Assignment from Video Lecture (TSA)

Table Of Contents (TOC)

TS Stationarity- Code Explanation

jupyter TimeSeries-Stationarity Last Checkpoint: a few seconds ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run | Code

In [8]:

```python

# Augmented Dickey-Fuller (ADF) Test - To determine the stationarity of a TS
# Function to print out results in customised manner

# Importing adfuller function from the module named statsmodels.tsa.stattools
from statsmodels.tsa.stattools import adfuller


# Defining the function named adf_test with 3 parameters
def adf_test(timeseries,df,pollutant):


    # Setting up Plotting Environment
    plt.figure(figsize=(16,5))


    from statsmodels.tsa.stattools import adfuller
    print ('Results of Dickey-Fuller Test:')


    # Performimg ADF test on timeseries data with AIC (Akaike Information Criterion)
    # used for automatic lag selection
    dftest = adfuller(timeseries, autolag='AIC')


    # Creating pandas series to organize & display the ADF test results
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Use


    # For looping for extracting & displaying critical values from the ADF test
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)
    ans=dfoutput
    print("Condition:")
    print("p-value<=0.05-->Accept Alternate Hypothesis")
    print("p-value>0.05-->Accept Null Hypothesis")


    # Creating conditions to check for stationarity & non-stationarity & printing the o/p
    if(ans['Test Statistic']<ans["Critical Value (1%)"] or ans['Test Statistic']<ans["Critical Value (5%)"] or a
        print("Condition: statictic < any critical value and p-value <0.05 to reject null hypothsis")
        print("Reject null hypothesis:Non Stationarity")
        print("Accept Alternate hypothesis:Staionarity ")
        message="Stationarity based on ADH"
    else:
        print("Condition: statictic < any critical value and p-value <0.05 to reject null hypothsis")
        print("Accept null hypothesis:Non Stationarity" )
        print("Reject Alternate hypothesis:Staionarity ")
        message="Non-stationarity based on ADH"



    # Plotting time series data using the matplotlib library customized with title legend
    # & saving the plot as PNG file
    plt.plot(df.index, df[pollutant], label = pollutant)
    plt.legend(loc='best')
    plt.title("{}_{}_2013 to 2021".format(message,pollutant))
    plt.savefig("{}_ADH.png".format(pollutant))
    plt.show()



    # Returns the message whether the  TS is stationarity or not based on ADF test
    return message
```

Approximate Entropy (ApEn)

Approximate Entropy (ApEn) is a metric used to quantify the amount of regularity or predictability within a time series. It was introduced by Pincus in 1991 as a way to assess the complexity of physiological time series data. The ApEn algorithm is particularly useful in the analysis of data where the underlying dynamics may be complex and unpredictable.

Here's a more detailed explanation of how Approximate Entropy works:

1. **Sequence Generation:**
    - Given a time series data set $U=\{u_1, u_2,\ldots,u_N\}$, the first step is to create overlapping sequences of a specified length m.
    - For each index i from 1 to $N-m+1$, a subsequence x_i is formed as $\{u_i,u_{i+1},\ldots,u_{i+m-1}\}$.
2. **Distance Calculation:**
    - Define a distance metric between two sequences x_i and x_j. In the code, the **_maxdist** function calculates the maximum absolute difference between corresponding elements of two sequences.
    - The distance between x_i and x_j is considered significant if it is less than or equal to a specified threshold r.
3. **Similarity Count:**
    - For each subsequence x_i, count the number of other subsequences x_j that are similar to x_i within the threshold r.
    - The variable C is a list storing these counts.
4. **Probability Calculation:**
    - Calculate the probability that two similar sequences x_i and x_j remain similar in the next incremental comparisons.
    - This is done by dividing the count of similar sequences (C) by the total number of sequences $(N-m+1)$.
5. **Entropy Calculation:**
    - For each subsequence x_i, compute the natural logarithm of the probability and take the average over all subsequences.
    - This is done in the **_phi** function.
6. **Final ApEn Calculation:**
    - Compute the ApEn value as the absolute difference between the average logarithmic probabilities for subsequences of lengths m+1 and m.
    - The larger the ApEn, the less predictable or more complex the time series is considered to be.

In summary, ApEn provides a measure of irregularity or complexity in a time series by examining the likelihood that similar patterns persist as the length of the patterns increases. A higher ApEn suggests greater complexity or unpredictability in the time series data. It has applications in various fields, including the analysis of physiological signals, financial time series, and other systems with dynamic and complex behaviour.

Apen - Approximate Entropy Code Explanation

Apen - Approximate Entropy Code Explanation

```
Code: def ApEn(U, m, r):
    """Compute Aproximate entropy"""

Explanation: The function ApEN takes 3 parameters namely U(TS data), m(length of comparable sequence) & r
(threshold which defines how similar the 2 sequences must be to be considered as a match )
```

```
Code: def _maxdist(x_i, x_j):
        return max([abs(ua - va) for ua, va in zip(x_i, x_j)])

Explanation: _maxdist is a helper function which is used to calculate the maximum absolute difference between
the elements of 2 subsequences x_i & x_j.
```

```
Code: def _phi(m):
        x = [[U[j] for j in range(i, i + m - 1 + 1)] for i in range(N - m + 1)]
        C = [len([1 for x_j in x if _maxdist(x_i, x_j) <= r]) / (N - m + 1.0) for x_i in x]
        return (N - m + 1.0)**(-1) * sum(np.log(C))


Explanation: _phi is a helper function which is used to create a overlapping sub-sequence of length m for the
given input series data (U). For each subsequence it counts the number similar sequence within the distance r.
The result calculates the average logrithamic of these counts.
```

```
Code: N = len(U)
    return abs(_phi(m+1) - _phi(m))

Explanation: The main function calculates the ApEn value by setting N as the length of the input series U &
then returning the absolute differnce between the phi values for m+1 & m.
```

```
Code Summary : The code calculates the ApEn value of the given time series data by measuring regularity or
forecastability of the TS data. This is achieved by comparing the overlapping subsequences of different lengths
& counting the number of similar sequences within a given threshold distance. The result is the measure of
system's complexity or irregularity.
```

Sample Entropy (SampEn)

Sample Entropy is another complexity measure used in time series analysis, similar to Approximate Entropy (ApEn). Sample Entropy is designed to overcome some limitations of ApEn, particularly its sensitivity to the length of the time series. Here's a step-by-step explanation of how Sample Entropy works:

1. **Sequence Generation:**
    - Given a time series data set $U=\{u_1, u_2,\ldots,u_N\}$, the first step is to create non overlapping sequences of a specified length m.
    - For each index i from 1 to $N-m+1$, a subsequence x_i is formed as $\{u_i,u_{i+1},\ldots,u_{i+m-1}\}$.
2. **Distance Calculation:**
    - Define a distance metric between two sequences x_i and x_j. In the context of Sample Entropy, the distance is the maximum absolute difference between corresponding elements of two sequences.
3. **Pattern Matching:**
    - Count the number of similar sequences x_j to a reference sequence x_i within a specified threshold r.
    - This is similar to the process in ApEn, but Sample Entropy considers non-overlapping sequences.
4. **Self-Matching Exclusion:**
    - Exclude the self-matching cases (when i=j) from the count.
5. **Probability Calculation:**
    - Calculate the probability that two sequences are similar within the threshold r for a given length m.
    - Divide the count of similar sequences by the total number of non-self-matching pairs.
6. **Entropy Calculation:**
    - Compute the natural logarithm of the probability for each length m.
    - Take the average of these logarithmic probabilities.
7. **Final Sample Entropy Calculation:**
    - Sample Entropy (SampEn) is defined as the negative natural logarithm of the average probability: SampEn(U, m, r)=−ln(count(m)/count(m+1))

- This formula measures the likelihood that patterns of length m remain similar when the length is increased to m+1. A lower Sample Entropy value indicates a more regular or predictable time series.

In summary, Sample Entropy quantifies the regularity or predictability of a time series by comparing non-overlapping subsequences. It provides a complexity measure that is less sensitive to the length of the time series compared to ApEn. Lower Sample Entropy values suggest a more regular or ordered time series, while higher values indicate greater complexity or irregularity.

SampEN- Sample Entropy Code Explanation

spen - Sample Entropy Code Explanation

```
Code: def SampEn(U, m, r):
    """Compute Sample entropy"""

Explanation: The function SampEN takes 3 parameters namely U(TS data), m(length of comparable sequence) & r
(threshold which defines how similar the 2 sequences must be to be considered as a match )
```

```
Code: def _maxdist(x_i, x_j):
        return max([abs(ua - va) for ua, va in zip(x_i, x_j)])

Explanation: _maxdist is a helper function which is used to calculate the maximum absolute difference between
the elements of 2 subsequences x_i & x_j.
```

```
Code: def _phi(m):
        x = [[U[j] for j in range(i, i + m - 1 + 1)] for i in range(N - m + 1)]
        C = [len([1 for j in range(len(x)) if i != j and _maxdist(x[i], x[j]) <= r]) for i in range(len(x))]
        return sum(C)

Explanation: _phi is a helper function which is used to create a non-overlapping sub-sequence of length m for
the given input series data (U). For each subsequence it counts the number similar sequence within the distance
r, excluding self matching cases. The result calculates the sum of these counts.
```

```
Code: N = len(U)
    return -np.log(_phi(m+1) / _phi(m))

Explanation: The main function calculates the Sample Entropy value by setting N as the length of the input
series U & it is computed using the formula -log(_phi(m+1) / _phi(m)). This formula measures the likelihood
that a pattern of length m to be in similar when the length is increased to m+1. The negative natural log is
taken to obtain a positive value. Lower the Sample entropy value higher the regularity or forecastability of
the TS data.
```

```
Code Summary : The code calculates the SampEn value of the given time series data by measuring regularity or
forecastability of the TS data. This is achieved by comparing the non- overlapping subsequences of specified
length & counting the number of similar sequences within a given threshold distance & applying logrithamic
transformation to measure the likelihood of pattern similarity accross different lengths.
```

TS- Model- VAR Function Code Execution

Assignment : Line by Line Code Execution of Below Function

```
def cominbation(dataset,listt):
    print(data1)
    datasetTwo=dataset[listt]
    print(datasetTwo)
    test_obs = 28
    train =datasetTwo[:-test_obs]
    print(train)
    test = datasetTwo[-test_obs:]
    print(test)

    from statsmodels.tsa.api import VAR
    for i in [1,2,3,4,5,6,7,8,9,10]:
        model = VAR(train)
        print(model)
        results = model.fit(i)
        print(results)
        print('Order =', i)
        print('AIC: ', results.aic)
        print('BIC: ', results.bic)
        print()
    x = model.select_order(maxlags=12)
    print(X)
    order=x.selected_orders["aic"]
    print(order)
    result = model.fit(order)
    print(resul)

    #result.summary()
    lagged_Values = train.values[-order:]
    print(lagged_Values)
    pred = result.forecast(y=lagged_Values,steps=28)
    print(pred)
    preds=pd.DataFrame(pred,columns=listt)
    print(preds)
    preds.to_csv("varforecasted_{}.csv".format(test_obs))


    from sklearn.metrics import mean_squared_error
    rmse= round(mean_squared_error(test,pred,squared=False))
    print(rmse)
    from sklearn.metrics import mean_absolute_percentage_error
    mape=mean_absolute_percentage_error(test,pred)
    print(mape)


    performance["Model"].append(listt)
    performance["RMSE"].append(rmse)
    performance["MaPe"].append(mape)
    performance["Lag"].append(order)
    performance["Test"].append(test_obs)


    perf=pd.DataFrame(performance)
```

In [14]:
```
print(data1)
print(listt)
```

```
          Open      High       Low     Close
0     0.517223  0.485749  0.529877  0.522210
1     0.500522  0.491400  0.530969  0.519394
2     0.498434  0.472236  0.528786  0.517972
3     0.506785  0.472236  0.532251  0.520225
4     0.507411  0.479533  0.527149  0.513788
...        ...       ...       ...       ...
2139  0.877349  0.819410  0.900409  0.880633
2140  0.856733  0.807862  0.885130  0.870440
2141  0.848643  0.799017  0.873670  0.865612
2142  0.864301  0.810565  0.892224  0.873927
2143  0.864301  0.821130  0.894952  0.885998

[2144 rows x 4 columns]
['Close', 'High', 'Open', 'Low']
```

In [15]:
```
datasetTwo=data1[listt]
datasetTwo
```

Out[15]:

	Close	High	Open	Low
0	0.522210	0.485749	0.517223	0.529877
1	0.519394	0.491400	0.500522	0.530969
2	0.517972	0.472236	0.498434	0.528786
3	0.520225	0.472236	0.506785	0.532251
4	0.513788	0.479533	0.507411	0.527149
...	...	...	...	...
2139	0.880633	0.819410	0.877349	0.900409
2140	0.870440	0.807862	0.856733	0.885130
2141	0.865612	0.799017	0.848643	0.873670
2142	0.873927	0.810565	0.864301	0.892224
2143	0.885998	0.821130	0.864301	0.894952

2144 rows × 4 columns

In [16]:
```python
test_obs = 28
train =datasetTwo[:-test_obs]
train
```

Out[16]:

	Close	High	Open	Low
0	0.522210	0.485749	0.517223	0.529877
1	0.519394	0.491400	0.500522	0.530969
2	0.517972	0.472236	0.498434	0.528786
3	0.520225	0.472236	0.506785	0.532251
4	0.513788	0.479533	0.507411	0.527149
...	...	...	...	...
2111	0.906384	0.840786	0.786013	0.807640
2112	0.874195	0.833170	0.888570	0.892497
2113	0.868830	0.814742	0.853862	0.884038
2114	0.853541	0.789189	0.845772	0.872578
2115	0.878487	0.812285	0.832463	0.875307

2116 rows × 4 columns

In [17]:
```python
test = datasetTwo[-test_obs:]
test
```

Out[17]:

	Close	High	Open	Low
2116	0.883315	0.815725	0.858820	0.896589
2117	0.903433	0.834889	0.863779	0.897408
2118	0.887607	0.824570	0.882568	0.907503
2119	0.882779	0.808354	0.848382	0.889768
2120	0.878755	0.814496	0.863779	0.897681
2121	0.891899	0.819902	0.863779	0.905593
2122	0.904238	0.839803	0.874739	0.915962
2123	0.906652	0.844472	0.889353	0.925784
2124	0.907994	0.835381	0.886482	0.923056
2125	0.916577	0.849140	0.878914	0.917053
2126	0.923015	0.858968	0.899791	0.931514
2127	0.903970	0.851597	0.902923	0.921692
2128	0.903970	0.826044	0.879958	0.906958
2129	0.913627	0.852580	0.887787	0.929877
2130	0.912822	0.852580	0.878392	0.920600
2131	0.850858	0.830958	0.879958	0.857844
2132	0.871781	0.800000	0.825157	0.829468
2133	0.878487	0.808845	0.837944	0.873943
2134	0.874463	0.858968	0.863518	0.884038
2135	0.895118	0.825061	0.866388	0.906958
2136	0.887875	0.823096	0.877610	0.897954
2137	0.917918	0.839066	0.869259	0.915962
2138	0.895386	0.835872	0.893006	0.912688
2139	0.880633	0.819410	0.877349	0.900409
2140	0.870440	0.807862	0.856733	0.885130
2141	0.865612	0.799017	0.848643	0.873670
2142	0.873927	0.810565	0.864301	0.892224
2143	0.885998	0.821130	0.864301	0.894952

In [18]:

```python
from statsmodels.tsa.api import VAR
for i in [1,2,3,4,5,6,7,8,9,10]:
    model = VAR(train)
    print(model)
    results = model.fit(1)
    print(results)
    print('Order =', 1)
    print('AIC: ', results.aic)
    print('BIC: ', results.bic)
    print()
```

```
<statsmodels.tsa.vector_ar.var_model.VAR object at 0x12ea9bb90>
<statsmodels.tsa.vector_ar.var_model.VARResultsWrapper object at 0x12f4352d0>
Order = 1
AIC:  -39.01658695115506
BIC:  -38.96309465714579

<statsmodels.tsa.vector_ar.var_model.VAR object at 0x12e126750>
<statsmodels.tsa.vector_ar.var_model.VARResultsWrapper object at 0x12f436890>
Order = 1
AIC:  -39.01658695115506
BIC:  -38.96309465714579

<statsmodels.tsa.vector_ar.var_model.VAR object at 0x12ee94650>
<statsmodels.tsa.vector_ar.var_model.VARResultsWrapper object at 0x12f435e10>
Order = 1
AIC:  -39.01658695115506
BIC:  -38.96309465714579

<statsmodels.tsa.vector_ar.var_model.VAR object at 0x12e126750>
<statsmodels.tsa.vector_ar.var_model.VARResultsWrapper object at 0x12f436dd0>
Order = 1
AIC:  -39.01658695115506
BIC:  -38.96309465714579

<statsmodels.tsa.vector_ar.var_model.VAR object at 0x12f403a90>
<statsmodels.tsa.vector_ar.var_model.VARResultsWrapper object at 0x12f437f90>
Order = 1
AIC:  -39.01658695115506
BIC:  -38.96309465714579

<statsmodels.tsa.vector_ar.var_model.VAR object at 0x12e126750>
<statsmodels.tsa.vector_ar.var_model.VARResultsWrapper object at 0x12f437fd0>
Order = 1
AIC:  -39.01658695115506
BIC:  -38.96309465714579

<statsmodels.tsa.vector_ar.var_model.VAR object at 0x12f403a90>
<statsmodels.tsa.vector_ar.var_model.VARResultsWrapper object at 0x12f436250>
Order = 1
AIC:  -39.01658695115506
BIC:  -38.96309465714579

<statsmodels.tsa.vector_ar.var_model.VAR object at 0x12e126750>
<statsmodels.tsa.vector_ar.var_model.VARResultsWrapper object at 0x12f436290>
Order = 1
AIC:  -39.01658695115506
BIC:  -38.96309465714579

<statsmodels.tsa.vector_ar.var_model.VAR object at 0x12f403a90>
<statsmodels.tsa.vector_ar.var_model.VARResultsWrapper object at 0x12f436690>
Order = 1
AIC:  -39.01658695115506
BIC:  -38.96309465714579

<statsmodels.tsa.vector_ar.var_model.VAR object at 0x12f422610>
<statsmodels.tsa.vector_ar.var_model.VARResultsWrapper object at 0x12f44c150>
Order = 1
AIC:  -39.01658695115506
BIC:  -38.96309465714579
```

In [19]:

```python
x = model.select_order(maxlags=12)
x
```

Out[19]:

```
<statsmodels.tsa.vector_ar.var_model.LagOrderResults at 0x12f437690>
```

In [20]:

```python
order=x.selected_orders["aic"]
order
```

Out[20]:

```
10
```

In [21]:

```python
result = model.fit(order)
result
```

Out[21]:

```
<statsmodels.tsa.vector_ar.var_model.VARResultsWrapper at 0x12f435d90>
```

In [22]:
```python
lagged_Values = train.values[-order:]
lagged_Values
```

Out[22]:
```
array([[0.77870172, 0.73267815, 0.78601252, 0.77653481],
       [0.80606218, 0.75257989, 0.7651357 , 0.80763984],
       [0.83476393, 0.77100741, 0.80114819, 0.83683496],
       [0.82376606, 0.76633912, 0.81732776, 0.84065486],
       [0.81330471, 0.75184274, 0.8063674 , 0.82592091],
       [0.90638414, 0.84078626, 0.78601252, 0.80763984],
       [0.8741953 , 0.83316954, 0.8885699 , 0.89249661],
       [0.86883049, 0.81474202, 0.85386221, 0.8840382 ],
       [0.85354076, 0.78918921, 0.84577243, 0.87257841],
       [0.87848708, 0.81228502, 0.83246343, 0.87530692]])
```

In [23]:
```python
pred = result.forecast(y=lagged_Values,steps=28)
pred
```

Out[23]:
```
array([[0.87807691, 0.82598206, 0.86232426, 0.8845791 ],
       [0.87568858, 0.81755924, 0.858365  , 0.87981049],
       [0.87931792, 0.8170355 , 0.85661658, 0.88266697],
       [0.87537281, 0.81807134, 0.85883094, 0.88569871],
       [0.87012273, 0.815102  , 0.85419909, 0.87398718],
       [0.87623064, 0.8164118 , 0.85774628, 0.87913361],
       [0.87594138, 0.81723438, 0.8564004 , 0.87980614],
       [0.87737979, 0.81786679, 0.85566662, 0.88170201],
       [0.87502015, 0.81770654, 0.85700998, 0.88122196],
       [0.87435608, 0.81651541, 0.8559047 , 0.8793504 ],
       [0.87379514, 0.81499682, 0.85491574, 0.87861856],
       [0.87443972, 0.81548939, 0.8542209 , 0.87883689],
       [0.87517773, 0.8158027 , 0.85486808, 0.88008921],
       [0.87506145, 0.81623921, 0.85524984, 0.87990944],
       [0.87444131, 0.81546427, 0.85550376, 0.87921127],
       [0.8744636 , 0.81564719, 0.85491286, 0.87892095],
       [0.87449472, 0.81571873, 0.85496938, 0.87919274],
       [0.87451894, 0.81574177, 0.85499225, 0.879387  ],
       [0.87434412, 0.81552834, 0.85488921, 0.87917138],
       [0.87431191, 0.81544117, 0.85481425, 0.8790541 ],
       [0.87422643, 0.81546109, 0.85478552, 0.87899345],
       [0.87425804, 0.81540053, 0.85473186, 0.87908656],
       [0.87430662, 0.81540676, 0.85470382, 0.87912503],
       [0.8744096 , 0.8154542 , 0.85475926, 0.87917185],
       [0.8744239 , 0.81554322, 0.8548707 , 0.87916934],
       [0.87443778, 0.81557926, 0.85490572, 0.87920031],
       [0.87444798, 0.81557689, 0.85490471, 0.87923497],
       [0.87446717, 0.81558836, 0.8549039 , 0.87926089]])
```

In [24]:
```python
preds=pd.DataFrame(pred,columns=listt)
preds
```

Out[24]:

	Close	High	Open	Low
0	0.878077	0.825982	0.862324	0.884579
1	0.875689	0.817559	0.858365	0.879810
2	0.879318	0.817036	0.856617	0.882667
3	0.875373	0.818071	0.858831	0.885699
4	0.870123	0.815102	0.854199	0.873987
5	0.876231	0.816412	0.857746	0.879134
6	0.875941	0.817234	0.856400	0.879806
7	0.877380	0.817867	0.855667	0.881702
8	0.875020	0.817707	0.857010	0.881222
9	0.874356	0.816515	0.855905	0.879350
10	0.873795	0.814997	0.854916	0.878619
11	0.874440	0.815489	0.854221	0.878837
12	0.875178	0.815803	0.854868	0.880089
13	0.875061	0.816239	0.855250	0.879909
14	0.874441	0.815464	0.855504	0.879211
15	0.874464	0.815647	0.854913	0.878921
16	0.874495	0.815719	0.854969	0.879193
17	0.874519	0.815742	0.854992	0.879387
18	0.874344	0.815528	0.854889	0.879171
19	0.874312	0.815441	0.854814	0.879054
20	0.874226	0.815461	0.854786	0.878993
21	0.874258	0.815401	0.854732	0.879087
22	0.874307	0.815407	0.854704	0.879125
23	0.874410	0.815454	0.854759	0.879172
24	0.874424	0.815543	0.854871	0.879169
25	0.874438	0.815579	0.854906	0.879200
26	0.874448	0.815577	0.854905	0.879235
27	0.874467	0.815588	0.854904	0.879261

```
In [25]: preds.to_csv("varforecasted_{}.csv".format(test_obs))

In [26]: from sklearn.metrics import mean_squared_error
         rmse= round(mean_squared_error(test,pred,squared=False))
         rmse

Out[26]: 0

In [27]: from sklearn.metrics import mean_absolute_percentage_error
         mape=mean_absolute_percentage_error(test,pred)
         mape

Out[27]: 0.023686276741415255

In [28]: performance["Model"].append(listt)
         performance["RMSE"].append(rmse)
         performance["MaPe"].append(mape)
         performance["Lag"].append(order)
         performance["Test"].append(test_obs)
         print(performance)

         {'Model': [['Close', 'High', 'Open', 'Low'], ['Close', 'High', 'Open', 'Low']], 'RMSE': [0, 0], 'MaPe': [0.02368627
         6741415255, 0.023686276741415255], 'Lag': [10, 10], 'Test': [28, 28]}

In [29]: perf=pd.DataFrame(performance)
         perf

Out[29]:
```

	Model	RMSE	MaPe	Lag	Test
0	[Close, High, Open, Low]	0	0.023686	10	28
1	[Close, High, Open, Low]	0	0.023686	10	28

__Note__ : IPyNb of this code execution can be found in my Github – Assignments/TSA/TS-Models/VAR.

Ipython Notebooks of other TS Models like VARMA, SES, HWES can also be found in Github - Assignments/TSA/TS-Models

----------END OF THE DOCUMENT---------