Granger’s Causality and Vector Auto Regressive Model for Time Series Forecasting

Sarit Maitra Follow
Oct 7 · 11 min read ★

FORECASTING of ***Gold*** and ***Oil*** have garnered major attention from academics, investors and Government agencies like. These two products are known for their substantial influence on global economy. I will show here, how to use ***Granger’s Causality Test*** to test the relationships of multiple variables in the time series and ***Vector***

Auto Regressive Model (VAR) to forecast the future ***Gold & Oil*** prices from the historical data of ***Gold prices, Silver prices, Crude Oil prices, Stock index , Interest Rate and USD rate***.

The ***Gold*** prices are closely related with other commodities. A hike in ***Oil*** prices will have positive impact on ***Gold*** prices and vice versa. Historically, we have seen that, when there is a hike in equities, ***Gold*** prices goes down.

- Time-series forecasting problem formulation
- Uni-variate and multi-variate time series time-series forecasting
- Apply ***VAR*** to this problem

Let's understand how a multivariate time series is formulated. Below are the simple K-equations of multivariate time series where each equation is a lag of the other series. X is the exogeneous series here. The objective is to see if the series is affected by its own past and also the past of the other series.

$$y_{1,t} = f_1\left(y_{1,t-1}\ldots, y_{k,t-1}\ldots\ y_{1,t-p}, \ \ldots\ y_{k,t-p}\ldots X_{t-1}\ X_{t-2}\ldots\right)$$

$$\vdots \qquad \vdots$$

$$y_{K,t} = f_k\left(y_{1,t-1}\ldots, y_{k,t-1}\ldots\ y_{1,t-p}, \ \ldots\ y_{k,t-p}\ldots X_{t-1}\ X_{t-2}\ldots\right)$$

This kind of series allow us to model the dynamics of the series itself and also the interdependence of other series. We will explore this inter-dependence through ***Granger's Causality Analysis***.

Exploratory analysis

Let's load the data and do some analysis with visualization to know insights of the data. Exploratory data analysis is quite extensive in multivariate time series. I will cover some areas here to get insights of the data. However, it is advisable to conduct all statistical tests to ensure our clear understanding on data distribution.

> *Exploratory analysis is an iterative process where some of the tests (e.g. stationarity tests) might need to do repeatedly to ensure the required output.*

```
dataset = pd.concat([df1.Gold, df2.Silver, df3.USD, df4.Oil, df5.Interest, df6.Stock], axis=1) # combining dataframes
dataset.head()
```

Date	Gold	Silver	USD	Oil	Interest	Stock
1999-12-31	17.9375	NaN	NaN	NaN	6.377	6876.100098
2000-01-03	17.5625	NaN	99.89	NaN	6.498	6762.109863
2000-01-04	17.3125	5.335	100.10	25.55	6.530	6543.759766
2000-01-05	17.5625	5.170	100.05	24.91	6.521	6567.029785
2000-01-06	17.7500	5.127	100.34	24.78	6.558	6635.439941

Let's fix the dates for all the series.

```
dataset = dataset.loc['20000101':'20180501']
dataset.head()
```

Date	Gold	Silver	Oil	USD	Interest	Stock
2000-01-03	17.5625	NaN	NaN	101.39	6.498	6762.109863
2000-01-04	17.3125	5.290	25.20	100.18	6.530	6543.759766
2000-01-05	17.5625	5.170	25.50	100.09	6.521	6567.029785
2000-01-06	17.7500	5.127	24.80	100.05	6.558	6635.439941
2000-01-07	17.8125	5.150	24.65	100.18	6.545	6792.669922

```
dataset.isnull().sum() # missing values
```

```
Gold        519
Silver      130
Oil         477
USD         435
Interest    523
Stock       519
dtype: int64
```

```
dataset= dataset.fillna(method ='pad') # filling the missing values with previous ones
dataset.isnull().sum()
```

```
Gold        0
```

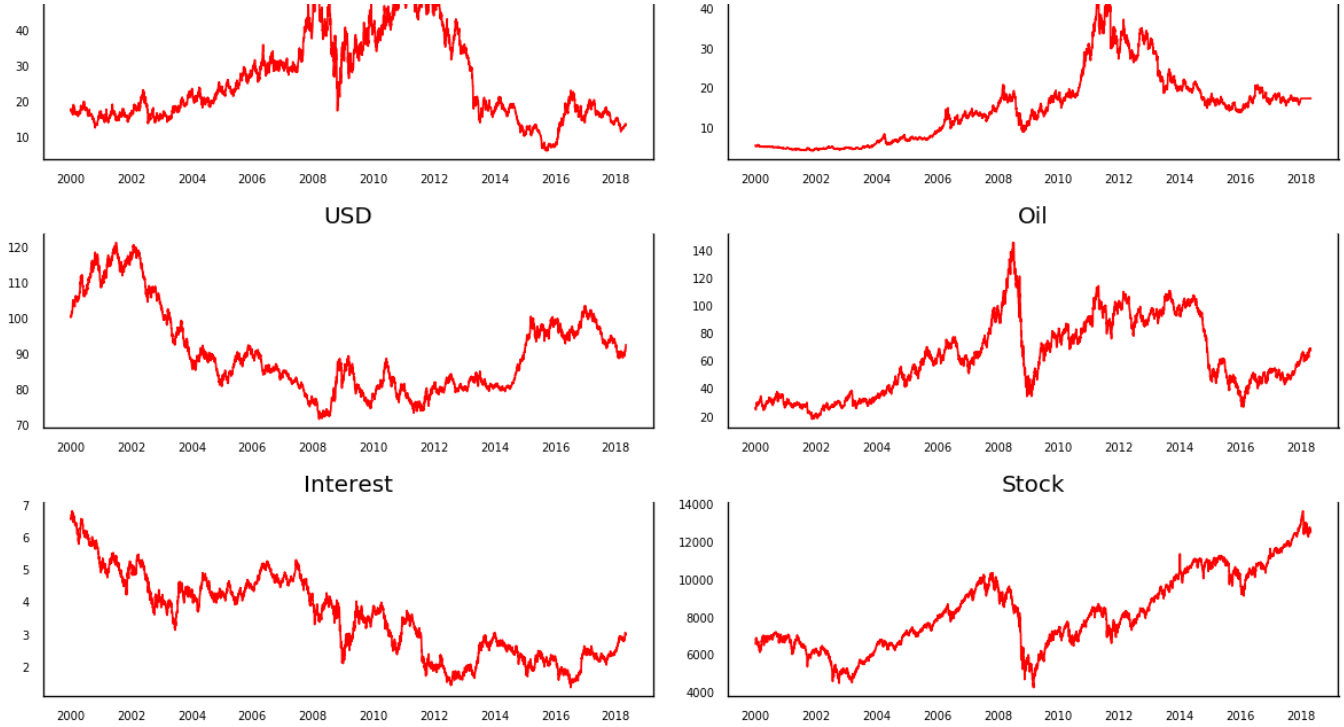
```
Silver      1
Oil         1
USD         0
Interest    0
Stock       0
dtype: int64
```

```
dataset = dataset.dropna()
dataset.head()
```

Date	Gold	Silver	Oil	USD	Interest	Stock
2000-01-04	17.3125	5.290	25.20	100.18	6.530	6543.759766
2000-01-05	17.5625	5.170	25.50	100.09	6.521	6567.029785
2000-01-06	17.7500	5.127	24.80	100.05	6.558	6635.439941
2000-01-07	17.8125	5.150	24.65	100.18	6.545	6792.669922
2000-01-10	17.4375	5.145	24.22	100.36	6.540	6838.450195

The NaN values in the data are filled with previous days data. After doing some necessary pre-processing, the dataset now clean for further analysis.

```
from pandas.plotting import register_matplotlib_converters
register_matplotlib_converters()

# Plot
fig, axes = plt.subplots(nrows=3, ncols=2, dpi=120, figsize=(10,6))
for i, ax in enumerate(axes.flatten()):
data = dataset[dataset.columns[i]]
ax.plot(data, color='red', linewidth=1)
# Decorations
ax.set_title(dataset.columns[i])
ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('none')
ax.spines["top"].set_alpha(0)
ax.tick_params(labelsize=6)

plt.tight_layout();
```



To extract maximum information from our data, it is important to have a normal or Gaussian distribution of the data. To check for that, we have done a normality test based on the Null and Alternate Hypothesis intuition.

```
from scipy import stats
stat,p = stats.normaltest(dataset.Gold)
print('Statistics=%.3f, p=%.3f' % (stat,p))
alpha = 0.05
if p > alpha:
  print('Data looks Gaussian (fail to reject H0)')
else:
  print('Data do not look Gaussian (reject H0)')
```

```
Statistics=658.293, p=0.000
Data do not look Gaussian (reject H0)
```

```
print('Kurtosis of normal distribution: {}'. format(stats.kurtosis(dataset.Gold)))
print('Skewness of normal distribution: {}'. format(stats.skew(dataset.Gold)))
```

```
Kurtosis of normal distribution: -0.7547875128340102
Skewness of normal distribution: 0.6561298945285786
```

```
print('Kurtosis of normal distribution: {}'. format(stats.kurtosis(dataset.Oil)))
print('Skewness of normal distribution: {}'. format(stats.skew(dataset.Oil)))
```

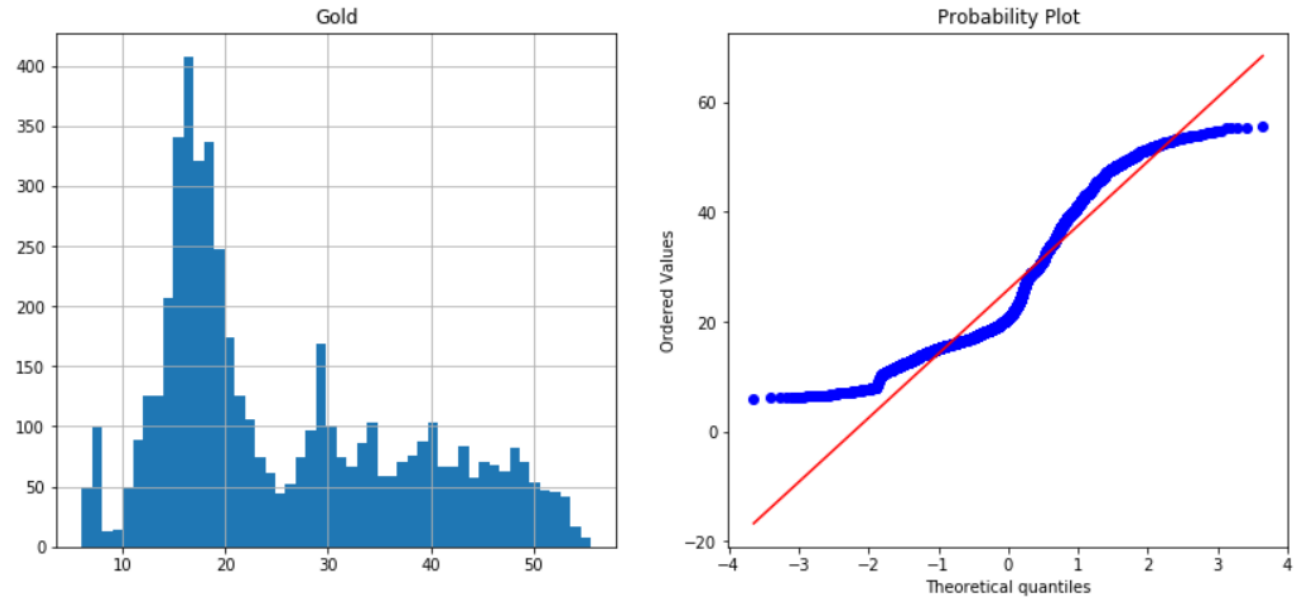
```
Kurtosis of normal distribution: -0.9559303293518187
```

```
Skewness of normal distribution: 0.2854153692916605
```

These two distributions give us some intuition about the distribution of our data. A value close to 0 for Kurtosis indicates a Normal Distribution where asymmetrical nature is signified by a value between -0.5 and +0.5 for skewness. The tails are heavier for kurtosis greater than 0 and vice versa. Moderate skewness refers to the value between -1 and -0.5 or 0.5 and 1.

```
plt.figure(figsize=(14,6))
plt.subplot(1,2,1)
dataset['Gold'].hist(bins=50)
plt.title('Gold')
plt.subplot(1,2,2)
stats.probplot(dataset['Gold'], plot=plt);
dataset.Gold.describe().T
```

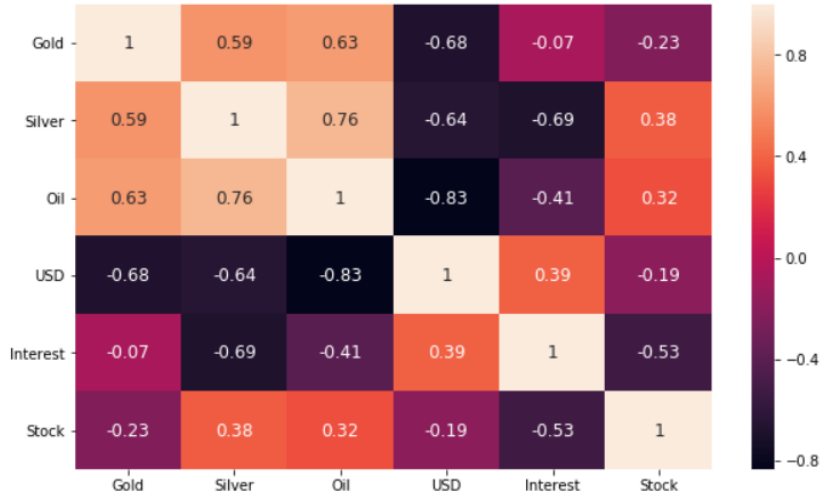
```
count    5129.000000
mean       25.828860
std        12.218568
min         6.080000
25%        16.379999
50%        20.700001
75%        34.980000
max        55.540001
Name: Gold, dtype: float64
```



Normal probability plot also shows the data is far from normally distributed.

```
corr = dataset.corr()
```

```
sns.heatmap(corr, xticklabels=corr.columns.values, yticklabels=corr.columns.values, annot = True, annot_kws={'size':12})
heat_map=plt.gcf()
heat_map.set_size_inches(10,6)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
plt.show()
```



Steps to build VAR model

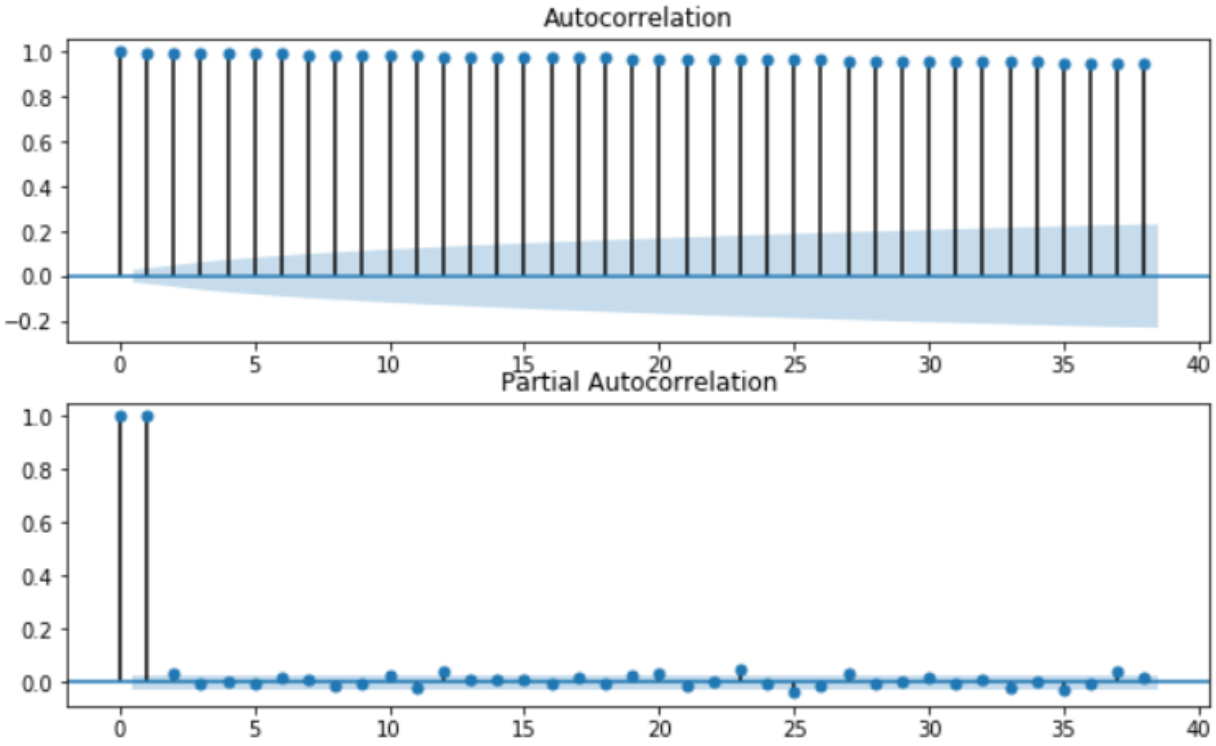
I will follow some procedural steps as mentioned below to achieve the goal here-

- Exploratory data analysis on training set
- Stationarity test
- Split the series into training and validation sets
- Transform the series
- Build a model on transformed series
- Model diagnostic
- Model selection (based on pre-defined criteria)
- Conduct forecast using the final, chosen model
- Inverse transform the forecast
- Conduct forecast evaluation

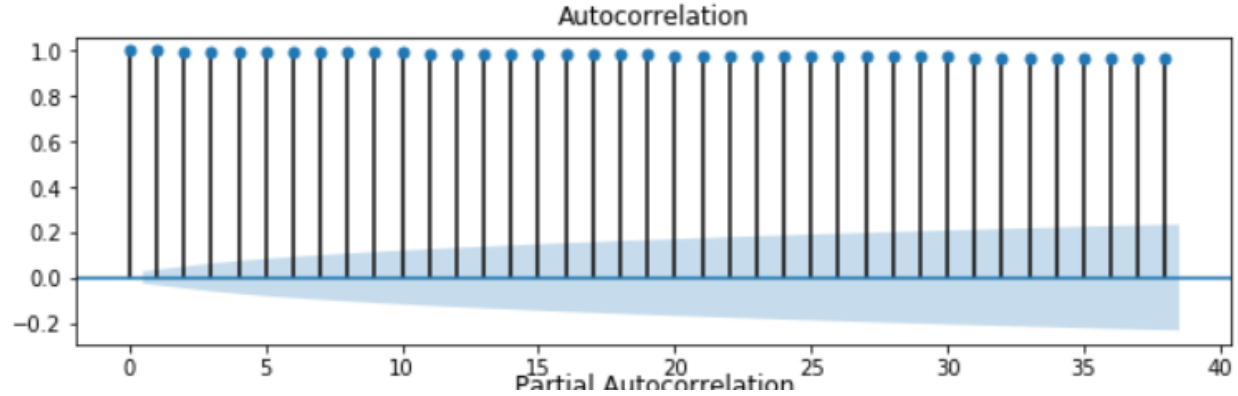
Autocorrelation

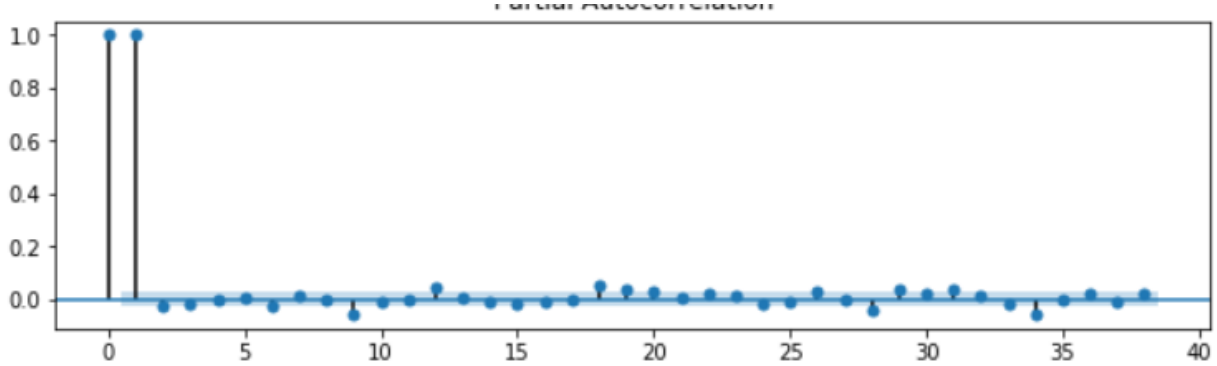
Autocorrelation or serial correlation can be a significant problem in analyzing historical data if we do not know how to look out for it.

```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
fig, ax = plt.subplots(2, figsize=(10,6)) # Gold acf & pacf on training data
ax[0] = plot_acf(dataset.Gold, ax=ax[0])
ax[1] = plot_pacf(dataset.Gold, ax=ax[1])
```




```
fig, ax = plt.subplots(2, figsize=(10,6)) # Silver acf & pacf on training data
ax[0] = plot_acf(dataset.Silver, ax=ax[0])
ax[1] = plot_pacf(dataset.Silver, ax=ax[1])
```





We see here from the above plots, the autocorrelation of +1 which represents a perfect positive correlation which means, an increase seen in one time series leads to a proportionate increase in the other time series. We definitely need to apply transformation and neutralize this to make the series stationary. It measures linear relationships; even if the autocorrelation is minuscule, there may still be a nonlinear relationship between a time series and a lagged version of itself.

Split the Series into Training and Testing Data

The VAR model will be fitted on X_train and then used to forecast the next 15 observations. These forecasts will be compared against the actuals present in test data.

```
nobs = 15
X_train, X_test = dataset[0:-nobs], dataset[-nobs:]

# Check size
print(X_train.shape)
print(X_test.shape)
```

```
(5114, 6)
(15, 6)
```

Transformation

Applying first differencing on training set should make all the 6 series stationary. However, this is an iterative process where we after first differencing, the series may still be non-stationary. We shall have to apply second difference or log transformation to standardize the series in such cases.

```
transform_data = X_train.diff().dropna()
transform_data.head()
```

Date	Gold	Silver	Oil	USD	Interest	Stock
2000-01-05	0.2500	-0.120	0.30	-0.09	-0.009	23.270019
2000-01-06	0.1875	-0.043	-0.70	-0.04	0.037	68.410156
2000-01-07	0.0625	0.023	-0.15	0.13	-0.013	157.229981
2000-01-10	-0.3750	-0.005	-0.43	0.18	-0.005	45.780273
2000-01-11	0.0000	0.050	0.49	0.29	0.060	-63.970215

```
transform_data.describe()
```

	Gold	Silver	Oil	USD	Interest	Stock
count	5113.000000	5113.000000	5113.000000	5113.000000	5113.000000	5113.000000
mean	-0.000886	0.002330	0.007452	-0.002080	-0.000730	1.141561
std	0.699671	0.359989	1.300758	0.460318	0.056205	84.506113
min	-5.310002	-5.677000	-11.490000	-2.545000	-0.522000	-990.589850
25%	-0.280001	-0.085000	-0.580000	-0.255000	-0.031000	-32.030274
50%	0.000000	0.002000	0.000000	0.000000	0.000000	0.000000
75%	0.280001	0.106000	0.640000	0.250000	0.029000	38.540039
max	6.309999	2.585000	9.770000	2.365000	0.334000	992.450200

Stationarity check

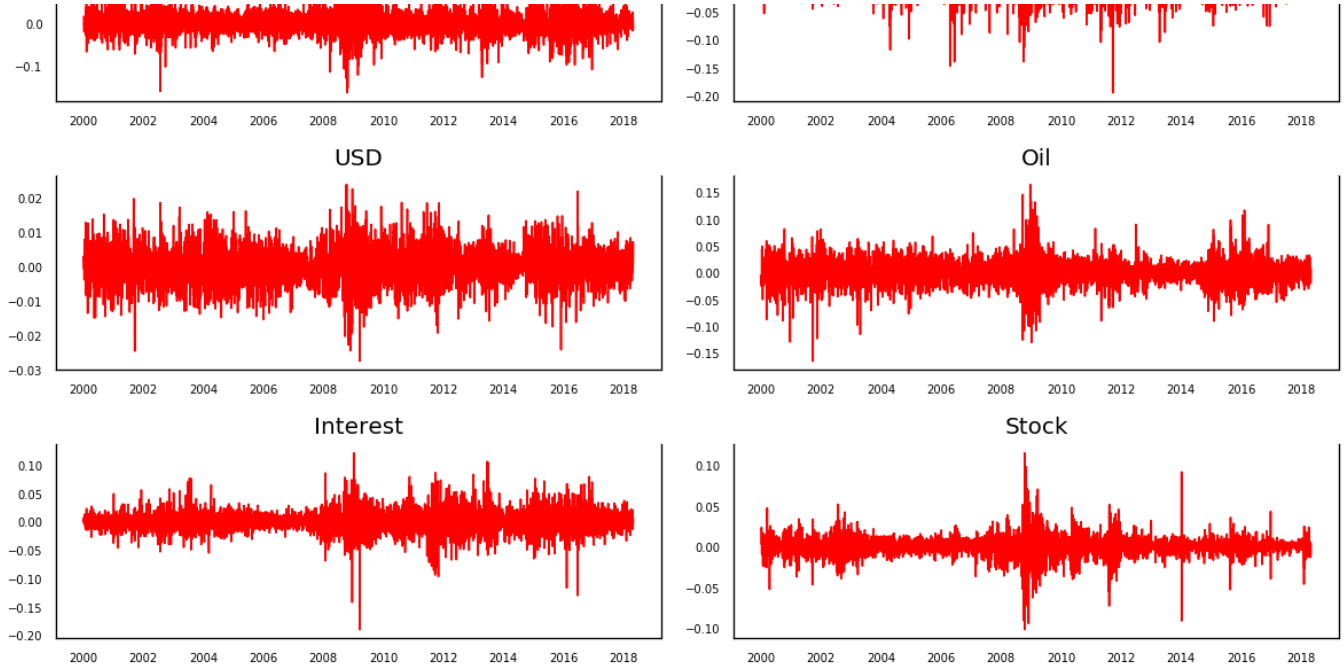
```
def adfuller_test(series, signif=0.05, name='', verbose=False):
    """Perform ADFuller to test for Stationarity of given series and print report"""
    r = adfuller(series, autolag='AIC')
    output = {'test_statistic':round(r[0], 4), 'pvalue':round(r[1], 4), 'n_lags':round(r[2], 4), 'n_obs':r[3]}
    p_value = output['pvalue']
    def adjust(val, length= 6): return str(val).ljust(length)

    # Print Summary
    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.")
```

```
        print(f' => Series is Non-Stationary.')

# ADF Test on each column
for name, column in transform_data.iteritems():
    adfuller_test(column, name=column.name)
    print('\n')
```

```
Augmented Dickey-Fuller Test on "Gold"
    -----------------------------------------------
 Null Hypothesis: Data has unit root. Non-Stationary.
 Significance Level    = 0.05
 Test Statistic        = -15.4519
 No. Lags Chosen       = 25
 Critical value 1%     = -3.432
 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 "Silver"
    -----------------------------------------------
 Null Hypothesis: Data has unit root. Non-Stationary.
 Significance Level    = 0.05
 Test Statistic        = -13.1089
 No. Lags Chosen       = 32
 Critical value 1%     = -3.432
 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 "Oil"
    -----------------------------------------------
 Null Hypothesis: Data has unit root. Non-Stationary.
 Significance Level    = 0.05
 Test Statistic        = -16.8349
 No. Lags Chosen       = 14
 Critical value 1%     = -3.432
 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 "USD"
    -----------------------------------------------
 Null Hypothesis: Data has unit root. Non-Stationary.
 Significance Level    = 0.05
 Test Statistic        = -74.7055
 No. Lags Chosen       = 0
 Critical value 1%     = -3.432
```

```
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 "Interest"
   -----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = -53.5275
No. Lags Chosen       = 1
Critical value 1%     = -3.432
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 "Stock"
   -----------------------------------------------
Null Hypothesis: Data has unit root. Non-Stationary.
Significance Level    = 0.05
Test Statistic        = -17.3646
No. Lags Chosen       = 17
Critical value 1%     = -3.432
Critical value 5%     = -2.862
Critical value 10%    = -2.567
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.
```

```
# Plot
fig, axes = plt.subplots(nrows=3, ncols=2, dpi=120, figsize=(10,6))
for i, ax in enumerate(axes.flatten()):
    data = transform_data[transform_data.columns[i]]
    ax.plot(data, color='red', linewidth=1)
    # Decorations
    ax.set_title(transform_data.columns[i])
    ax.xaxis.set_ticks_position('none')
    ax.yaxis.set_ticks_position('none')
    ax.spines["top"].set_alpha(0)
    ax.tick_params(labelsize=6)

plt.tight_layout();
```



Testing Causation using Granger's Causality Test

The basis behind VAR is that each of the time series in the system influences each other. That is, we can predict the series with past values of itself along with other series in the system. Using Granger's Causality Test, it's possible to test this relationship before even building the model.

Granger's causality Tests the null hypothesis that the coefficients of past values in the regression equation is zero. In simpler terms, the past values of time series (x) do not cause the other series (y). So, if the p-value obtained from the test is lesser than the significance level of 0.05, then, you can safely reject the null hypothesis. This has been performed on original dataset.

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

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

	Gold_x	Silver_x	Oil_x	USD_x	Interest_x	Stock_x

Gold_y	1.0000	0.0000	0.0542	0.5176	0.0333	0.0212
Silver_y	0.0000	1.0000	0.1449	0.2425	0.4341	0.4381
Oil_y	0.0000	0.0000	1.0000	0.0018	0.0000	0.0128
USD_y	0.0000	0.0000	0.2319	1.0000	0.1949	0.3819
Interest_y	0.0083	0.0182	0.2298	0.1985	1.0000	0.2038
Stock_y	0.0000	0.0000	0.0347	0.0297	0.0000	1.0000

The row are the response (y) and the columns are the predictor series (x).

- If we take the value 0.0000 in (row 1, column 2), it refers to the p-value of the Granger's Causality test for Silver_x causing Gold_y. The 0.0000 in (row 2, column 1) refers to the p-value of Gold_y causing Silver_x and so on.
- We can see that, in the case of *Interest* and *USD* variables, we cannot reject null hypothesis e.g. *USD* & *Silver, USD & Oil.* Our variables of interest are *Gold* and *Oil* here. So, for *Gold,* all the variables cause but for *USD* doesn't causes any effect on *Oil*.

So, looking at the p-Values, we can assume that, except USD, all the other variables (time series) in the system are interchangeably causing each other. This justifies the VAR modeling approach for this system of multi time-series to forecast.

Cointegration Test

Cointegration test helps to establish the presence of a statistically significant connection between the variables. Let's define the 'order of integration' (d) which is the number of differencing required to make a non-stationary time series stationary. Here, in the time-series there exists a linear combination of the series that has an order of integration (d) less than that of the individual series, then the collection of series is said to be cointegrated. Also, when the time series are cointegrated, it means they have a long run, statistically significant relationships. This is the basic premise on which VAR model is based on. Johansen cointegration test is also performed on original dataset.

```
from statsmodels.tsa.vector_ar.vecm import coint_johansen

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

    # Summary
    print('Name   ::  Test Stat > C(95%)    =>   Signif  \n', '--'*20)
```

```
    for col, trace, cvt in zip(transform_data.columns, traces, cvts):
        print(adjust(col), ':: ', adjust(round(trace,2), 9), ">", adjust(cvt, 8), ' =>  ' , trace > cvt)

cointegration_test(X_train)
```

```
Name   ::  Test Stat > C(95%)    =>   Signif
 --------------------------------------
Gold   ::  65.62     > 83.9383   =>   False
Silver ::  40.49     > 60.0627   =>   False
Oil    ::  17.53     > 40.1749   =>   False
USD    ::  9.19      > 24.2761   =>   False
Interest ::  4.64      > 12.3212   =>   False
Stock  ::  0.41      > 4.1296    =>   False
```

VAR model

VAR requires stationarity of the series which means the mean to the series do not change over time (we can find this out from the plot drawn next to Augmented Dickey-Fuller Test).

Let's understand the mathematical intuition of VAR model.

$$y_{1,t} = c_1 + \Phi_{11,1} y_{1,t-1} + \Phi_{K2,1} y_{2,t-1} + \ldots\ldots\ldots \Phi_{1K,1} y_{K,t-1} + \varepsilon_{1,t}$$

$$y_{2,t} = c_2 + \Phi_{21,1} y_{1,t-1} + \Phi_{K2,1} y_{2,t-1} + \ldots\ldots\ldots \Phi_{2K,1} y_{K,t-1} + \varepsilon_{2,t}$$

$$\vdots$$

$$y_{K,t} = c_K + \Phi_{K1,1} y_{1,t-1} + \Phi_{K2,1} y_{2,t-1} + \ldots\ldots\ldots \Phi_{KK,1} y_{K,t-1} + \varepsilon_{K,t}$$

Here each series is modeled by it's own lag and other series' lag. y{1,t-1, y{2,t-1}…. are the lag of time series y1,y2,…. respectively. The above equations are referred to as a VAR (1) model, because, each equation is of order 1, that is, it contains up to one lag of each of the predictors (y1, y2, …..). Since the y terms in the equations are interrelated, the y's are considered as endogenous variables, rather than as exogenous predictors. To thwart the issue of structural instability, the VAR framework was utilized choosing the lag length according to AIC.

So, I will fit the VAR model on training set and then used the fitted model to forecast the next 15 observations. These forecasts will be compared against the actuals present in test data. I have taken the maximum lag (15) to identify the required lags for VAR model.

```
import statsmodels.tsa.api as smt
from statsmodels.tsa.api import VAR
mod = smt.VAR(transform_data)
res = mod.fit(maxlags=15, ic='aic')
print(res.summary())
```

```
Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Thu, 14, Nov, 2019
Time:                     23:14:37
--------------------------------------------------------------------
No. of Equations:         6.00000    BIC:                   -1.19025
Nobs:                     5111.00    HQIC:                  -1.25510
Log likelihood:          -40138.5    FPE:                    0.275259
AIC:                     -1.29004    Det(Omega_mle):         0.271096
--------------------------------------------------------------------
```

```
Correlation matrix of residuals
               Gold    Silver       Oil       USD  Interest     Stock
Gold       1.000000  0.387653  0.201638 -0.200735 -0.009341  0.161670
Silver     0.387653  1.000000  0.194438 -0.140164  0.004120  0.141765
Oil        0.201638  0.194438  1.000000 -0.199088  0.107060  0.280690
USD       -0.200735 -0.140164 -0.199088  1.000000  0.160394 -0.162544
Interest  -0.009341  0.004120  0.107060  0.160394  1.000000  0.209950
Stock      0.161670  0.141765  0.280690 -0.162544  0.209950  1.000000
```

Durbin-Watson Statistic

Durbin-Watson Statistic is performed to check for Serial Correlation of Residuals (Errors). If there is any correlation left in the residuals, then, there is some pattern in the time series that is still left to be explained by the model. In that case, the typical course of action is to either increase the order of the model or induce more predictors into the system or look for a different algorithm to model the time series. So, checking for serial correlation is to ensure that the model is sufficiently able to explain the variances and patterns in the time series.

> *The Durbin-Watson statistic will always have a value between 0 and 4. A value of 2.0 means that there is no autocorrelation detected in the sample. Values from 0 to less than 2 indicate positive autocorrelation and values from 2 to 4 indicate negative autocorrelation. A rule of thumb is that test statistic values in the range of 1.5 to 2.5 are relatively normal. Any value outside this range could be a cause for concern.*

A stock price displaying positive autocorrelation would indicate that the price yesterday has a positive correlation on the price today — so if the stock fell yesterday, it is also likely that it falls today. A stock that has a negative autocorrelation, on the other hand, has a negative influence on itself over time — so that if it fell yesterday, there is a greater likelihood it will rise today.

```
from statsmodels.stats.stattools import durbin_watson
out = durbin_watson(res.resid)

for col, val in zip(transform_data.columns, out):
    print((col), ':', round(val, 2))
```

```
Gold : 2.0
Silver : 2.0
Oil : 2.0
USD : 2.0
Interest : 2.0
Stock : 2.0
```

There is no auto-correlation (2.0) exist; so, we can proceed with the forecast.

Prediction

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

```
pred= res.forecast(transform_data.values[-3:], 15)
pred_df = pd.DataFrame(pred, index=dataset.index[-15:], columns= dataset.columns )
pred_df
```

Date	Gold	Silver	Oil	USD	Interest	Stock
2018-04-11	-0.002624	0.001585	-0.030177	0.036694	0.002666	11.756011
2018-04-12	-0.011787	-0.001876	0.013445	0.003336	0.000837	3.269195
2018-04-13	-0.001740	0.001800	0.003695	-0.002411	-0.001249	0.492346
2018-04-16	-0.001624	0.002012	0.005875	-0.002270	-0.000819	1.040094
2018-04-17	-0.000753	0.002307	0.007203	-0.001844	-0.000702	1.121373
2018-04-18	-0.000946	0.002386	0.007628	-0.002042	-0.000729	1.131713
2018-04-19	-0.000980	0.002357	0.007529	-0.002059	-0.000736	1.129534
2018-04-20	-0.000974	0.002352	0.007488	-0.002042	-0.000734	1.125906
2018-04-23	-0.000971	0.002353	0.007493	-0.002042	-0.000734	1.126257
2018-04-24	-0.000972	0.002354	0.007497	-0.002043	-0.000734	1.126638
2018-04-25	-0.000972	0.002354	0.007496	-0.002043	-0.000734	1.126575
2018-04-26	-0.000972	0.002354	0.007496	-0.002043	-0.000734	1.126550
2018-04-27	-0.000972	0.002354	0.007496	-0.002043	-0.000734	1.126556
2018-04-30	-0.000972	0.002354	0.007496	-0.002043	-0.000734	1.126557
2018-05-01	-0.000972	0.002354	0.007496	-0.002043	-0.000734	1.126557

Invert the transformation to get the real forecast

The forecasts are generated but it is on the scale of the training data used by the model. So, to bring it back up to its original scale, we need to de-difference it.

The way to convert the differencing is to add these differences consecutively to the base number. An easy way to do it is to first determine the cumulative sum at index and then add it to the base number.

This process can be reversed by adding the observation at the prior time step to the difference value. inverted(ts) = differenced(ts) + observation(ts-1)

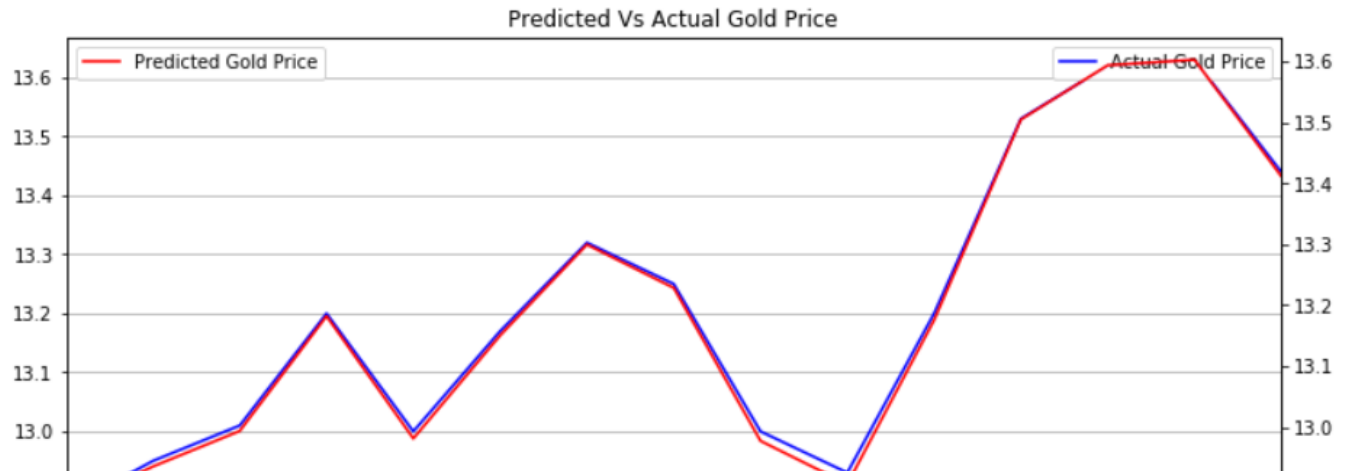
```
pred_inverse = pred_df.cumsum() # reversing difference
f = pred_inverse + X_test # inverse the differece values
print(f)
```

Date	Gold	Silver	Oil	USD	Interest	Stock
2018-04-11	12.867376	17.206585	65.559823	89.316694	2.779666	12521.145661
2018-04-12	12.935589	17.204710	66.733268	89.250031	2.795503	12574.005676
2018-04-13	12.993849	17.206510	67.166962	89.497620	2.840254	12595.737282
2018-04-16	13.182225	17.208522	67.232837	89.535350	2.865435	12617.958035
2018-04-17	12.981472	17.210829	66.420040	89.148507	2.838733	12698.068669
2018-04-18	13.150526	17.213215	66.687668	89.256465	2.840004	12755.940612
2018-04-19	13.299545	17.215572	68.765197	89.384406	2.892268	12734.550616
2018-04-20	13.228572	17.217924	68.282685	89.682364	2.915533	12691.986091
2018-04-23	12.977601	17.220277	68.250178	90.165322	2.972799	12640.262738
2018-04-24	12.906629	17.222631	68.987675	90.718280	2.972065	12683.248756
2018-04-25	13.175658	17.224985	67.795172	90.561237	3.028331	12515.875331
2018-04-26	13.504686	17.227338	68.082668	91.024194	2.985597	12565.222581
2018-04-27	13.593715	17.229692	68.270164	91.402151	2.962863	12594.368667
2018-04-30	13.602743	17.232046	68.217660	91.360109	2.949129	12643.174914
2018-05-01	13.411771	17.234399	68.635157	91.633066	2.959395	12519.192091

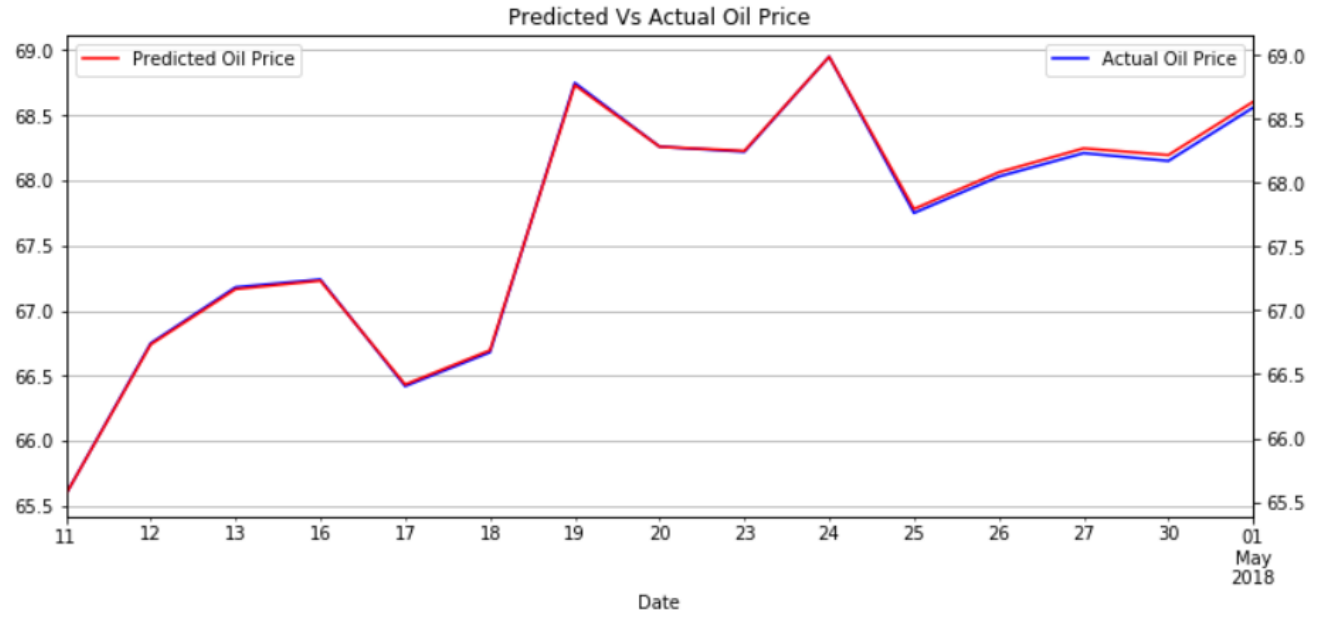
Plot of Forecast vs Actuals

The forecasts are back to the original scale. Let's plot the forecasts against the actuals from test data.

```
plt.figure(figsize= (12,5))
plt.xlabel('Date')

ax1 = X_test.Gold.plot(color='blue', grid = True, label = 'Actual Gold Price')
ax2 = f.Gold.plot(color='red', grid = True, secondary_y=True, label = 'Predicted Gold Price')

ax1.legend(loc=1)
ax2.legend(loc=2)
plt.title('Predicted Vs Actual Gold Price')
plt.show()
```




```
plt.figure(figsize= (12,5))
plt.xlabel('Date')

ax1 = X_test.Oil.plot(color='blue', grid = True, label = 'Actual Oil Price')
ax2 = f.Oil.plot(color='red', grid = True, secondary_y=True, label = 'Predicted Oil Price')

ax1.legend(loc=1)
ax2.legend(loc=2)
plt.title('Predicted Vs Actual Oil Price')
plt.show()
```



Evaluate the Forecasts

To evaluate the forecasts, let's compute a comprehensive set of metrics, namely, the MAPE, ME, MAE, MPE, RMSE, corr and minmax.

```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_error
from math import sqrt

forecast_errors = [X_test.Gold[i]-f.Gold[i] for i in range(len(X_test.Gold))]
bias = sum(forecast_errors) * 1.0/len(X_test.Gold)
print('Bias: %f' % bias)

mae = mean_absolute_error(X_test.Gold, f.Gold)
```

```
print('MAE: %f' % mae)

mse = mean_squared_error(X_test.Gold, f.Gold)
print('MSE: %f' % mse)

rmse = sqrt(mse)
print('RMSE: %f' % rmse)
```

```
Bias: 0.020536
MAE: 0.020536
MSE: 0.000460
RMSE: 0.021453
```

```
forecast_errors = [X_test.Oil[i]-f.Oil[i] for i in range(len(X_test.Oil))]
bias = sum(forecast_errors) * 1.0/len(X_test.Oil)
print('Bias: %f' % bias)

mae = mean_absolute_error(X_test.Oil, f.Oil)
print('MAE: %f' % mae)

mse = mean_squared_error(X_test.Oil, f.Oil)
print('MSE: %f' % mse)

rmse = sqrt(mse)
print('RMSE: %f' % rmse)
```

```
Bias: -0.023144
MAE: 0.032092
MSE: 0.001547
RMSE: 0.039334
```

Mean absolute error tells us how big of an error we can expect from the forecast on average. Our error rates are quite low here indicating we have the right fit of the model.

Summary

The VAR model is a popular tool for the purpose of predicting joint dynamics of multiple time series based on linear functions of past observations. More analysis e.g. impulse response (IRF) and forecast error variance decomposition (FEVD) can also be done along-with VAR to for assessing the impacts of shock from one asset on another to assess the impacts of shock from one asset on another. However, I will keep this simple here for easy understanding. However, in real-life scenario, we should do multiple models with

different approach to do the comparative analysis before zeroed down on one or a hybrid model.

I can be connected here.

Machine Learning Time Series Forecasting Autoregressive Data Science Analytics