towards
data science

Follow 521K Followers

PREDICTIVE ANALYTICS AND TIME SERIES DATA

Future Price Prediction Beyond Test Data Using Vector Auto Regression

Simple steps to multi-step future prediction

Sarit Maitra Sep 9, 2020 · 7 min read ★

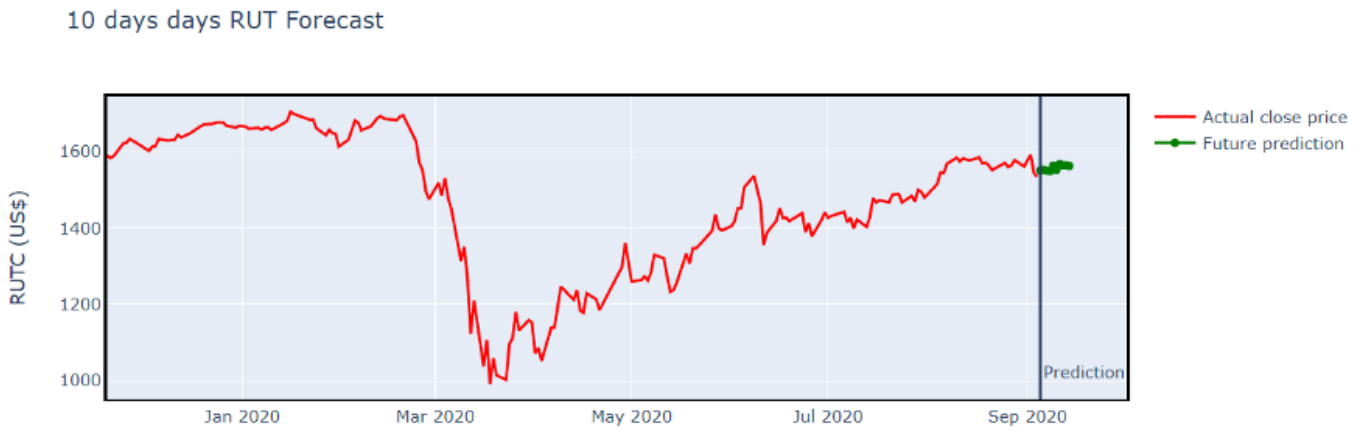


Image by author

Vector Auto Regression (VAR) comes with an advantage in easy implementation. Every equation in the VAR has the same number of variables on the right-hand side, the coefficients {α1, α2, …, β11, β21, …, γ 11, γ 21, … } of the overall system can be easily estimated by applying (OLS) regression to each equation individually. We could estimate this model using the ordinary least squares (OLS) estimator computed

equations, with the standard t and F statistics. The VAR models have the advantage over traditional machine learning models in that the results are not hidden by a large and complicated structure (“black box”), but are easily interpreted and available.

Testing of Granger causality helps to know whether one or more variables have predictive content to forecast and Impulse-response function and variance decomposition are normally used to quantify the effects over time. I already have written in the past about VAR and how multivariate time series can be fitted to generate prediction covering both.

Here, I will discuss simple steps to predict unknown future using VAR.

```
stock = ['^RUT', '^GSPC', '^DJI', '^IXIC' ]
start = pd.to_datetime('1990-01-03')
df = web.DataReader(stock, data_source = 'yahoo', start = start);
print(df.tail());
```

Attributes	Adj Close				Close				High	
Symbols	^RUT	^GSPC	^DJI	^IXIC	^RUT	^GSPC	^DJI	^IXIC	^RUT	^GSPC
Date										
2020-08-31	1561.880005	3500.310059	28430.050781	11775.459961	1561.880005	3500.310059	28430.050781	11775.459961	1577.550049	3514.770020
2020-09-01	1578.579956	3526.649902	28645.660156	11939.669922	1578.579956	3526.649902	28645.660156	11939.669922	1578.579956	3528.030029
2020-09-02	1592.290039	3580.840088	29100.500000	12056.440430	1592.290039	3580.840088	29100.500000	12056.440430	1595.040039	3588.110107
2020-09-03	1544.680054	3455.060059	28292.730469	11458.099609	1544.680054	3455.060059	28292.730469	11458.099609	1592.469971	3564.850098
2020-09-04	1535.300049	3426.959961	28133.310547	11313.129883	1535.300049	3426.959961	28133.310547	11313.129883	1563.380005	3479.149902

Attributes			Low				Open			
Symbols	^DJI	^IXIC	^RUT	^GSPC	^DJI	^IXIC	^RUT	^GSPC	^DJI	^IXIC
Date										
2020-08-31	28643.660156	11829.839844	1561.880005	3493.250000	28363.550781	11697.419922	1577.550049	3509.729980	28643.660156	11718.809570
2020-09-01	28659.259766	11945.719727	1554.189941	3494.600098	28290.720703	11794.780273	1561.099976	3507.439941	28439.609375	11850.959961
2020-09-02	29162.880859	12074.059570	1567.430054	3535.229980	28713.529297	11836.179688	1578.780029	3543.760010	28736.789062	12047.259766
2020-09-03	29199.349609	11894.400391	1538.530029	3427.409912	28074.759766	11361.360352	1592.469971	3564.739990	29090.699219	11861.900391
2020-09-04	28539.750000	11531.179688	1501.520020	3349.629883	27664.679688	10875.870117	1546.640015	3453.600098	28341.050781	11396.240234

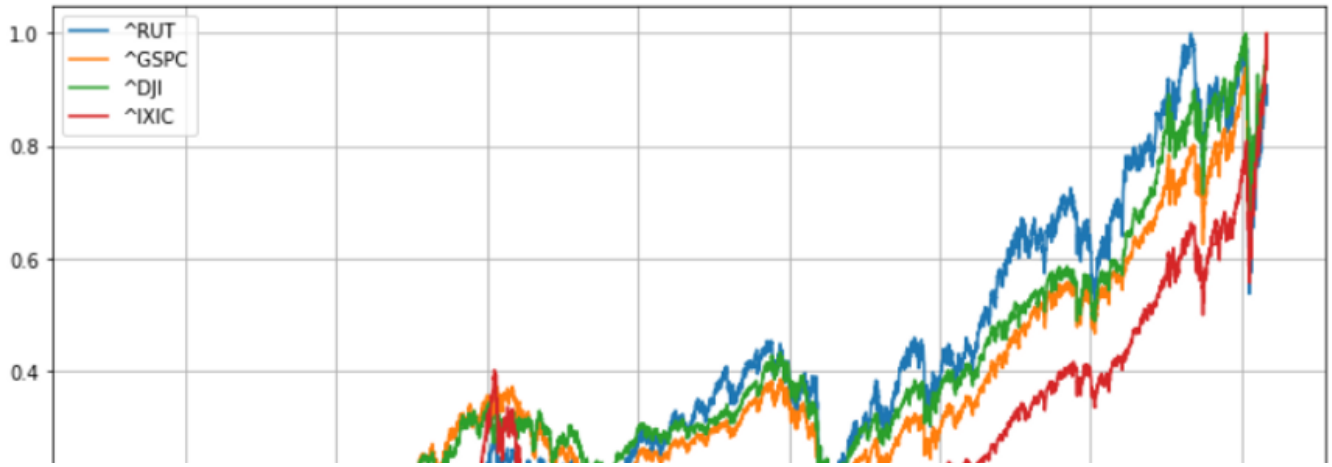
Attributes	Volume			
Symbols	^RUT	^GSPC	^DJI	^IXIC
Date				
2020-08-31	43422900	4342290000	517320000	3596980000
2020-09-01	40831100	4083110000	423410000	3480780000
2020-09-02	42851900	4285190000	539510000	3966140000
2020-09-03	48986800	4898680000	650080000	4437500000
2020-09-04	44314400	4431440000	694640000	4269190000

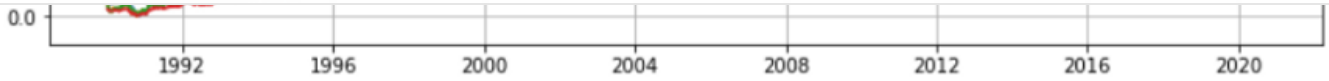
Here, we have data from Russell 2000 (^RUT), S&P 500 (^GSPC), NASDAQ Composite (^IXIC) and Dow Jones Industrial Average (^DJI). Let us separate Adj Close columns for all variables.

Symbols	^RUT	^GSPC	^DJI	^IXIC
Date				
2020-08-31	1561.880005	3500.310059	28430.050781	11775.459961
2020-09-01	1578.579956	3526.649902	28645.660156	11939.669922
2020-09-02	1592.290039	3580.840088	29100.500000	12056.440430
2020-09-03	1544.680054	3455.060059	28292.730469	11458.099609
2020-09-04	1535.300049	3426.959961	28133.310547	11313.129883

Let us visually compare these series in one chart after applying normalization. We can see high correlation among all the variables selected here. This makes a good candidate for multivariate VAR.

```
scaler = MinMaxScaler(feature_range=(0, 1))
sdf_np = scaler.fit_transform(data)
sdf = DataFrame(sdf_np, columns=data.columns, index=data.index)
plt.figure(figsize=(12,6))
plt.grid()
plt.plot(sdf)
plt.legend(sdf.columns)
plt.show()
```

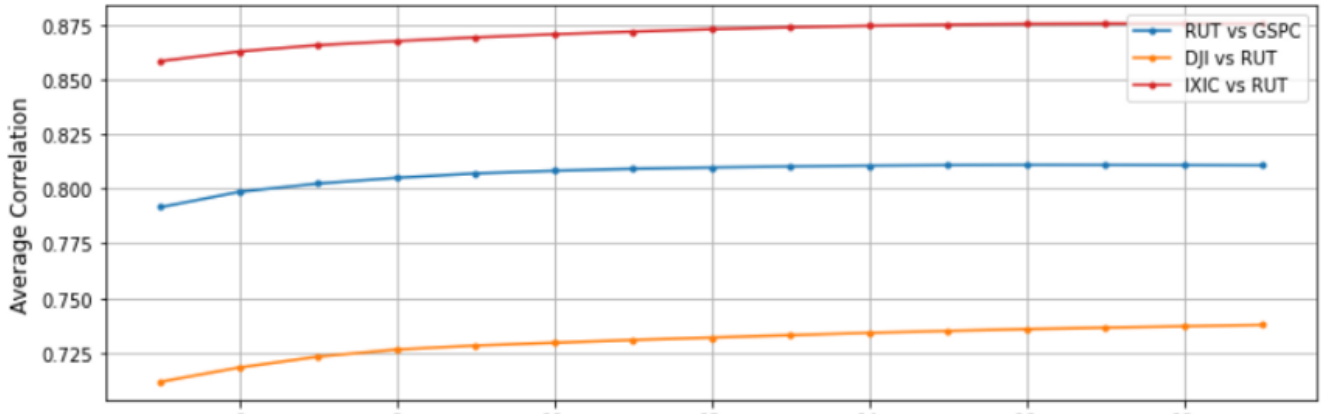




Our goal here is to predict expected future prices of ^RUT and a motivation for the selection of DJI, GSPC and IXIC is their high correlation with RUT.

We can also diagnose the correlation by measuring the average linear correlation over the rolling window in a function of rolling window size; here I have taken window size of 5 and 20 days for visual display.

```
blue, orange, red = '#1f77b4', '#ff7f0e', '#d62728'  # color codes
plt.figure(figsize=(12,4))
plt.grid()
cor1, cor2, cor3 = list(), list(), list()
# average correlation for increasing rolling window size
for win in range(5, 20):  # Days
cor1.append(data['^GSPC'].rolling(win).corr(data['^RUT']) \
.replace([np.inf, -np.inf], np.nan).dropna().mean())
cor2.append(data['^DJI'].rolling(win).corr(data['^RUT']) \
.replace([np.inf, -np.inf], np.nan).dropna().mean())
cor3.append(data['^IXIC'].rolling(win).corr(data['^RUT']) \
.replace([np.inf, -np.inf], np.nan).dropna().mean())

plt.plot(range(5, 20), cor1, '.-', color=blue, label='RUT vs GSPC')
plt.plot(range(5, 20), cor2, '.-', color=orange, label='DJI vs RUT')
plt.plot(range(5, 20), cor3, '.-', color=red, label='IXIC vs RUT')
plt.legend()
plt.xlabel('Rolling Window Length [Days]', fontsize=12)
plt.ylabel('Average Correlation', fontsize=12)
plt.show()
```



In the context of variables or features selection, we need to decide which variables to include into the model. Since we can not and should not include all variables of potential interest, we have to have a priori ideas when choosing variables.

ADFuller to test for Stationarity

We need to get rid of trend in the data to let the model work on prediction. Let us check the stationarity of our data set.

```
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.")

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

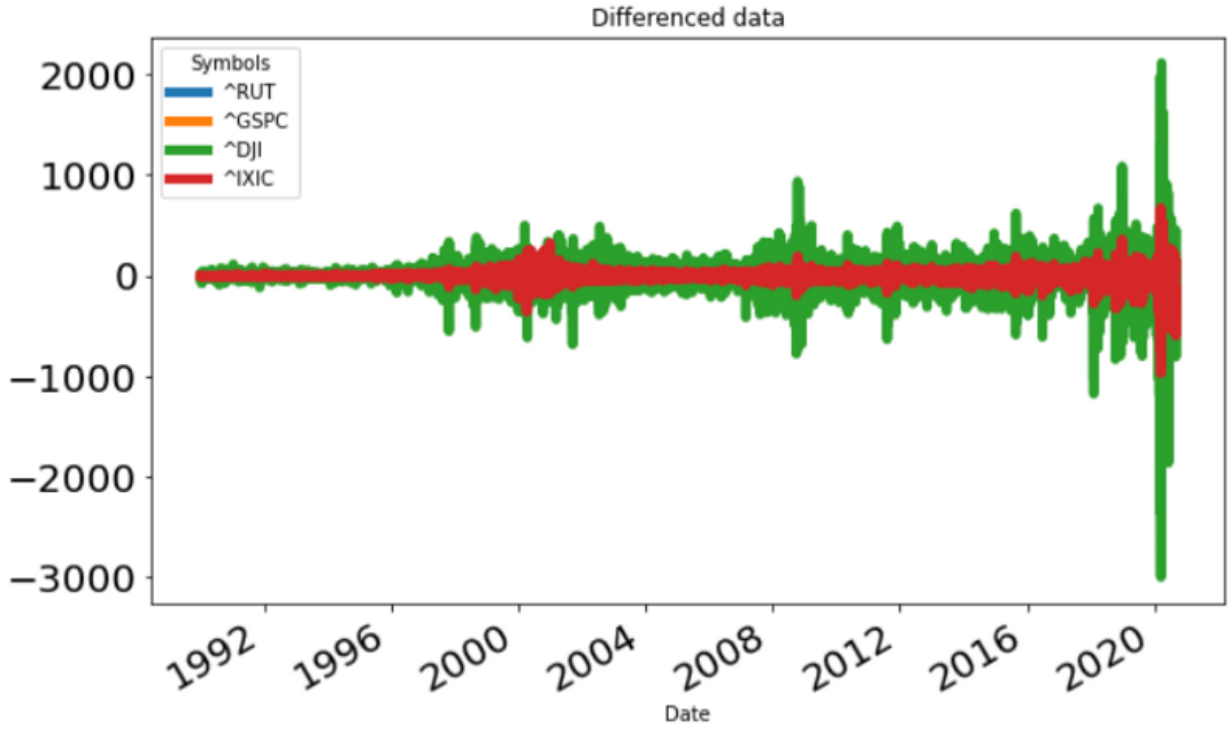
```
Augmented Dickey-Fuller Test on "^RUT"
    -----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level = 0.05
Test Statistic = -0.3981
No. Lags Chosen = 35
 Critical value 1%     = -3.431
```

```
 => Series is Non-Stationary.
Augmented Dickey-Fuller Test on "^GSPC"
    ----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level = 0.05
Test Statistic = 1.1281
No. Lags Chosen = 36
 Critical value 1%     = -3.431
 Critical value 5%     = -2.862
 Critical value 10%    = -2.567
 => P-Value = 0.9954. Weak evidence to reject the Null Hypothesis.
 => Series is Non-Stationary.
Augmented Dickey-Fuller Test on "^DJI"


Augmented Dickey-Fuller Test on "^DJI"
    ----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level = 0.05
Test Statistic = 0.5528
No. Lags Chosen = 36
 Critical value 1%     = -3.431
 Critical value 5%     = -2.862
 Critical value 10%    = -2.567
 => P-Value = 0.9864. Weak evidence to reject the Null Hypothesis.
 => Series is Non-Stationary.
Augmented Dickey-Fuller Test on "^IXIC"
    ----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level = 0.05
Test Statistic = 2.5676
No. Lags Chosen = 36
 Critical value 1%     = -3.431
 Critical value 5%     = -2.862
 Critical value 10%    = -2.567
 => P-Value = 0.9991. Weak evidence to reject the Null Hypothesis.
 => Series is Non-Stationary.
```

It is clear that our existing data set is non-stationary. Let us take 1st order difference and check the stationarity again.

```
data_diff = data.diff()
data_diff.dropna(inplace=True)
print('Glimpse of differenced data:')
print(data_diff.head())

# plotting differenced data
data_diff.plot(figsize=(10,6), linewidth=5, fontsize=20)
plt.title('Differenced data')
plt.show()
```

```
Glimpse of differenced data:
Symbols         ^RUT     ^GSPC       ^DJI     ^IXIC
Date
1990-01-04 -0.699997 -3.089996 -13.649902 -1.500000
1990-01-05 -0.440002 -3.470001 -22.830078 -1.199982
1990-01-08 -0.099991  1.589996  21.120117  0.500000
1990-01-09 -0.250000 -4.170013 -28.370117 -1.900024
1990-01-10 -1.360001 -2.309998 -15.360107 -6.099976
```



From the plot, we can assess that 1st order differenced has made the data stationary. However, let us run ADF test gain to validate.

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

```
Augmented Dickey-Fuller Test on "^RUT"
    ----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level = 0.05
Test Statistic = -15.9264
No. Lags Chosen = 34
 Critical value 1%     = -3.431
 Critical value 5%     = -2.862
 Critical value 10%    = -2.567
 => P-Value = 0.0. Rejecting Null Hypothesis.
 => Series is Stationary.
Augmented Dickey-Fuller Test on "^GSPC"
    ----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level = 0.05
Test Statistic = -15.7142
No. Lags Chosen = 35
 Critical value 1%     = -3.431
 Critical value 5%     = -2.862
 Critical value 10%    = -2.567
 => P-Value = 0.0. Rejecting Null Hypothesis.
 => Series is Stationary.
Augmented Dickey-Fuller Test on "^DJI"
```

```
Augmented Dickey-Fuller Test on "^DJI"
    ----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level = 0.05
Test Statistic = -15.8337
No. Lags Chosen = 35
 Critical value 1%     = -3.431
 Critical value 5%     = -2.862
 Critical value 10%    = -2.567
 => P-Value = 0.0. Rejecting Null Hypothesis.
 => Series is Stationary.
Augmented Dickey-Fuller Test on "^IXIC"
    ----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level = 0.05
Test Statistic = -14.6937
No. Lags Chosen = 36
 Critical value 1%     = -3.431
 Critical value 5%     = -2.862
```

Our data is stationarized to fit into regression model.

Vector Auto Regression

One model is specified, the appropriate lag length of the VAR model has to be decided. In deciding the number of lags, it has been common to use a statistical method, like the Akaike information criteria.

```
var_model = smt.VAR(data_diff)
res = var_model.select_order(maxlags=15)
print(res.summary())

results = var_model.fit(maxlags=15, ic='aic')
print(results.summary())
```

VAR Order Selection (* highlights the minimums)

	AIC	BIC	FPE	HQIC
0	22.27	22.27	4.692e+09	22.27
1	22.24	22.25*	4.543e+09	22.24
2	22.22	22.26	4.482e+09	22.23
3	22.22	22.26	4.459e+09	22.23
4	22.21	22.27	4.435e+09	22.23
5	22.21	22.29	4.432e+09	22.24
6	22.20	22.29	4.385e+09	22.23
7	22.18	22.29	4.310e+09	22.22
8	22.17	22.29	4.257e+09	22.21
9	22.16	22.30	4.221e+09	22.21*
10	22.16	22.31	4.205e+09	22.21
11	22.16	22.32	4.212e+09	22.22
12	22.15	22.33	4.183e+09	22.21
13	22.15	22.34	4.172e+09	22.22
14	22.15	22.36	4.165e+09	22.22
15	22.15*	22.37	4.156e+09*	22.22

```
^GSPC    0.868357  1.000000  0.966471  0.895372
^DJI     0.834560  0.966471  1.000000  0.813201
^IXIC    0.813366  0.895372  0.813201  1.000000
```

Future predictions

Now that the model is fitted, let us call for the prediction.

```
# make predictions
pred = results.forecast(results.y, steps=nobs)
pred = DataFrame(pred, columns = data.columns+ '_pred')
print(pred)
```

Symbols	^RUT_pred	^GSPC_pred	^DJI_pred	^IXIC_pred
0	-1.111631	-3.210144	-54.811203	-18.877887
1	2.102828	8.201075	40.130150	25.051525
2	-2.482367	1.747004	-3.403546	-2.523552
3	-1.419123	3.711129	1.150103	14.349551
4	14.423217	18.877488	134.556490	65.639334
5	-10.781957	-22.649263	-201.510033	-72.717388
6	14.871588	26.855996	237.667891	78.530426
7	-3.151807	-5.778967	-53.187367	-15.475223
8	0.008099	1.054497	22.572021	10.507356
9	-1.555348	-4.087194	-11.226234	-19.971864

Similar observations can be obtained with the below lines of codes. In both cases we get the output but on differenced scale because our input data was differenced in order to stationarize.

	0	1	2	3
0	-1.111631	-3.210144	-54.811203	-18.877887
1	2.102828	8.201075	40.130150	25.051525
2	-2.482367	1.747004	-3.403546	-2.523552
3	-1.419123	3.711129	1.150103	14.349551
4	14.423217	18.877488	134.556490	65.639334
5	-10.781957	-22.649263	-201.510033	-72.717388
6	14.871588	26.855996	237.667891	78.530426
7	-3.151807	-5.778967	-53.187367	-15.475223
8	0.008099	1.054497	22.572021	10.507356
9	-1.555348	-4.087194	-11.226234	-19.971864

Invert transform data to original shape

```
pred = DataFrame(pred, columns=data.columns+ '_pred')

def invert_transformation(data_diff, pred):
  forecast = pred.copy()
  columns = data.columns
  for col in columns:
    forecast[str(col)+'_pred'] = data[col].iloc[-1] +
forecast[str(col) +'_pred'].cumsum()
  return forecast

output = invert_transformation(data_diff, pred)
print(output.loc[:, ['^RUT_pred']])
output = DataFrame(output['^RUT_pred'])
print(output)
```

	^RUT_pred
0	1551.368349

2	1550.988810
3	1549.569688
4	1563.992905
5	1553.210948
6	1568.082536
7	1564.930729
8	1564.938828
9	1563.383480

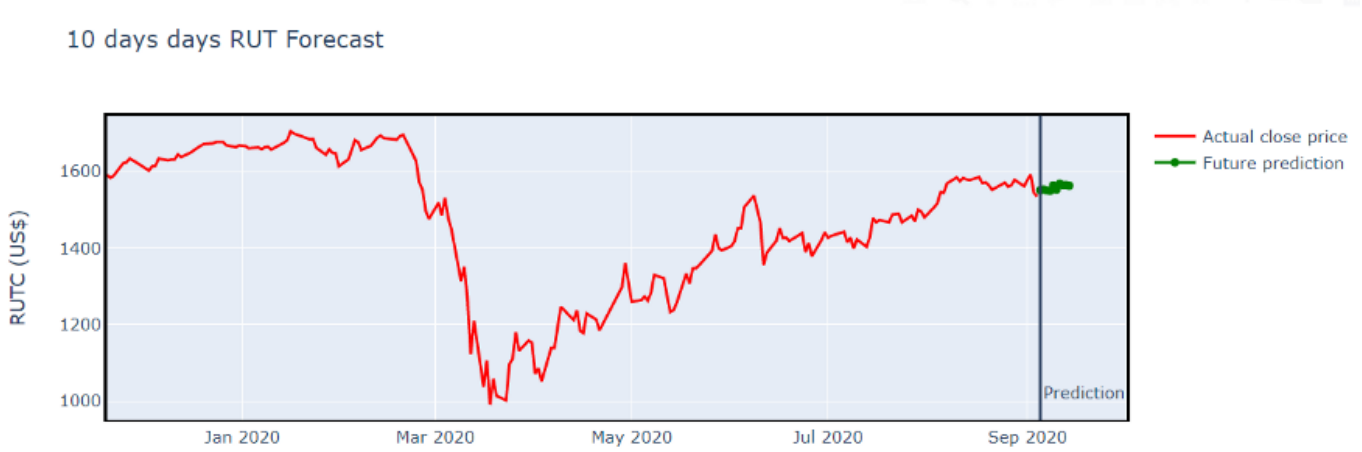
Above are the future 10 days prediction; let us assign future dates to these values.

Assigning future dates

```
d = data.tail(nobs)
d.reset_index(inplace = True)
d = d.append(DataFrame({'Date': pd.date_range(start =
d.Date.iloc[-1], periods = (len(d)+1), freq = 'd', closed =
'right')}))
d.set_index('Date', inplace = True)
d = d.tail(nobs)
output.index = d.index
print(output)
```

Date	^RUT_pred
2020-09-05	1551.368349
2020-09-06	1553.471177
2020-09-07	1550.988810
2020-09-08	1549.569688
2020-09-09	1563.992905
2020-09-10	1553.210948
2020-09-11	1568.082536
2020-09-12	1564.930729
2020-09-13	1564.938828
2020-09-14	1563.383480

```
fig = go.Figure()
n = output.index[0]
fig.add_trace(go.Scatter(x = data.index[-200:], y = data['^RUT']
[-200:], marker = dict(color ="red"), name = "Actual close price"))
fig.add_trace(go.Scatter(x = output.index, y = output['^RUT_pred'],
marker=dict(color = "green"), name = "Future prediction"))
fig.update_xaxes(showline = True, linewidth = 2, linecolor='black',
mirror = True, showspikes = True,)
fig.update_yaxes(showline = True, linewidth = 2, linecolor='black',
mirror = True, showspikes = True,)
fig.update_layout(title= "10 days days RUT Forecast", yaxis_title =
'RUTC (US$)', hovermode = "x", hoverdistance = 100, # Distance to
show hover label of data point spikedistance = 1000,shapes = [dict(
x0 = n, x1 = n, y0 = 0, y1 = 1, xref = 'x', yref = 'paper',
line_width = 2)], annotations = [dict(x = n, y = 0.05, xref = 'x',
yref = 'paper', showarrow = False, xanchor = 'left', text =
'Prediction')])
fig.update_layout(autosize = False, width = 1000, height = 400,)

fig.show()
```



Here, we see how close VAR can predict future prices.

Key takeaways

VAR is easy to estimate. It has good forecasting capabilities; VAR model has the ability to capture dynamic structure of the time series variables and typically treat all variables as

We have covered here as how to find the maximum lags and fitting a VAR model with transformed data. Once the model is fitted, next step is to predict multi-step future prices and invert the transformed data back to original shape to see the actual predicted price.

In the final stage, plotting the historical and predicted price gives a clear visual representation of our prediction.

I can be reached ***here***.

Note: The programs described here are experimental and should be used with caution for any commercial purpose. All such use at your own risk.

Vector Auto Regression | Predictive Modeling | Ordinary Least Square | Linear Regression Python