE: 1.107
```



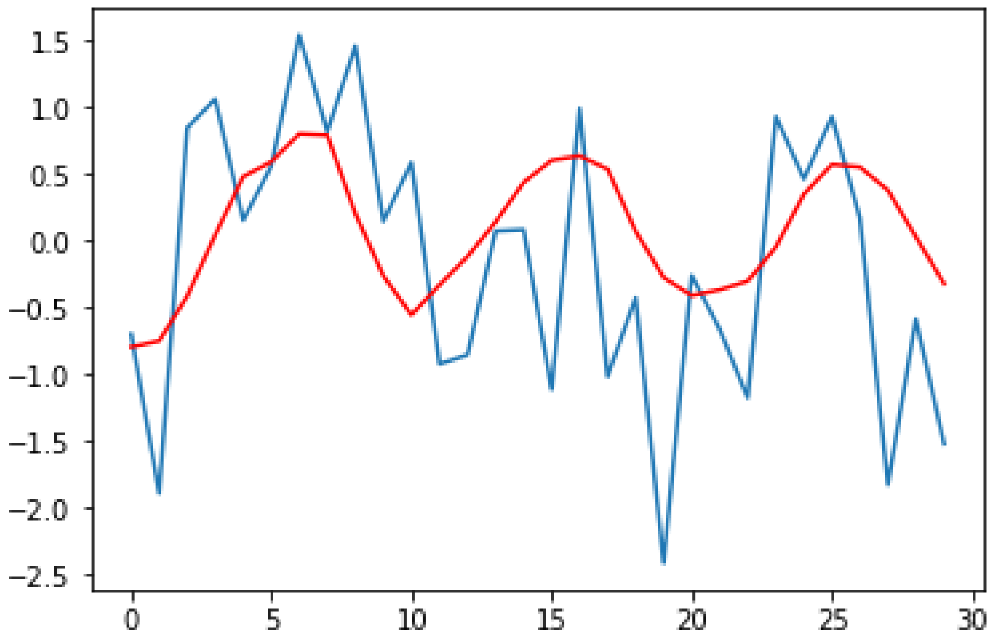

```
Lag: 27
RMSE: 1.270
```



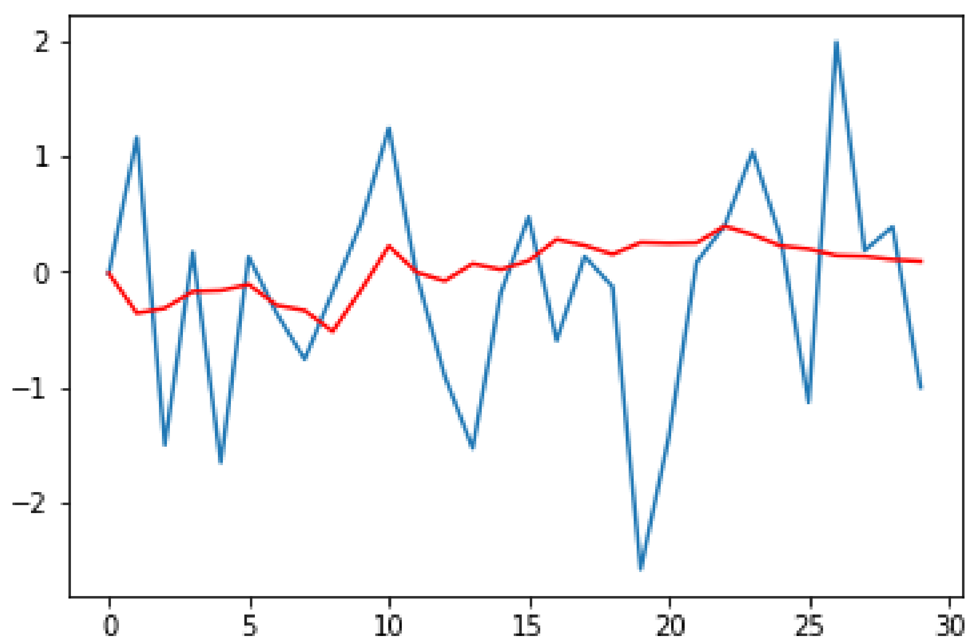
```
Lag: 27
RMSE: 1.106
```



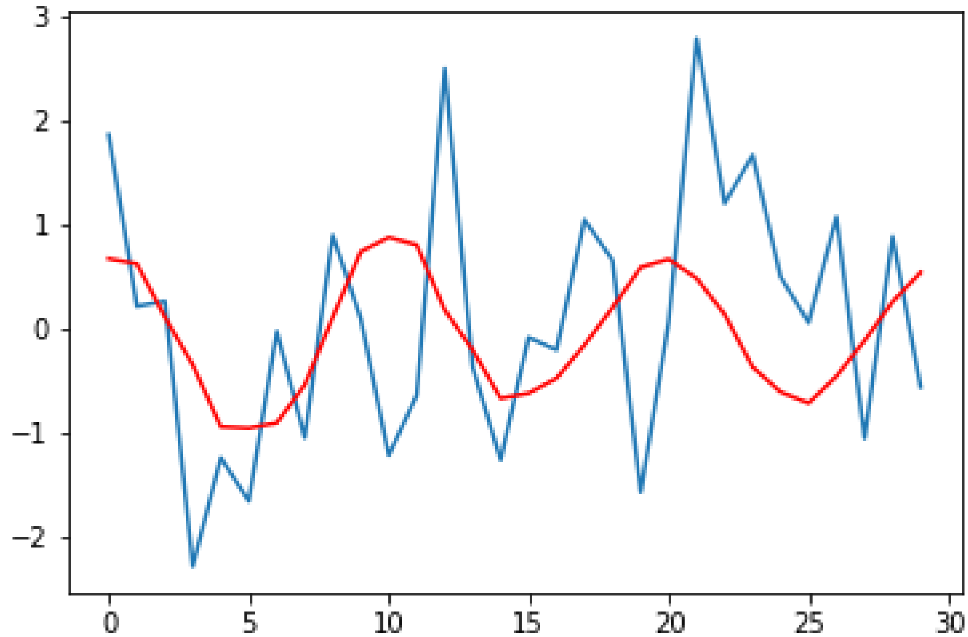

```
Lag: 27
RMSE: 0.962
```



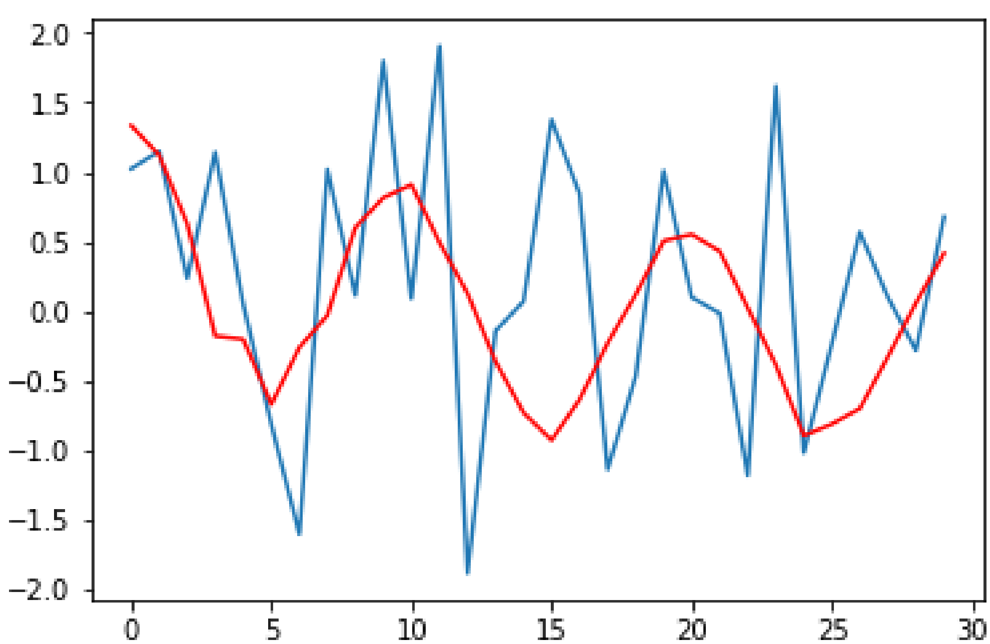

```
Lag: 27
RMSE: 0.999
```



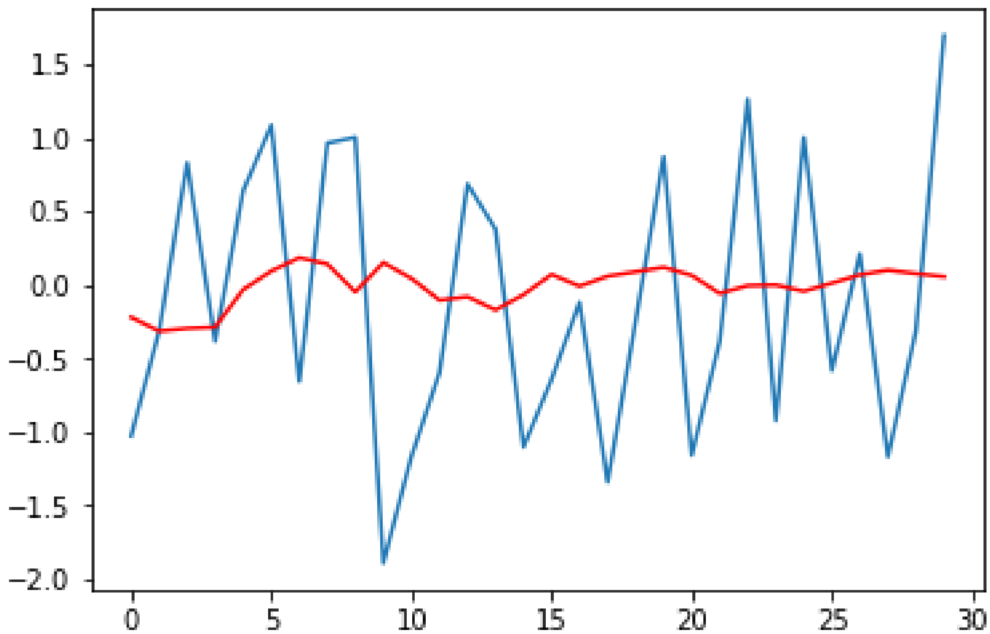
```
Lag: 27
RMSE: 1.219
```



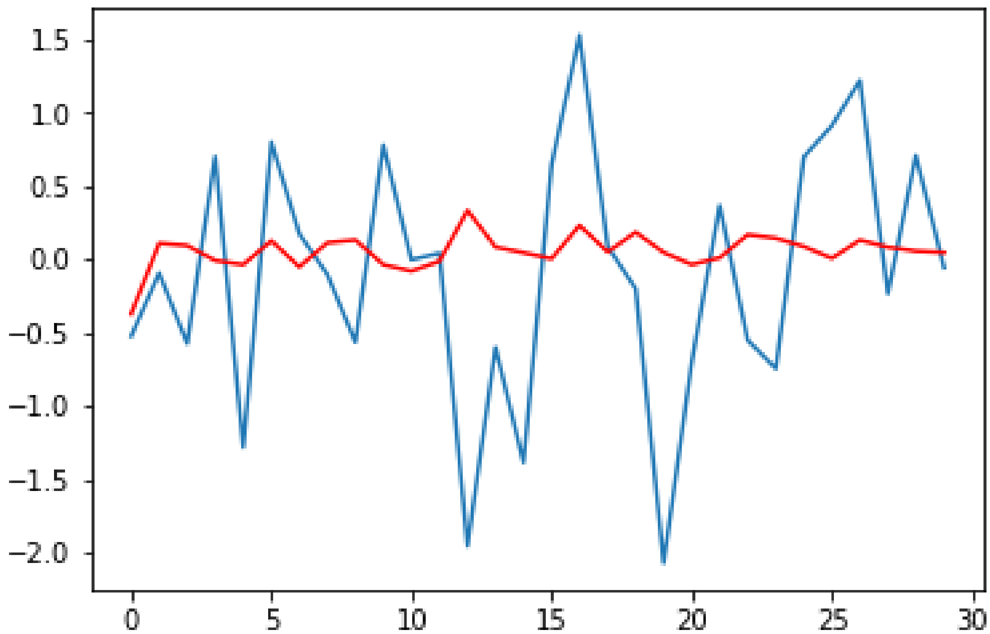

```
Lag: 27
RMSE: 1.013
```



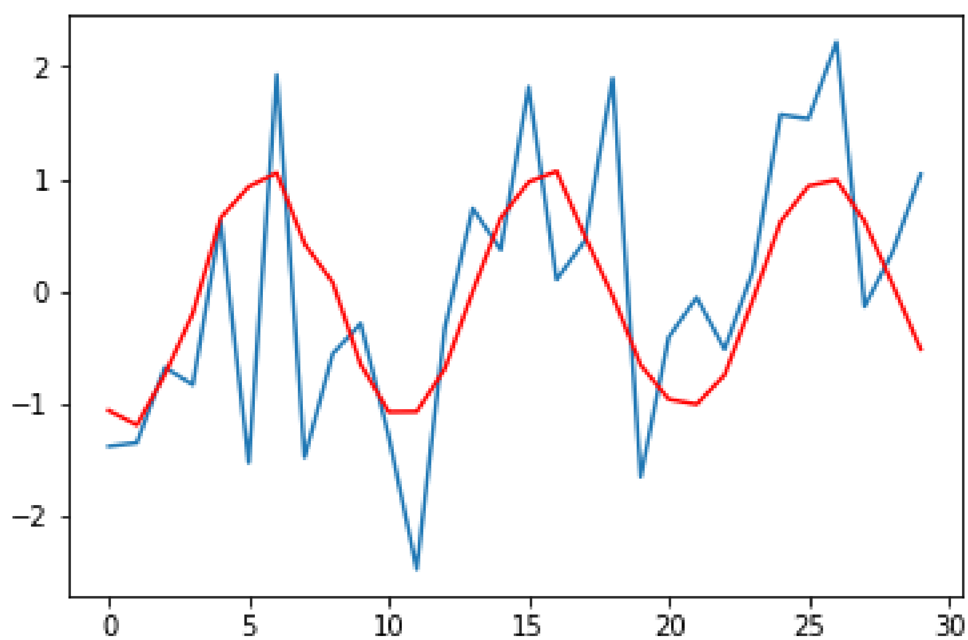
```
Lag: 27
RMSE: 0.947
```



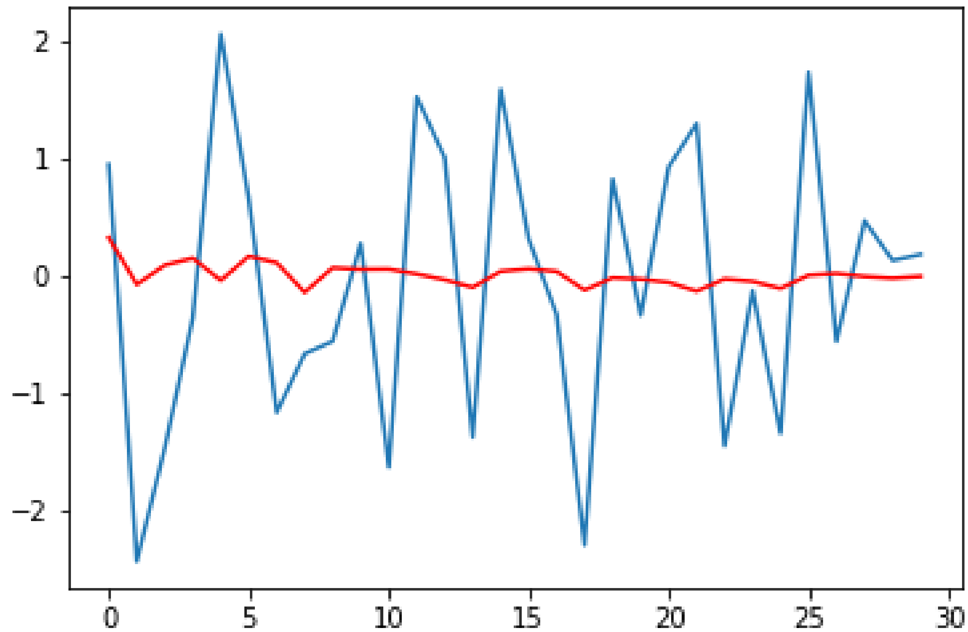

```
Lag: 27
RMSE: 0.885
```



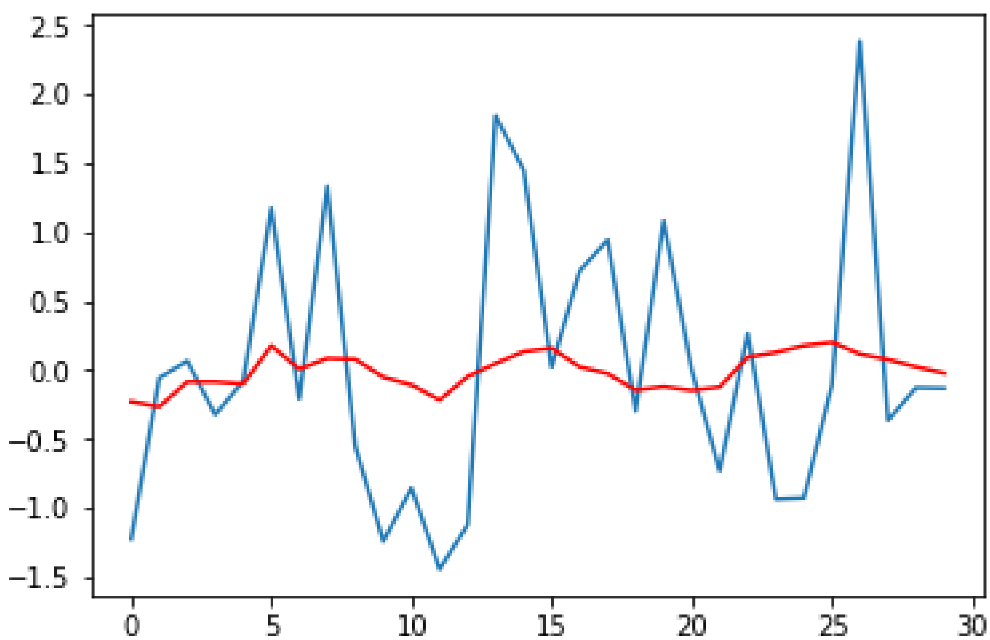

```
Lag: 27
RMSE: 0.962
```



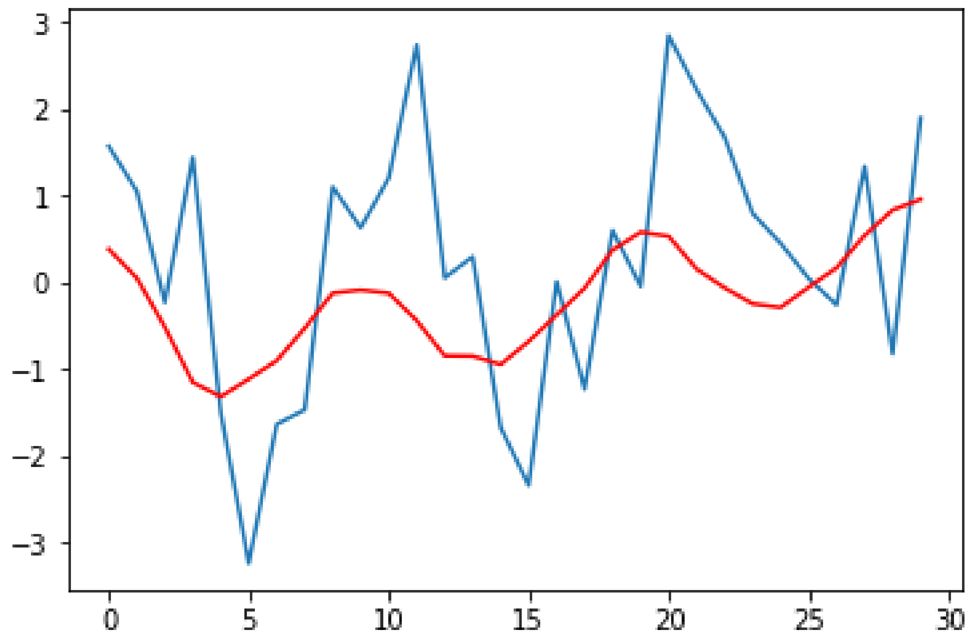
Lag: 27
RMSE: 1.182



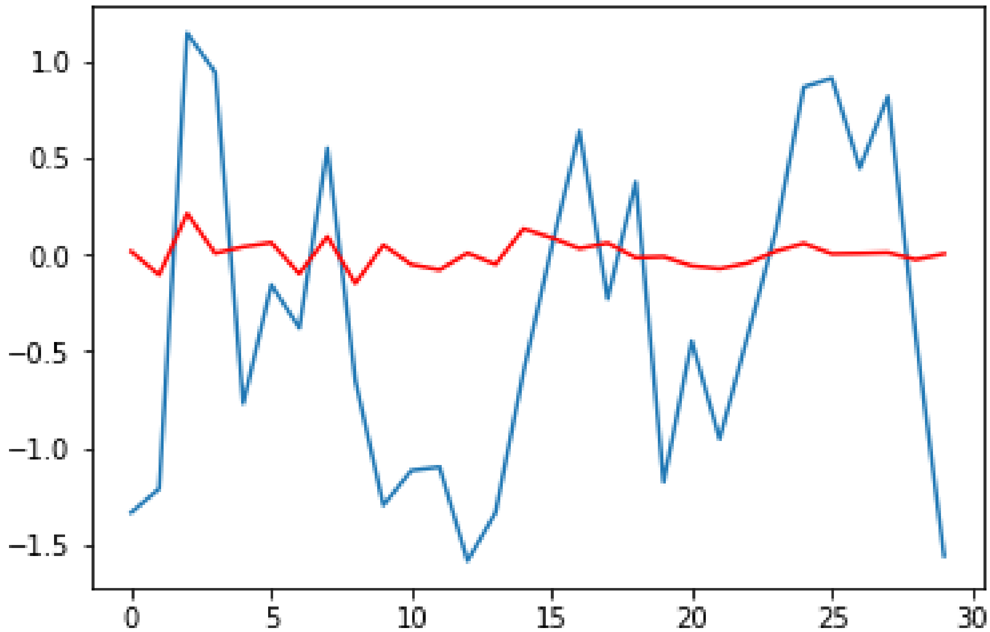
Lag: 27
RMSE: 0.909



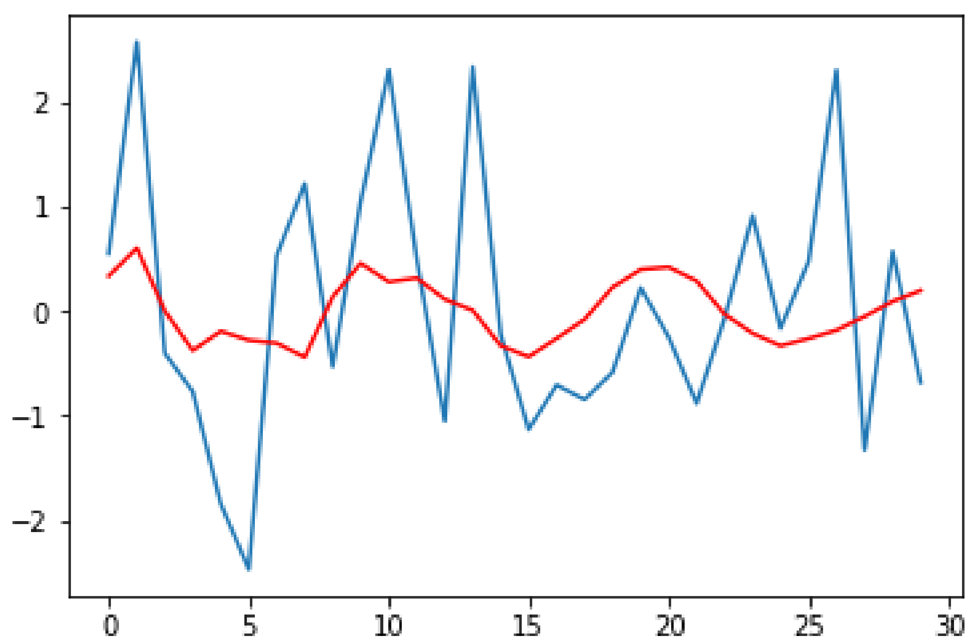
```
Lag: 27
RMSE: 1.356
```



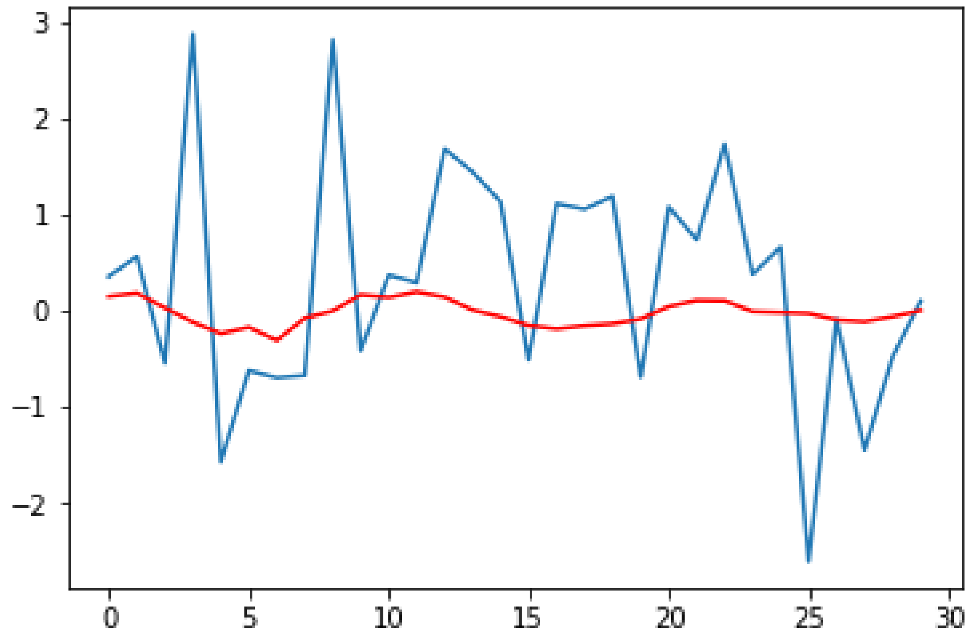

```
Lag: 27
RMSE: 0.872
```



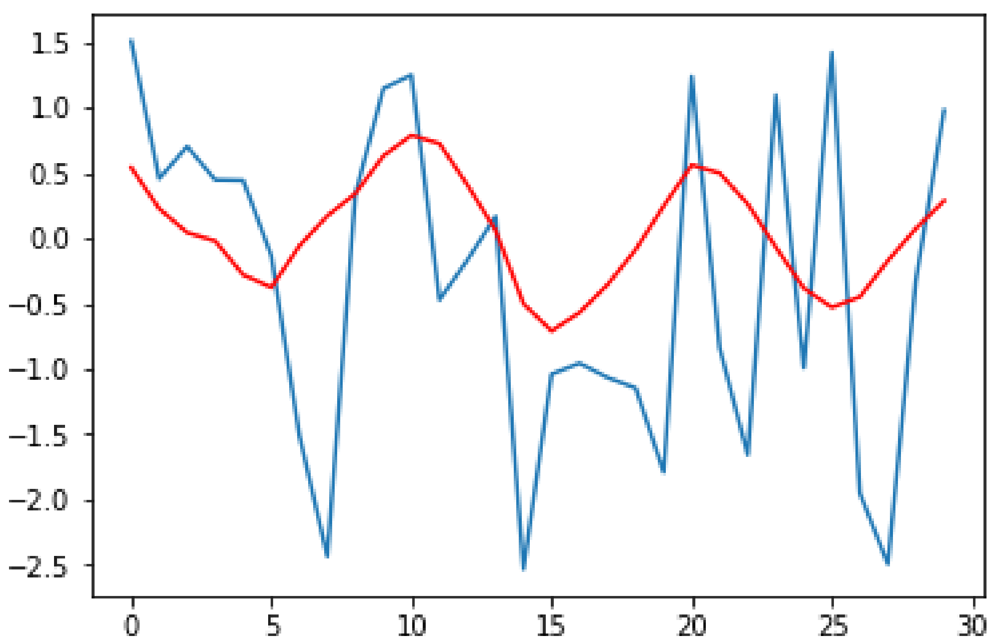

```
Lag: 27
RMSE: 1.174
```



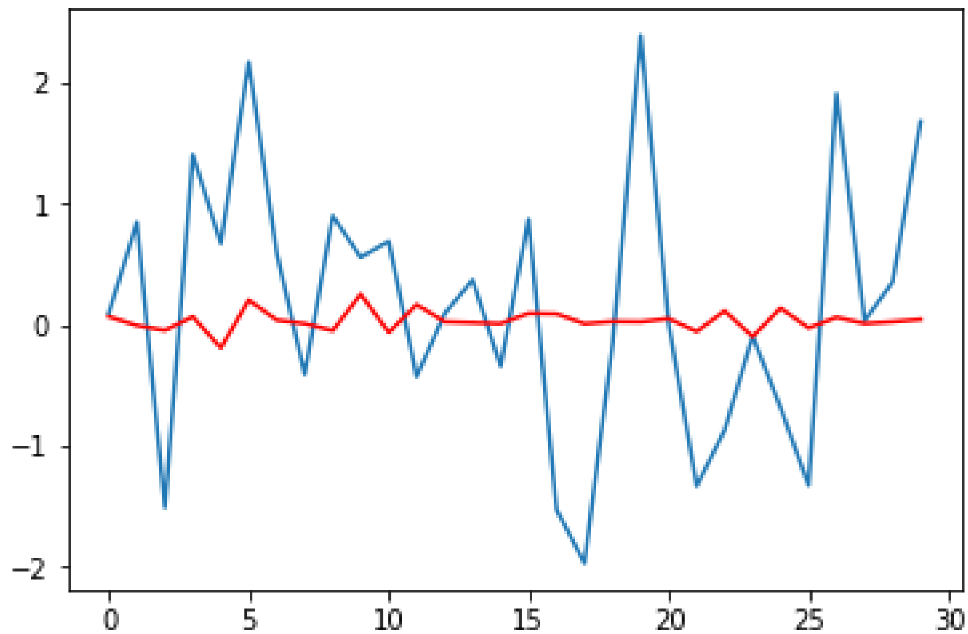
```
Lag: 27
RMSE: 1.225
```



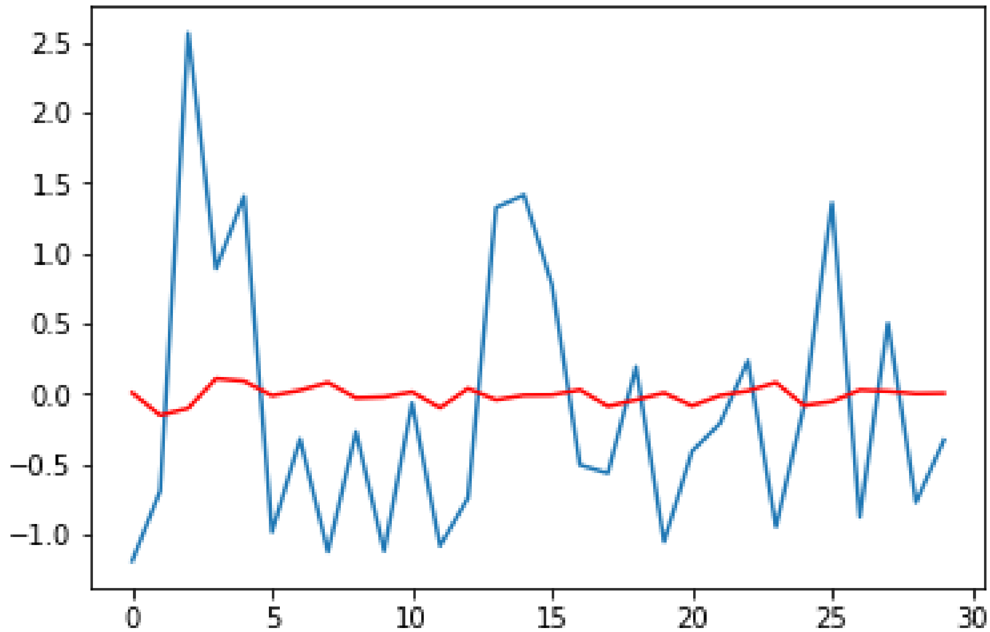

```
Lag: 27
RMSE: 1.201
```



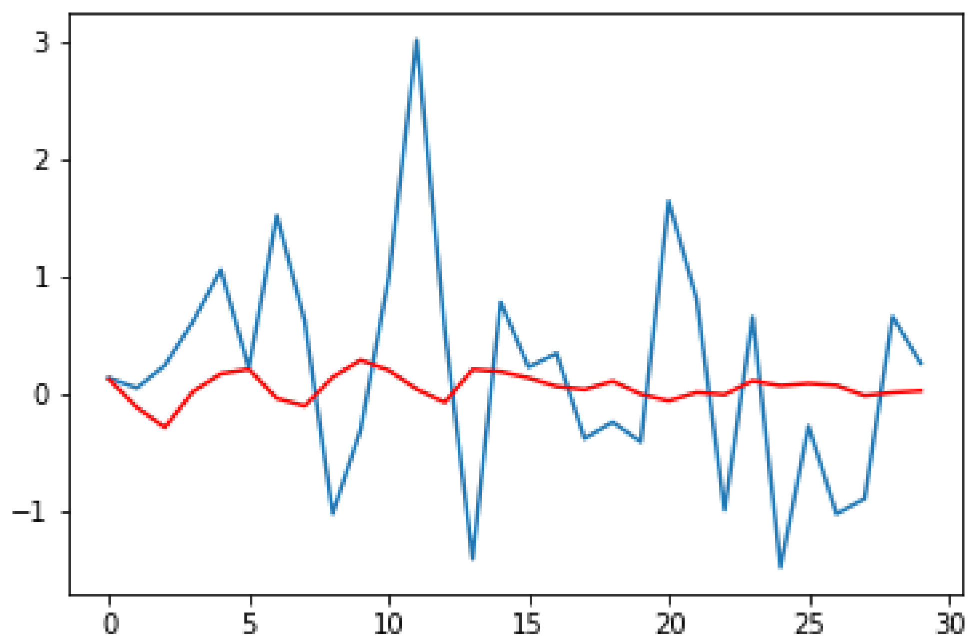
```
Lag: 27
RMSE: 1.093
```



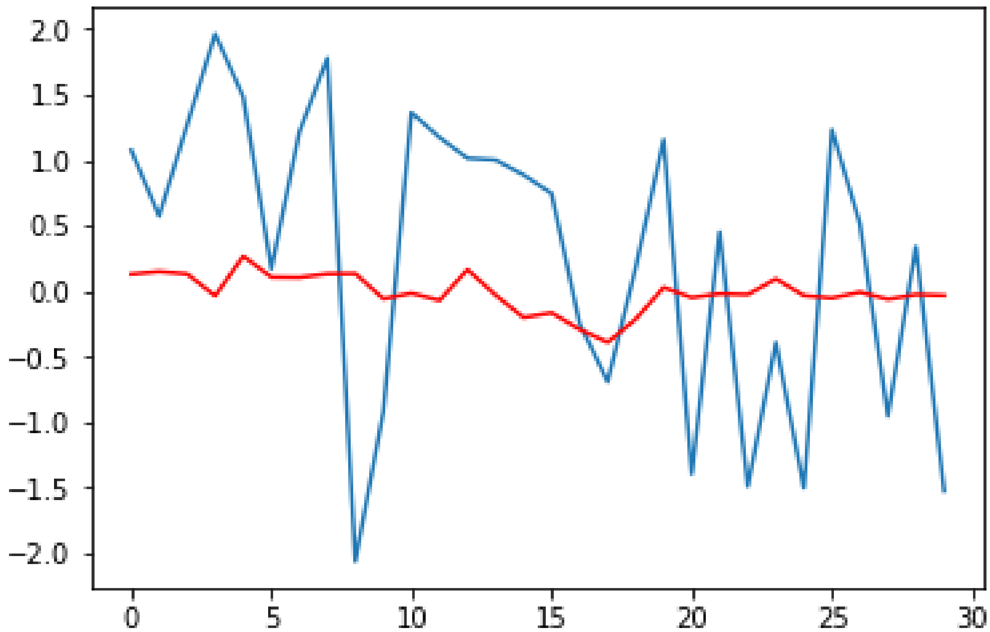

```
Lag: 27
RMSE: 0.960
```



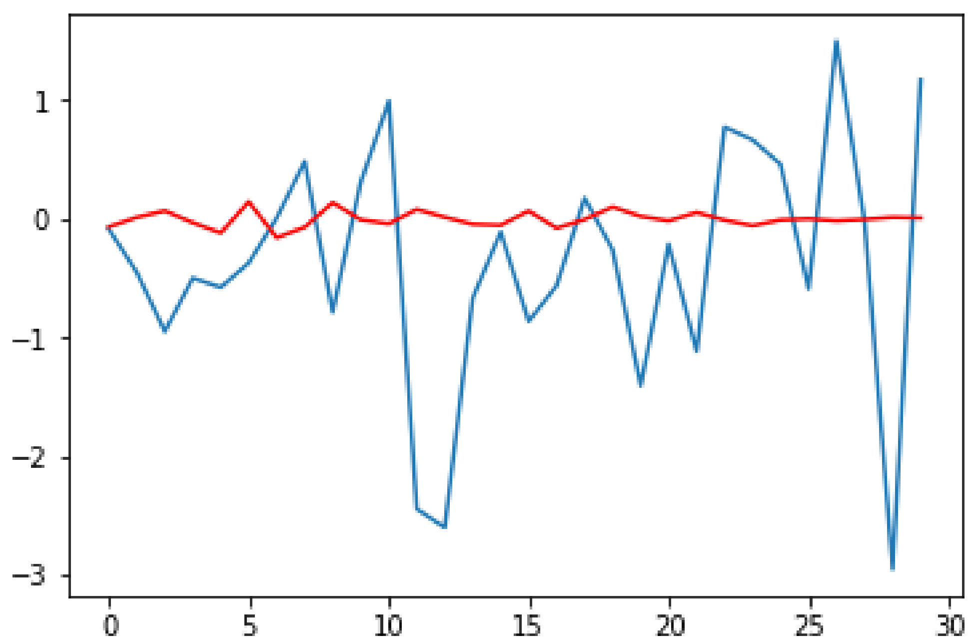

```
Lag: 27
RMSE: 0.988
```



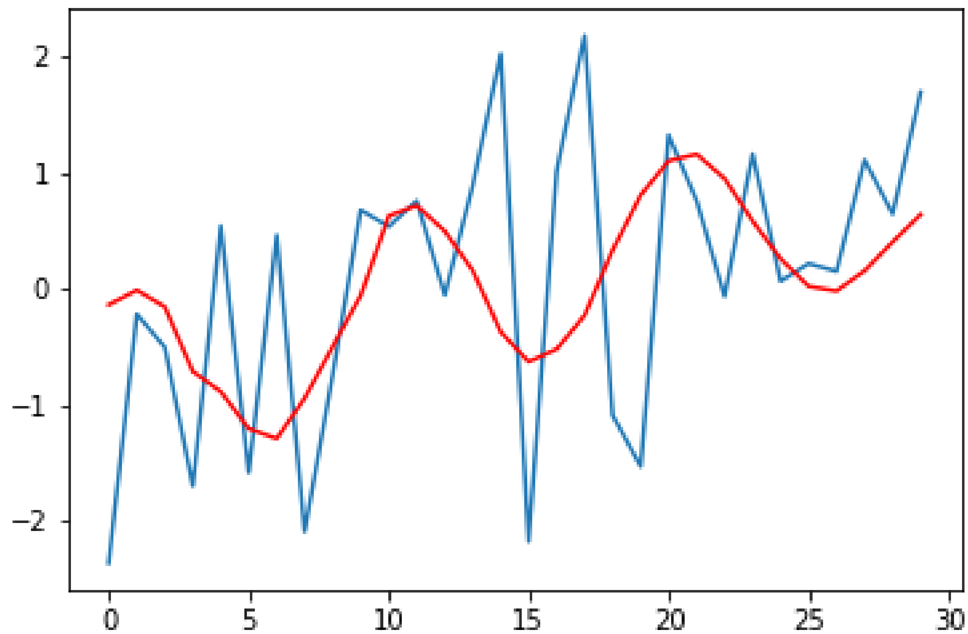
```
Lag: 27
RMSE: 1.122
```



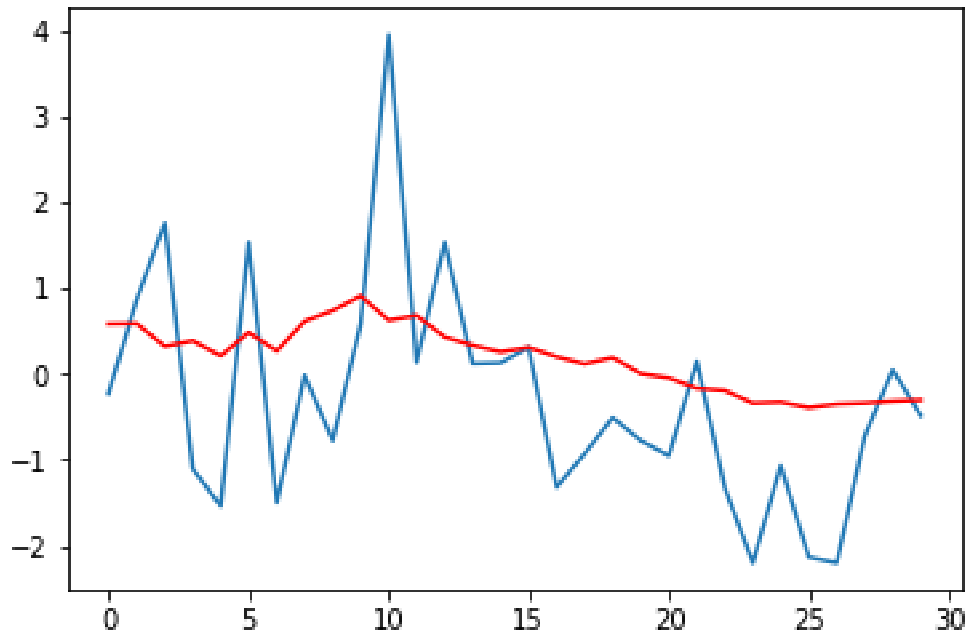

```
Lag: 27
RMSE: 1.107
```



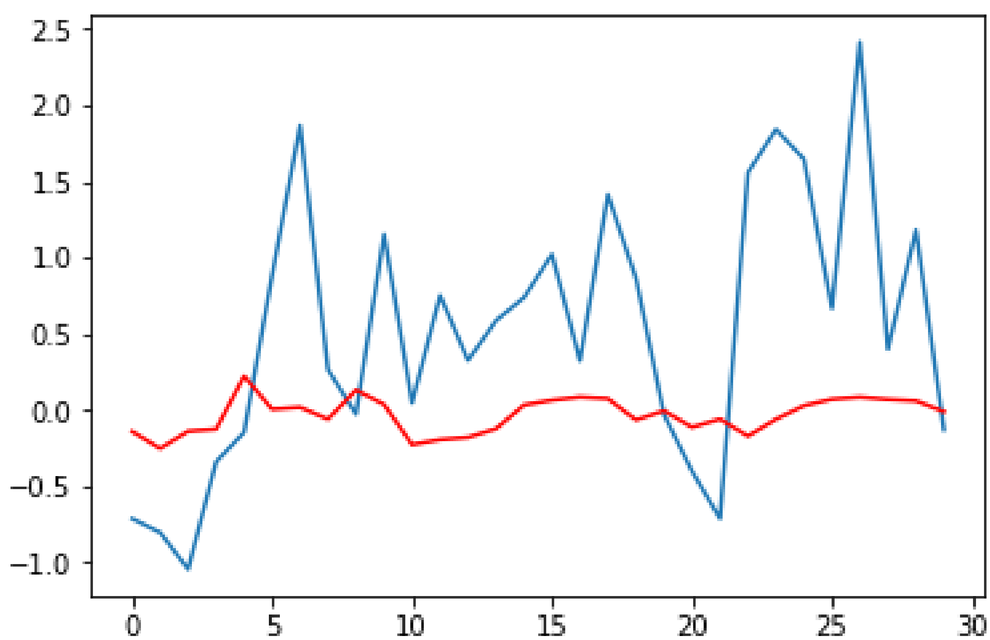
```
Lag: 27
RMSE: 1.178
```



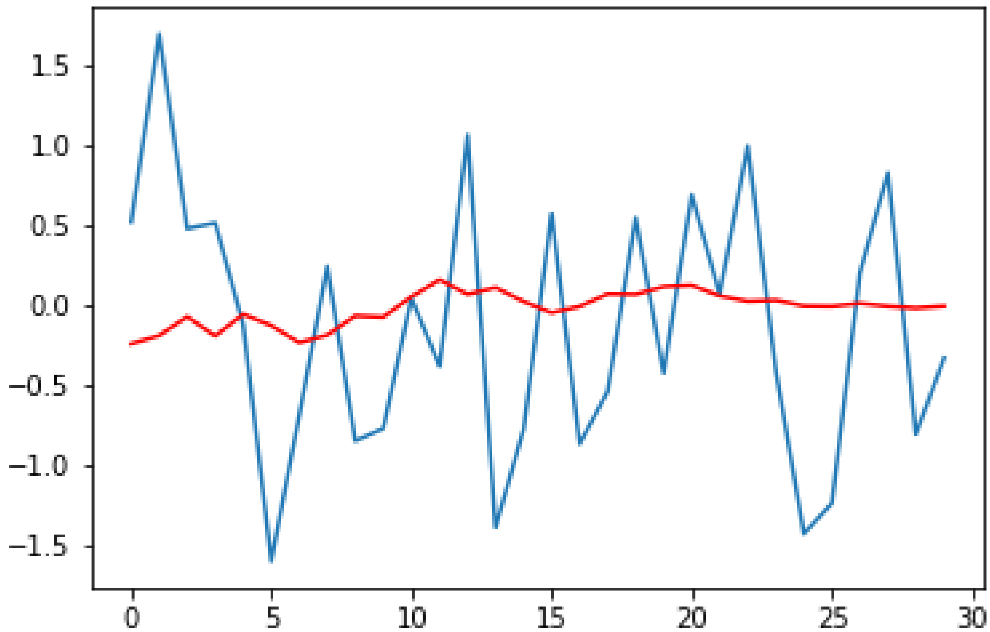

```
Lag: 27
RMSE: 1.228
```



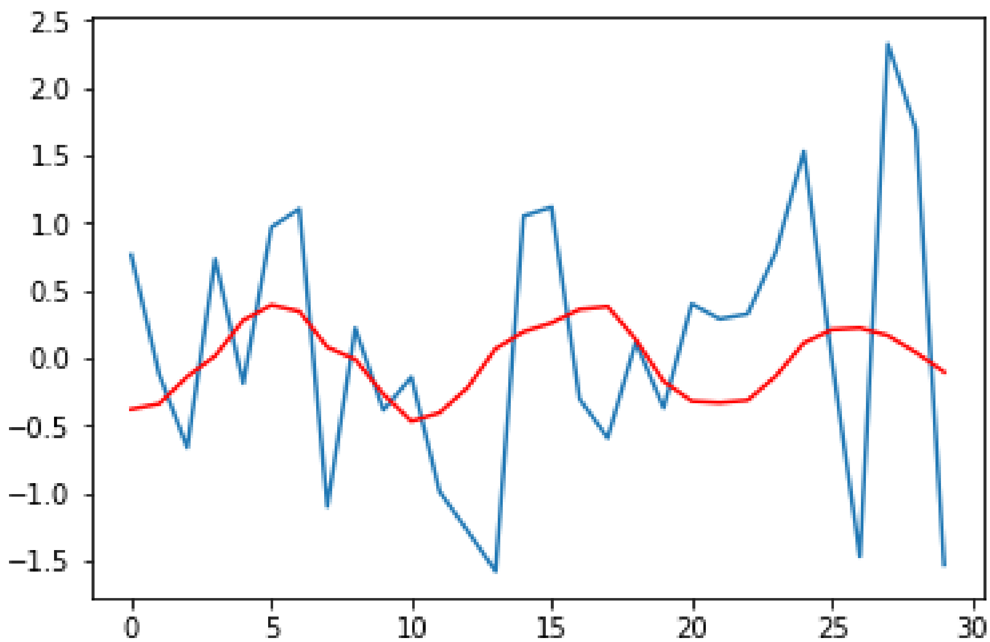

```
Lag: 27
RMSE: 0.999
```



```
Lag: 27
RMSE: 0.842
```

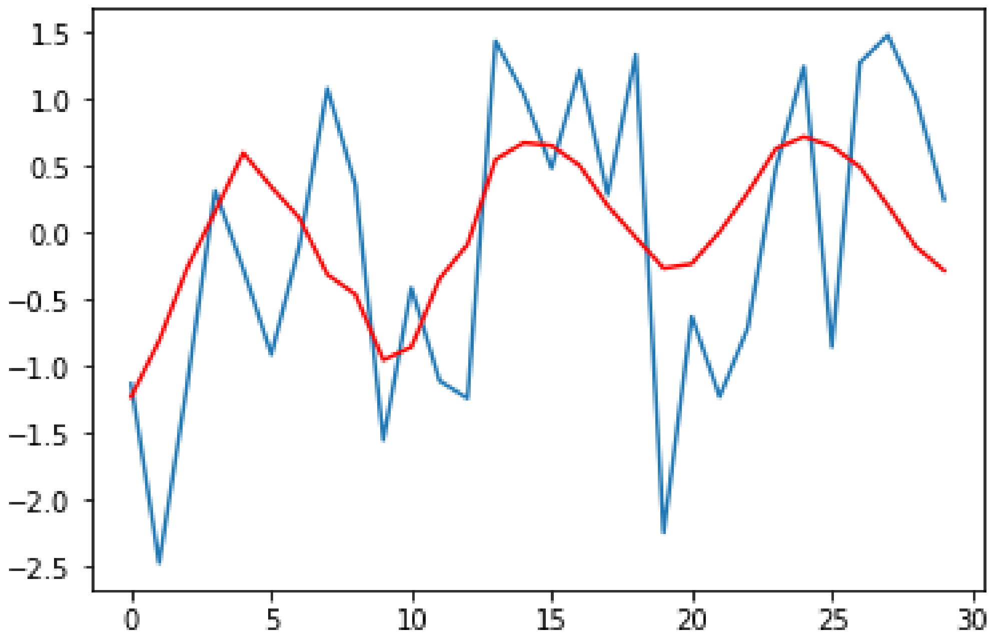



```
Lag: 27
RMSE: 0.972
```

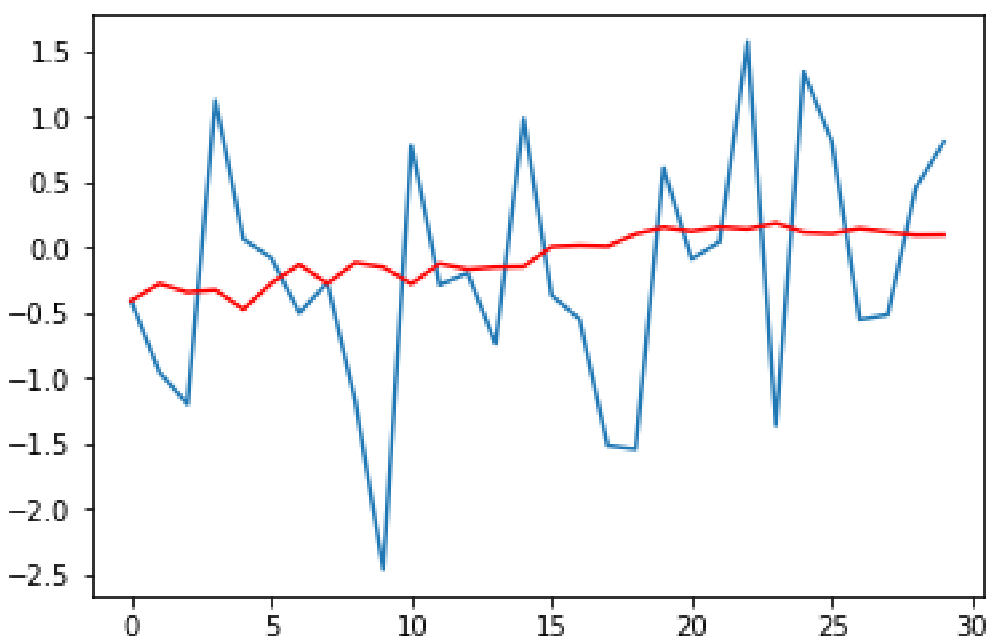


```
Lag: 27
RMSE: 0.909
```

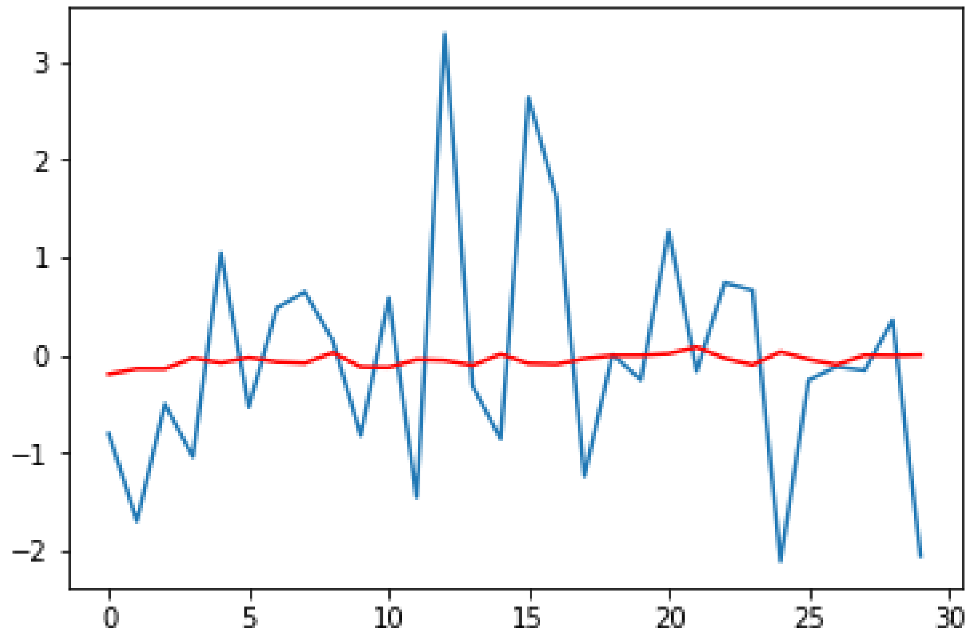

```
Lag: 27
RMSE: 0.954
```



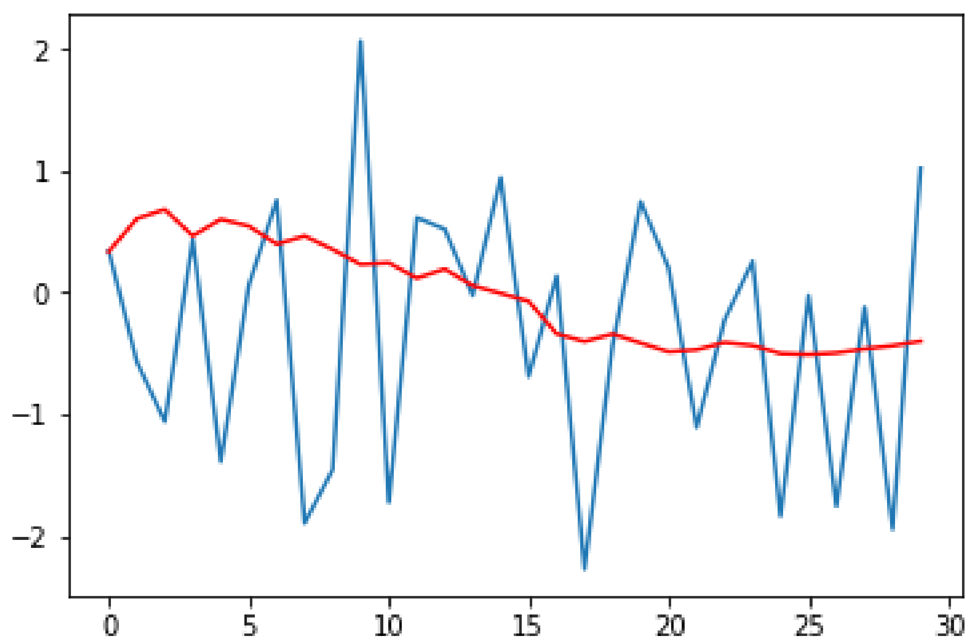

```
Lag: 27
RMSE: 0.945
```



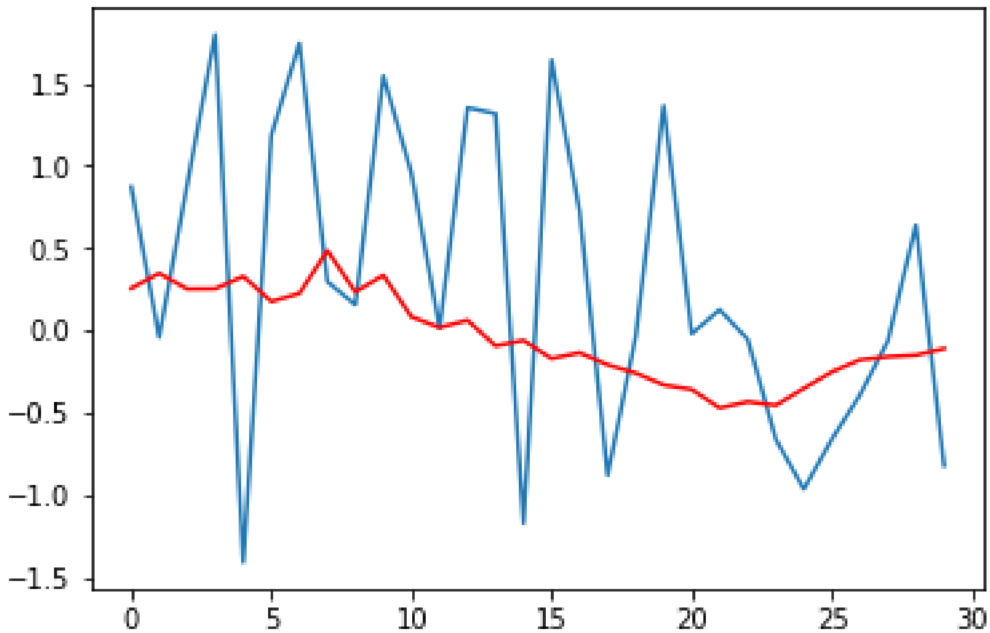
```
Lag: 27
RMSE: 1.222
```




```
Lag: 27
RMSE: 1.169
```



```
Lag: 27
RMSE: 0.942
```




```
In [33]: # Train on 2100 days, test on 390 days and validate on 10 days
         siz=2100
         data=data[:-11]
```

```
In [34]: from pandas import DataFrame
         from pandas import concat

         def convert(data, n_in=1, n_out=1, drop = True):
             n_vars = 1 if type(data) is list else data.shape[1]
             df = DataFrame(data)
             cols, names = list(), list()


             for i in range(n_in, 0, -1):
                 cols.append(df.shift(i))
                 names += [('sig%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
             # forecast sequence (t, t+1, ... t+n)
             for i in range(0, n_out):
                 cols.append(df.shift(-i))
```

```
            if i == 0:
                names += [('sig%d(t)' % (j+1)) for j in range(n_vars)]
            else:
                names += [('sig%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
        # put it all together
        final = concat(cols, axis=1)
        final.columns = names
        if drop:
            final.dropna(inplace=True)
        return final
```

```
In [35]: def prepare_data(series, n_test, n_lag, n_seq,scale=False, drop = True):
             # extract raw values
             raw_values = series.values
             # rescale values to -1, 1
             scaler = MinMaxScaler(feature_range=(-1, 1))
             if scale:
                 scaled_values = scaler.fit_transform(raw_values)
             else:
                 scaled_values = raw_values
             supervised_values = convert(scaled_values, n_lag, n_seq, drop)
             #supervised_values = supervised.values
             # split into train and test sets
             train, test = supervised_values[0:n_test], supervised_values[n_test:]
             return scaler, train, test
```

```
In [36]: # we choose lag number 27 as from AR model
         n_lag = 27
         # n_cluster mean the number of predicting days that grouped together for training and
         n_cluster = 10
         # 30 days to predict
         n_seq = 30
         n_test = siz
         n_batch = 1
         n_neurons = 50

         # fit an LSTM network to training data
         def fit_lstm(train, n_lag=n_lag, n_seq=n_seq, n_batch=n_batch,n_cluster = n_cluster):
             # reshape training into [samples, timeseps, features]
             X, y = train.values[:, 0:n_lag*df.shape[1]], train.values[:, n_lag*df.shape[1]:]
             X = X.reshape(int(X.shape[0]/n_cluster), n_cluster, X.shape[1])
             y = y.reshape(int(y.shape[0]/n_cluster), n_cluster,y.shape[1])
             # design network
             model = Sequential()
             model.add(LSTM(n_neurons, recurrent_dropout =0.5,batch_input_shape=(n_batch, X.sha
             model.add(Dropout(0.5))
             model.add(Dense(y.shape[-1]))
             model.compile(loss='mean_squared_error', optimizer='adam')
```

```
    model.fit(X, y, epochs=20, batch_size=n_batch, verbose=0, shuffle=False)

    return model

# prediction with LSTM
def forecast_lstm(model, X, n_batch):
    # reshape input pattern to [samples, timesteps, features]
    X = X.reshape(1, X.shape[0], X.shape[1])
    # make forecast
    forecast = model.predict(X, batch_size=n_batch)
    # convert to array
    return [x for x in forecast[0, :]]


def make_forecasts(model,test,n_batch=n_batch,  n_lag=n_lag, n_seq=n_seq):
    forecasts = list()
    for i in range(int(len(test)/n_cluster)):
        X = test.values[i*n_cluster:(i+1)*n_cluster, 0:n_lag*df.shape[1]]
        # make forecast
        forecast = forecast_lstm(model, X, n_batch)
        # store the forecast
        forecasts.append(forecast)
    return forecasts

# inverse data transform on forecasts
def inverse_transform(series, forecasts, scaler):
    inverted = list()
    for i in range(0,len(forecasts)):
        for j in range(0,len(forecasts[i])):
            # create array from forecast
            forecast = np.array(forecasts[i][j])
            forecast = forecast.reshape(int(len(forecast)/series.shape[1]), series.sha
            # invert scaling
            inv_scale = scaler.inverse_transform(forecast)
            inv_scale = inv_scale.reshape(len(forecast)*series.shape[1],1)
            inverted.append(inv_scale)
    return inverted

# inverse data transform on actual data
def inverse_transform2(series, forecasts, scaler):
    inverted = list()
    for i in range(0,len(forecasts)):
        forecast = np.array(forecasts[i])
        forecast = forecast.reshape(int(len(forecast)/series.shape[1]), series.shape[
        # invert scaling
        inv_scale = scaler.inverse_transform(forecast)
        inv_scale = inv_scale.reshape(len(forecast)*series.shape[1],1)
        inverted.append(inv_scale)
```

```
        return inverted

    # evaluate the RMSE for each forecast time step
    def evaluate_forecasts(test, forecasts):
        r = 0
        for i in range(len(test)):
            r2 = np.sqrt(metrics.mean_squared_error(test[i], forecasts[i]))
            r = r+r2
        return r/len(test)
```

```
In [37]: # Test set result
         df = data
         # prepare data
         scaler, train, test = prepare_data(df, n_test, n_lag, n_seq,True)
         # fit model
         model = fit_lstm(train, n_lag, n_seq, n_batch)
         # make forecasts
         forecasts = make_forecasts(model, test,n_batch, n_lag, n_seq)
         # inverse transform forecasts and test
         predict = inverse_transform(df, forecasts, scaler)
         actual = [row[n_lag*df.shape[1]::,] for row in test.values]
         actual = inverse_transform2(df, actual, scaler)
         # evaluate forecasts|
         print(evaluate_forecasts(actual, predict))
```

```
0.994716702696419
```

```
In [38]: # test2 is used as the variables for validation set
         # test3 is used as the actual values
         scaler2, train2, test2 = prepare_data(data2, n_test, n_lag, n_seq,True)
         test2=test2[-10:]
         scaler2, train2, test3 = prepare_data(data2, n_test, n_lag, n_seq)
         test3=test3[-10:]
         forecasts = make_forecasts(model, test2,n_batch, n_lag, n_seq)
         # inverse transform forecasts and test
         predict = inverse_transform(df, forecasts, scaler)
         actual = [row[n_lag*df.shape[1]::,] for row in test3.values]
         # evaluate validation set forecasts
         print(evaluate_forecasts(actual, predict))
```

```
1.0337580140172933
```

```
In [42]: # Final prediction
         _, _, final = prepare_data(data2, n_test, n_lag, n_seq,False,False)
         final=final[-10:]
         forecasts = make_forecasts(model, final,n_batch, n_lag, n_seq)
         # inverse transform forecasts
```

```
predict = inverse_transform(data, forecasts, scaler)
# Take the 10th forecast as we have cluster size of 10 and reshape to it 30*50
final_result=predict[-1].reshape(30,50)
# Generate csv file
final_result = pd.DataFrame(final_result)
final_result.to_csv("final2.csv")
```

In [43]:
```
final.dropna(1)
```

Out[43]:
```
      sig1(t-27)  sig2(t-27)  sig3(t-27)  sig4(t-27)  sig5(t-27)  sig6(t-27)  \
2547    1.023995    1.098767    0.928377   -0.392709   -1.288621   -1.023378
2548    0.663519    1.085677    0.445288   -0.477389    1.858685   -1.947025
2549   -0.171672    0.592475   -0.409131   -1.020766    0.351257    2.104636
2550   -0.018651    0.936988    0.987878    0.155703    0.739478    0.121936
2551   -0.078472    2.094736   -1.294050   -0.124153    3.495681    0.095026
2552   -0.903780    0.697863   -0.935506   -1.267924    2.100156   -1.169816
2553   -0.652221   -1.422612   -0.910446   -0.290991    0.913679    1.091135
2554    1.048668    1.903046   -1.501638    2.683156    1.840334   -1.582142
2555   -2.689315   -0.399452   -1.538902   -0.978872   -0.094436    0.314684
2556   -1.861011   -0.623936    0.247723    0.276927    1.209945   -0.470826

      sig7(t-27)  sig8(t-27)  sig9(t-27)  sig10(t-27)    ...     sig41(t)  \
2547    0.278406   -0.071336   -0.214636    -0.354798    ...    -0.954851
2548   -0.052591   -0.657953   -0.099666    -0.529751    ...     0.145933
2549    0.284416    0.624867   -1.229096    -0.597778    ...    -1.326808
2550   -0.477191    0.145775    2.320475     0.846425    ...    -2.199024
2551   -1.189065    0.356216    0.550881     0.191164    ...    -1.065759
2552   -0.431294    0.886631    0.844365    -0.470599    ...    -2.133004
2553    0.520054   -0.085246   -0.049766    -1.872163    ...    -2.199526
2554    0.630651    0.780158   -2.385496    -0.063602    ...    -0.723788
2555   -1.867666    0.348521   -0.771091     0.218757    ...     0.049195
2556    2.211052   -0.334658   -1.073471     0.326255    ...    -0.485799

      sig42(t)  sig43(t)  sig44(t)  sig45(t)  sig46(t)  sig47(t)  sig48(t)  \
2547 -0.400922  0.693612  0.400481 -0.022681 -0.625497 -0.086370  1.268790
2548 -0.712463  0.071106  0.289297 -0.840983 -1.226397  0.044867 -0.164480
2549  1.559445  0.998510  0.326680 -0.385958 -0.711935  1.576858  0.739895
2550  1.842882 -0.409214  0.785073  0.022055  0.487854 -1.362406  0.662929
2551  1.648119 -1.426602  1.528350  1.022540  1.246178  1.343820 -2.105422
2552  0.667818 -1.236415 -0.010889  0.943599 -0.852539  0.814588 -0.256562
2553  2.415972  0.203597 -1.465696 -0.286231  1.274926 -0.551306 -0.116863
2554  0.399894  0.830297  2.322630  0.258503  1.481236 -0.511884 -0.156466
2555  1.182034 -0.805117  1.693488 -1.356019  1.015045  0.459394  0.359686
2556 -0.129288 -0.327848 -1.526784  0.406189  0.248475  0.810656 -2.052243

      sig49(t)  sig50(t)
2547  0.199710 -0.020765
2548 -1.107409  0.124367
```

```
2549 -0.224147 -0.054940
2550  0.257968 -0.663529
2551 -1.836638 -0.965719
2552 -0.022903 -0.661562
2553 -1.754780 -0.393737
2554 -0.120989 -0.064568
2555 -1.945520  0.638668
2556  1.023888 -0.822652

[10 rows x 1400 columns]
```

In [29]:
```
data2[-10:]
```

Out[29]:
```
             sig.0001  sig.0002  sig.0003  sig.0004  sig.0005  sig.0006  \
t
2012-12-22  0.156248 -0.493691 -0.567243  0.093972 -0.761663 -0.981315
2012-12-23  0.353165  0.029515  0.364593  0.308768  0.191406 -1.239654
2012-12-24  0.863059 -0.043869  1.361134 -0.072276  0.126012 -0.451944
2012-12-25  0.733182 -0.486291  0.839700  0.840329 -3.371280  0.030356
2012-12-26  0.560619  0.799245 -0.522565 -0.549462 -1.945217  1.191896
2012-12-27  0.902644  1.079399  1.365212 -0.634848 -1.125945  0.595615
2012-12-28  0.657525  0.009220  1.989058  0.342472 -0.754242  1.622422
2012-12-29 -0.510540 -0.473336 -1.019115 -1.286821 -2.033963  1.490937
2012-12-30 -1.323555  1.695200  1.112163  0.702562  0.434066  0.649184
2012-12-31 -0.050630 -0.068840  1.482284 -0.322835 -0.347853  0.979311

             sig.0007  sig.0008  sig.0009  sig.0010    ...     sig.0041  \
t                                                      ...
2012-12-22  0.256683 -0.002616 -0.441886 -0.016370    ...    -0.954851
2012-12-23  0.536371  1.312673 -0.012154  2.038924    ...     0.145933
2012-12-24 -0.551155 -0.662937  2.888111 -0.644336    ...    -1.326808
2012-12-25 -0.685219  0.911203  0.383256  1.241160    ...    -2.199024
2012-12-26 -0.685735 -0.701683 -0.407999  1.273379    ...    -1.065759
2012-12-27  0.493089  0.181727  0.980818  1.743926    ...    -2.133004
2012-12-28 -0.115295 -0.174491 -0.441849  2.378285    ...    -2.199526
2012-12-29  1.378728  0.589348 -0.174655 -0.234885    ...    -0.723788
2012-12-30 -0.507993  0.788753  1.382333  0.831482    ...     0.049195
2012-12-31  0.155016  0.117294  2.308014 -0.256135    ...    -0.485799

             sig.0042  sig.0043  sig.0044  sig.0045  sig.0046  sig.0047  \
t
2012-12-22 -0.400922  0.693612  0.400481 -0.022681 -0.625497 -0.086370
2012-12-23 -0.712463  0.071106  0.289297 -0.840983 -1.226397  0.044867
2012-12-24  1.559445  0.998510  0.326680 -0.385958 -0.711935  1.576858
2012-12-25  1.842882 -0.409214  0.785073  0.022055  0.487854 -1.362406
2012-12-26  1.648119 -1.426602  1.528350  1.022540  1.246178  1.343820
2012-12-27  0.667818 -1.236415 -0.010889  0.943599 -0.852539  0.814588
2012-12-28  2.415972  0.203597 -1.465696 -0.286231  1.274926 -0.551306
```

```
2012-12-29  0.399894  0.830297  2.322630  0.258503  1.481236 -0.511884
2012-12-30  1.182034 -0.805117  1.693488 -1.356019  1.015045  0.459394
2012-12-31 -0.129288 -0.327848 -1.526784  0.406189  0.248475  0.810656

             sig.0048  sig.0049  sig.0050
t
2012-12-22  1.268790  0.199710 -0.020765
2012-12-23 -0.164480 -1.107409  0.124367
2012-12-24  0.739895 -0.224147 -0.054940
2012-12-25  0.662929  0.257968 -0.663529
2012-12-26 -2.105422 -1.836638 -0.965719
2012-12-27 -0.256562 -0.022903 -0.661562
2012-12-28 -0.116863 -1.754780 -0.393737
2012-12-29 -0.156466 -0.120989 -0.064568
2012-12-30  0.359686 -1.945520  0.638668
2012-12-31 -2.052243  1.023888 -0.822652

[10 rows x 50 columns]
```

In [15]: final_result

Out[15]:
```
           0         1         2         3         4         5         6  \
0  -0.245654 -0.730917  0.479207  0.395333 -0.620082  0.046308  0.373379
1  -0.475143 -0.811006  0.391169  0.597390 -0.697514  0.046408  0.552335
2  -0.488926 -0.809988  0.199982  0.584877 -0.594900 -0.048769  0.549769
3  -0.202874 -0.609680  0.319473  0.318951 -0.503250  0.082815  0.342558
4   0.003304 -0.133988  0.266808  0.040919 -0.437280  0.318032  0.007815
5   0.378611  0.237581  0.394242 -0.235222 -0.355169  0.387157 -0.361138
6   0.610362  0.367212  0.265672 -0.358866 -0.165680  0.436119 -0.538835
7   0.543598  0.520735  0.190036 -0.292132 -0.106749  0.380651 -0.559254
8   0.328916  0.423090  0.032114 -0.088560  0.037259  0.236849 -0.372550
9  -0.017543  0.126543 -0.104766  0.052054  0.195654 -0.028152 -0.078048
10 -0.300911 -0.157168 -0.305924  0.295429  0.308647 -0.368879  0.323424
11 -0.648284 -0.265960 -0.484238  0.287854  0.463486 -0.395288  0.454703
12 -0.691105 -0.183536 -0.483660  0.252034  0.585352 -0.385371  0.507971
13 -0.359166  0.084776 -0.423446  0.126608  0.595519 -0.314807  0.272243
14 -0.150678  0.398026 -0.420981 -0.011607  0.624191 -0.175740 -0.064380
15  0.217755  0.768324 -0.363388 -0.301153  0.663388  0.047077 -0.464927
16  0.347000  0.913318 -0.377937 -0.401045  0.728310  0.090174 -0.699121
17  0.374971  1.047817 -0.422293 -0.357760  0.756084  0.057127 -0.697000
18  0.193090  0.773245 -0.579785 -0.262713  0.813341 -0.017614 -0.416305
19 -0.049477  0.493815 -0.615953 -0.060593  0.800113 -0.185164 -0.112485
20 -0.459704  0.139576 -0.672666  0.134990  0.867058 -0.349391  0.222118
21 -0.616607 -0.115998 -0.613550  0.359999  0.855758 -0.422532  0.424367
22 -0.601139 -0.080107 -0.678179  0.348008  0.726694 -0.512411  0.481229
23 -0.458923  0.035772 -0.535737  0.239061  0.670700 -0.402926  0.295307
24 -0.196379  0.319891 -0.425043  0.015836  0.563558 -0.103048 -0.016357
25  0.121700  0.525596 -0.227558 -0.198402  0.477851  0.062352 -0.330943
```

26	0.422707	0.742086	-0.195680	-0.438402	0.316015	0.151159	-0.579424
27	0.481797	0.710066	-0.077166	-0.411954	0.209772	0.274777	-0.596288
28	0.221284	0.468081	0.021044	-0.278320	0.108934	0.125979	-0.349457
29	-0.031071	-0.027164	-0.061272	0.051372	0.088802	0.097733	-0.014427

	7	8	9	...	40	41	42 \
0	0.316231	0.256049	0.268779	...	-0.163262	-0.120909	0.316724
1	0.250980	0.220736	0.184015	...	0.052575	-0.219684	0.322623
2	0.257667	0.112292	0.193388	...	0.034166	-0.205275	0.328963
3	0.079581	0.121606	0.160786	...	0.016161	-0.103137	0.300082
4	0.017936	0.074963	0.139132	...	-0.188610	0.032440	0.095340
5	-0.014948	0.113463	0.271760	...	-0.276076	0.256338	0.001812
6	-0.017757	0.117591	0.276321	...	-0.355050	0.417835	-0.158534
7	-0.037435	0.096305	0.211513	...	-0.279150	0.250903	-0.164123
8	0.013533	0.049626	0.032603	...	-0.153357	0.168695	-0.085535
9	0.002818	-0.042482	-0.124891	...	0.028333	-0.044852	0.049534
10	0.135308	-0.189210	-0.340713	...	0.387446	-0.213592	0.041173
11	0.091605	-0.193460	-0.356866	...	0.497093	-0.327462	0.079235
12	0.111601	-0.303947	-0.465960	...	0.636132	-0.333622	0.140699
13	0.026976	-0.253732	-0.415321	...	0.638786	-0.329749	-0.046614
14	-0.169264	-0.262142	-0.314337	...	0.449464	-0.099797	-0.156153
15	-0.231566	-0.200355	-0.295249	...	0.309957	0.086893	-0.280168
16	-0.163630	-0.229183	-0.280748	...	0.149140	0.178979	-0.360507
17	-0.259544	-0.147407	-0.302942	...	0.033467	0.308183	-0.306176
18	-0.110269	-0.208068	-0.411016	...	0.284387	0.182918	-0.311818
19	-0.049521	-0.300839	-0.507091	...	0.469881	-0.029744	-0.250576
20	0.067754	-0.387711	-0.577022	...	0.678272	-0.223921	-0.137845
21	0.192681	-0.418007	-0.630005	...	0.723911	-0.326752	-0.033713
22	-0.001426	-0.391912	-0.514526	...	0.728686	-0.405087	0.019009
23	-0.097278	-0.408298	-0.527794	...	0.670251	-0.176259	-0.033792
24	-0.160756	-0.289999	-0.312690	...	0.370307	-0.022129	-0.175758
25	-0.229262	-0.309573	-0.103120	...	0.036476	0.094966	-0.332735
26	-0.188909	-0.242886	-0.104042	...	-0.023267	0.284642	-0.275851
27	-0.178680	-0.002473	-0.103771	...	-0.131780	0.256384	-0.286247
28	-0.087897	-0.031272	-0.110439	...	-0.017700	0.214330	-0.210419
29	-0.043335	-0.001644	-0.210899	...	0.011456	0.113619	-0.148456

	43	44	45	46	47	48	49
0	-0.099420	-0.444744	-0.028906	0.329631	-0.197850	-0.390004	-0.383680
1	-0.266402	-0.675503	-0.189439	0.155201	-0.285163	-0.325432	-0.381305
2	-0.273109	-0.628249	-0.194233	0.233858	-0.294844	-0.316213	-0.375295
3	-0.139048	-0.390808	-0.146545	0.328970	-0.345261	-0.243213	-0.258165
4	-0.007882	-0.142552	0.246506	0.092896	-0.026465	-0.342257	-0.181443
5	0.341731	0.074024	0.482899	0.052194	0.136362	-0.180430	-0.073167
6	0.485427	0.368938	0.564538	-0.012571	0.237135	-0.225453	0.080694
7	0.542183	0.520064	0.489026	-0.063306	0.339405	-0.167181	0.138701
8	0.288632	0.346031	0.391385	0.041751	0.175212	0.022202	0.171442
9	0.070427	0.065267	-0.050542	-0.072220	-0.021982	0.117828	0.109732

10	-0.125748	-0.228775	-0.339318	-0.184763	-0.143388	0.163580	0.118564
11	-0.385762	-0.367223	-0.558832	-0.111914	-0.275713	0.329632	0.087747
12	-0.395130	-0.431975	-0.637604	-0.169453	-0.333786	0.431194	0.095393
13	-0.202598	-0.206983	-0.318014	-0.248853	-0.167142	0.501788	0.182621
14	-0.126321	0.051979	-0.180989	-0.301468	-0.019698	0.464813	0.133104
15	0.210192	0.417804	0.036599	-0.371629	0.211760	0.396645	0.366420
16	0.352063	0.649503	0.241780	-0.296489	0.306072	0.355137	0.401721
17	0.255271	0.786595	0.188532	-0.354835	0.263436	0.457131	0.402078
18	0.273687	0.586975	0.020442	-0.405403	0.188898	0.491910	0.423332
19	-0.007730	0.267103	-0.326785	-0.392258	0.045121	0.563519	0.316951
20	-0.356503	0.006562	-0.613605	-0.395713	-0.130273	0.682145	0.252043
21	-0.504819	-0.226352	-0.777862	-0.398330	-0.204398	0.688623	0.236444
22	-0.470788	-0.236396	-0.722095	-0.289335	-0.335567	0.695206	0.214347
23	-0.270826	-0.097225	-0.569869	-0.254596	-0.196278	0.520618	0.262283
24	-0.005133	0.089497	-0.110152	-0.442757	-0.011952	0.444057	0.237505
25	0.166965	0.430299	0.164086	-0.237259	0.222108	0.254708	0.321199
26	0.419302	0.559396	0.301253	-0.249732	0.290374	0.149171	0.150949
27	0.375649	0.543353	0.311836	-0.213249	0.355702	0.076567	0.176764
28	0.211726	0.432671	0.133082	-0.215844	0.157064	0.160411	0.086436
29	-0.017378	0.104594	-0.106662	-0.119956	-0.029579	0.086140	-0.025381

[30 rows x 50 columns]