VAR TO CAPTURE THE EVOLUTION AND THE INTER-DEPENDENCIES

Vector Autoregressive for Forecasting Time Series

Econometric model involving multiple time series

Sarit Maitra · Following

Published in Towards Data Science

11 min read · Jun 18, 2020

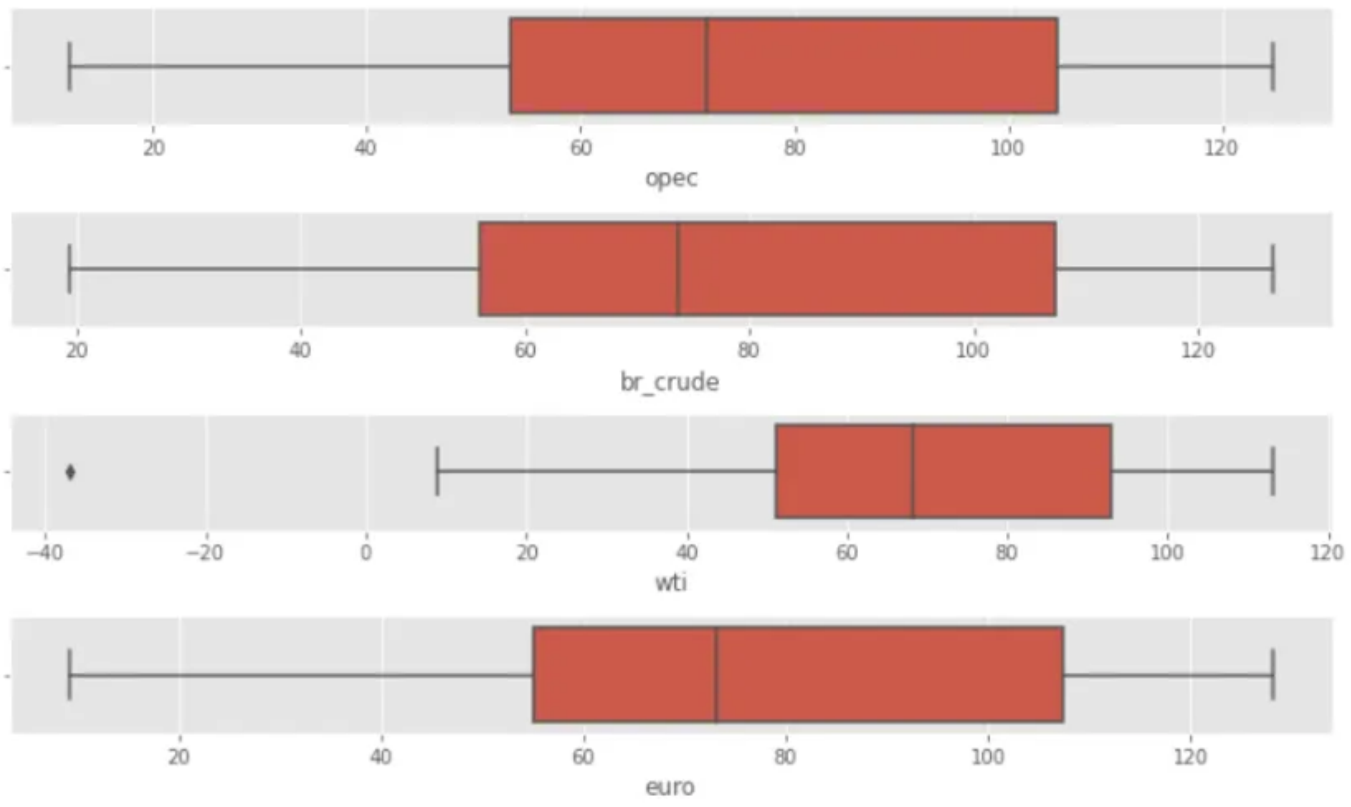

Image by author

https://sarit-maitra.medium.com/membership

Vector auto-regression (VAR) time series model has wide application in econometric forecasting model; VAR can capture the evolution and the inter-dependencies between multiple time-series. All the variables in a VAR are treated symmetrically by including for each variable an equation explaining its evolution based on its own lags and the lags of all the other variables in the model. We may call this as scientific method for trading strategy.

> Advantage of using the scientific method for trading strategy design is that if the strategy fails after a prior period of profitability, it is possible to revisit the initial hypothesis and re-evaluate it, potentially leading to a new hypothesis that leads to regained profitability for a strategy.

We have four time series here for Brent crude; USA & Europe price, West Texas crude price and OPEC crude price. We will develop a regression model and shall try to predict Brent Crude Oil, USA price given all the series in our model.

```
plt.plot(df['opec'], label="opec")
plt.plot(df['euro'], label="Euro")
plt.plot(df['br_crude'], label="Brent Crude Oil")
plt.plot(df['wti'], label="West Texas")

plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.title('Opec Oil, Europe Brent Crude, Brent Crude Oil & West
Texas Price over Time')
plt.show()
```

We can see here, all the series follow a stochastic trend, showing stronger inter-temporal variation with larger drops; more so, all of the series seem to be related in some way. Neither series looks stationary in its levels. They appear to have a common trend, an indication that they may be co-integrated.

IQR plot all variables:

```
plt.title('IQR plot for all variables')
sns.boxplot(x=df)
plt.show()
```

We can see one small outlier in the data, which can be ignored.

VAR model describe the dynamic interrelationship among stationary variables. So, the first step in time-series analysis should be unit root test to determine whether the series are stationary.

Correlation Check:

```
pearsoncorr = df.corr(method='pearson')
sns.heatmap(pearsoncorr,
xticklabels=pearsoncorr.columns,
yticklabels=pearsoncorr.columns,
cmap='RdBu_r',
annot=True,
linewidth=0.5)
plt.show()
```

Though correlations are often used in all multivariate financial time-series use cases, however, correlations could be quite unstable. Moreover, ADF or Hurst Exponent helps us to statistically confirm whether this series is mean-reverting. However, we cannot determine the hedging ratio needed to form the linear combination from these tests, they will only tell us whether, for a particular , the linear combination is stationary. The alternate arrangements is cointegration which is probably a more robust measure of linkage between two financial series.

Augmented Dickey-Fuller (ADF):

ADF test says that, if a price series possesses mean reversion, then the next price level will be proportional to the current price level. Mathematically, the ADF is based on the idea of testing for the presence of a unit root in an autoregressive time series sample.

```
def adfuller_test(series, signif=0.05, name='', verbose=False):
  r = adfuller(series, autolag='AIC')
  output = {'test_statistic':round(r[0], 4), 'pvalue':
round(r[1], 4), 'n_lags':round(r[2], 4), 'n_obs':r[3]}
  p_value = output['pvalue']
def adjust(val, length= 6):
  return str(val).ljust(length)

print(f'Augmented Dickey-Fuller Test on "{name}"', "\n   ", '-'*47)
print(f'Null Hypothesis: Data has unit root. Non-Stationary.')
print(f'Significance Level = {signif}')
print(f'Test Statistic = {output["test_statistic"]}')
print(f'No. Lags Chosen = {output["n_lags"]}')

for key,val in r[4].items():
  print(f' Critical value {adjust(key)} = {round(val, 3)}')

  if p_value <= signif:
    print(f" => P-Value = {p_value}. Rejecting Null Hypothesis.")
    print(f" => Series is Stationary.")
  else:
    print(f" => P-Value = {p_value}. Weak evidence to reject the
Null Hypothesis.")
    print(f" => Series is Non-Stationary.")
```

None of the statistic even close to being significant at the 5% level. So, we can confidently say that the series are non-stationary in level. Their co-integration is explored and in each case (p-values > 0.05), the null hypothesis of non-stationarity cannot be rejected at any reasonable level of significance.

Co-integration tests:

The steps in the cointegration test procedure:

1. Test for a unit root in each component series individually, using the univariate unit root tests, say ADF, PP test.
2. If the unit root cannot be rejected, then the next step is to test cointegration among the components, i.e., to test whether αYt is I(0).

If we find that the time series as a unit root, then we move on to the cointegration process. There are three main methods for testing for cointegration: Johansen, Engle-Granger, and Phillips-Ouliaris. We will use the Engle-Granger test. By employing co-integration test we will examine if there is a cointegrated, long-run relationship between wti price index and br_crude price index. Mathematically, we will consider the cointegrated relationship:

Engle & Granger’s two-step residual-based testing procedure first estimates a cross-sectional regression, and then tests the residuals from this regression using an ADF distribution with modified critical values.

Here a vector of observations is recorded at each time point. In such cases, there may be some linear combinations of the vectors that form stationary time-series and other linear combinations that are non-stationary.

From above model residual plot it appears that mean is 0, it might have a trend in it.

The estimated co-integrating vector is very close to [+1, −1], indicating a simple no-arbitrage relationship. Let us rerun the test with both a constant and a time trend (*"ct"*). Our test statistic of -4.231< critical values at the 5% (-3.79) and 10% (3.50) level; p-values < 0.05 and we can reject the null hypothesis of no co-integration.

The residuals are clearly mean zero but show evidence of a structural break around current year (2020).

Likewise, we have tested the other possible combination of pairs to obtain the similar output.

> Although the Engle-Granger approach is quite easy to use, one of its major drawbacks is that it can estimate only up to one co-integrating relationship between the variables.

Here, we have more than 1 variables, there can potentially be more than one linearly independent co-integrating relationship.

Importance of p-value:

There is an ongoing debate on p-value in real-life business case. p-value may look good on published articles, but it was never meant to be used the way it's used today.

For more details please read this interesting *article*. Traditionally, *p*-value <0.05 is the criterion for significance.

OLS regression:

Now, we will fit a regression between time series using OLS and visualize the residuals plot. If residuals are stationary, we can safely assume that, the given series are really cointegrated.

```
residuals = results.resid
br_crude_fit = results.fittedvalues

fig = plt.figure(1)
ax1 = fig.add_subplot(111)
ax1.plot(br_crude_fit, label='Linear Prediction')
ax1.plot(df['br_crude'], label='Brent Crude-USA')
```

```
  ax1.set_xlabel('Date')
  ax1.legend(loc=0)
  ax2 = plt.twinx()
  ax2.set_ylabel('Residuals')
  ax2.plot(residuals, label='Residuals')
  ax2.legend(loc=0)
  plt.grid(True)
  plt.show()
```

The levels of the residuals (red line), which looks like a stationary series than the original series (blue line). Here, we can see that, the actual and fitted lines are indistinguishable.

Let us now perform an ADF test on the residual series. We have used DFGLS test and specify 12 lags as the maximum lag order for DFGLS regressions.

The number of lags suggests an optimal lag length of 7. Our test statistic (–7.059) < Critical Values at 1%, 5% & 10% and null hypothesis of no cointegration can be rejected.

Breusch-Godfrey (BD)test:

BD is more general than DW and allows us to test for higher order auto-correlation.

The distribution of the residuals looks like a bell-shape although there are some outliers which might lead to skewness.

Jarque-Bera normality test:

χ2 two-tailed p-value for the test equals zero means we accept the null hypothesis of residuals are overall normally distributed, i.e., both the kurtosis and the skewness are those of a normal distribution.

Vector Auto regression (VAR):

VAR method models the next step in each time series using an AR model. The notation for the model involves specifying the order for the AR(p) model as parameters to a VAR function {VAR(p)}.

Let us take the first differences of the series and try ADF again. Usually, if the levels time series are not stationary, the first differences will be. Data differencing is, more or less, calculating the amount the time series changes from one value to the next.

```
df = df [['br_crude', 'wti', 'opec', 'euro']]
nobs = 5
df_train, df_test = df[0:-nobs], df[-nobs:]

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

df_tr_diff = df_train.diff()
df_tr_diff.dropna(inplace=True)
print(df_tr_diff.head())

df_tr_diff.plot(figsize=(10,6), linewidth=5, fontsize=20)
plt.show()
```

Creating a training and test set for time series problems is tricky considering the time component. Any wrong move at this stage will disrupt the pattern in the series.

The test set should be created considering the date and time values.

Now we repeat all of the above steps for the First-Difference of data frame.

```
# ADF Test on each column
for name, column in df_tr_diff.iteritems():
adfuller_test(column, name=column.name)
```

Here, we see all the series are stationary after first difference.

VAR model:

```
var_model = smt.VAR(df_tr_diff)
res = var_model.select_order(maxlags=10)
print(res.summary())
```

```
#Fit VAR model
var_model = VAR(endog=df_tr_diff)
var_model_fit = var_model.fit(maxlags=10)

#Print a summary of the model results
var_model_fit.summary()
```

What we really want to focus on in the model summary is the equation for br_crude, where br_crude estimates the other prices based on lagged values of itself and lagged prices of other variables. Higher the t-statistic value, the more likely that we can reject the H0 and more likely there is a correlation between the two variables. p-values are also used to reject the null hypothesis. The lower the p-value, the stronger the evidence against the null hypothesis.

Estimating the equations of a VAR does not really need strong assumptions; however, calculating impulse response functions (IRFs) or variance decompositions do require identifying restrictions. A typical restriction takes the form of an assumption about the dynamic relationship between a pair of variables, for example, that x affects y only with a lag, or that x does not affect y in the long run

Let us visualize the impact of changes in one variable on the others at different horizons using impulse responses plot.

```
# Impulse Response Analysis
irf = var_model_fit.irf(20)
irf.plot()
plt.show()
```

Given the parameter estimates and the *Engle Granger* test results, the linkages between the series are established here. It can be seen that, the responses to the shocks are captured here, and they die down to almost after 10th lag.

Forecast Error Variance Decomposition (FEVD):

The effect that a shock to the br_crude price has on the other prices and other values of themselves are shown in the first row of the FEVD plot. The % of the errors attributable to own shocks is 100% in the case of the br_crude price (dark black bar), for wti series explains around 40% of the variation in returns; for opec around 60% and for the euro series explains around 70% of the variation.

Note: Concept of above tests procedures have been taken from Brooks (2019); see the reference section.

Prediction:

Now the model is built, let us generate and compare the predictions to actual data in the test/validation set.

```
# Get the lag order
```

```
lag_order = var_model_fit.k_ar
print(lag_order)

# Input data for forecasting
input_data = df_tr_diff.values[-lag_order:]
print(input_data)

# forecasting
pred = var_model_fit.forecast(y=input_data, steps=nobs)
pred = (pd.DataFrame(pred, index=df_test.index, columns= df.columns
+ '_pred'))
print(pred)
```

Invert transformation:

```
def invert_transformation(df_tr_diff, pred):
  forecast = pred.copy()
  columns = df_train.columns

for col in columns:
  forecast[str(col)+'_pred'] = df_train[col].iloc[-1] +
                               forecast[str(col)+'_pred'].cumsum()
  return forecast

output = invert_transformation(df_tr_diff, pred)
output.loc[:, ['br_crude_pred']]
```

Forecast evaluation:

```
# forecast bias
forecast_errors = [combine['br_crude'][i]- combine['br_crude_pred']
[i] for i in range(len(combine['br_crude']))]
bias = sum(forecast_errors) * 1.0/len(combine['br_crude'])
print('Bias: %f' % bias)

# MAE
mae = mean_absolute_error(combine['br_crude'],
combine['br_crude_pred'])
print('MAE: %f' % mae)

# MSE & RMSE
mse = mean_squared_error(combine['br_crude'],
combine['br_crude_pred'])
print('MSE: %f' % mse)
rmse = sqrt(mse)
print('RMSE: %f' % rmse)
```

Actual vs Predicted:

```
combine = pd.concat([df_test['br_crude'], output['br_crude_pred']],
axis=1)
combine['accuracy'] = round(combine.apply(lambda row:
row.br_crude_pred /
row.br_crude *100, axis = 1),2)

combine['accuracy'] = pd.Series(["{0:.2f}%".format(val) for val in
combine['accuracy']],index = combine.index)
combine = combine.assign(day_of_week = lambda x: x.index.day_name())
combine = combine.round(decimals=2)
combine = combine.reset_index()
combine
```

```
fig = go.Figure()
fig.add_trace(go.Scatter(x=combine['Date'], y=combine['br_crude'],
name="Actual Crude price"))
fig.add_trace(go.Scatter(x=combine['Date'],y=combine['br_crude_pred'
],name="Predicted crude price"))
fig.update_layout(
title="Actual vs Predicted Brent Crude Price",
xaxis_title="Date", yaxis_title="Price", font=dict(family="Courier
New, monospace",size=18,color="#7f7f7f"))
fig.update_layout(autosize=False,width=1000,height=400,)
fig.update_layout(legend_orientation="h")
fig.show()
```

Conclusion:

Once we have a model for our data, it's important to analyze how we can evaluate its quality. The first option is to use the residuals. Residuals basically the squared difference between the predicted values and actual values. This could be a simple option to judge the accuracy of our model.

Residuals are proportional to the amount of data that we use to create the model. Therefore, it is not advisable to use the sum of all the residuals. One option could be to divide the residual by the number of time instants multiplied by the number of dependent variables.

I can be reached here.

References:

1. *Hamilton J.D.A., 1994. The time series analysis. New Jersey, Princeton University Press.*
2. *Engle, R.F., Granger, C.W.J, 1987. Co-integration and error correction: Representation, estimation and testing. Econometrica. 55, 251–276.*
3. *Brooks, C. (2019). Introductory econometrics for finance. Cambridge university press.*

Vector Auto Regression Econometrics Analytics Time Series Forecasting

Following

Written by Sarit Maitra

3K Followers · Writer for Towards Data Science

Analytics & Data Science Practice Lead

More from Sarit Maitra and Towards Data Science

Sarit Maitra in Towards Data Science

Time-series Analysis with VAR & VECM: Statistical approach with complete Python code

VECTOR auto-regressive (VAR) integrated model comprises multiple time series and quite useful tool for forecasting. It can be considered...

· 10 min read · Nov 13, 2019

302 6

Adrian H. Raudaschl in Towards Data Science

Forget RAG, the Future is RAG-Fusion

The Next Frontier of Search: Retrieval Augmented Generation meets Reciprocal Rank Fusion and Generated Queries

· 10 min read · Oct 6

2.3K 24

Marco Peixeiro in Towards Data Science

TimeGPT: The First Foundation Model for Time Series Forecasting

Explore the first generative pre-trained forecasting model and apply it in a project with Python

· 12 min read · Oct 24

1.98K 18

Sarit Maitra in Towards Data Science

State Space Model and Kalman Filter for Time-Series Prediction

Time-series forecasting using financial data

· 11 min read · Dec 29, 2019

190 1

See all from Sarit Maitra

See all from Towards Data Science

Recommended from Medium

Sofien Kaabar, CFA

The Secret Sauce of Trading— My Top Technical Indicators Revealed

Presenting my Dream Team of Technical Indicators

· 5 min read · Nov 2

294 3

Bradley Stephen Shaw in Towards AI

Let's Do: Time Series Decomposition

A guide to effectively breaking a time series into its constituent parts

8 min read · Jun 17

42 1

Lists

Practical Guides to Machine Learning
10 stories · 650 saves

Predictive Modeling w/ Python
20 stories · 569 saves

New_Reading_List
174 stories · 183 saves

Productivity
230 stories · 179 saves

Valeriy Manokhin, PhD, MBA, CQF

Avoiding the Forecasting Pitfalls: 10 Red Flags in Hiring Data Scientists

Navigate Through Your Next Data Science Interview Successfully by Adhering to These Key Forecasting Rules

✦ · 25 min read · Nov 2

165 2

Everton Gomede, PhD

Forecasting Non-Stationary Time Series

Introduction

6 min read · Oct 13

192

Vaibhav Rastogi

Decomposition in time series analysis

Decomposition in time series analysis is a method used to separate a time series into three distinct components: trend, seasonality, and...

2 min read · Aug 13

Arun Jagota

Anomaly Detection In Time Series

Basic scenarios and methods

✦ · 6 min read · Oct 6

98

See more recommendations