Open in app

Sarit Maitra

1.4K Followers About

STOCHASTIC TIME-SERIES MODELING

How to Model & Predict Future Time-Steps Using ARIMA Statistical Model

Time-Series modeling using Natural Gas data

Sarit Maitra Sep 18, 2020 · 6 min read ★

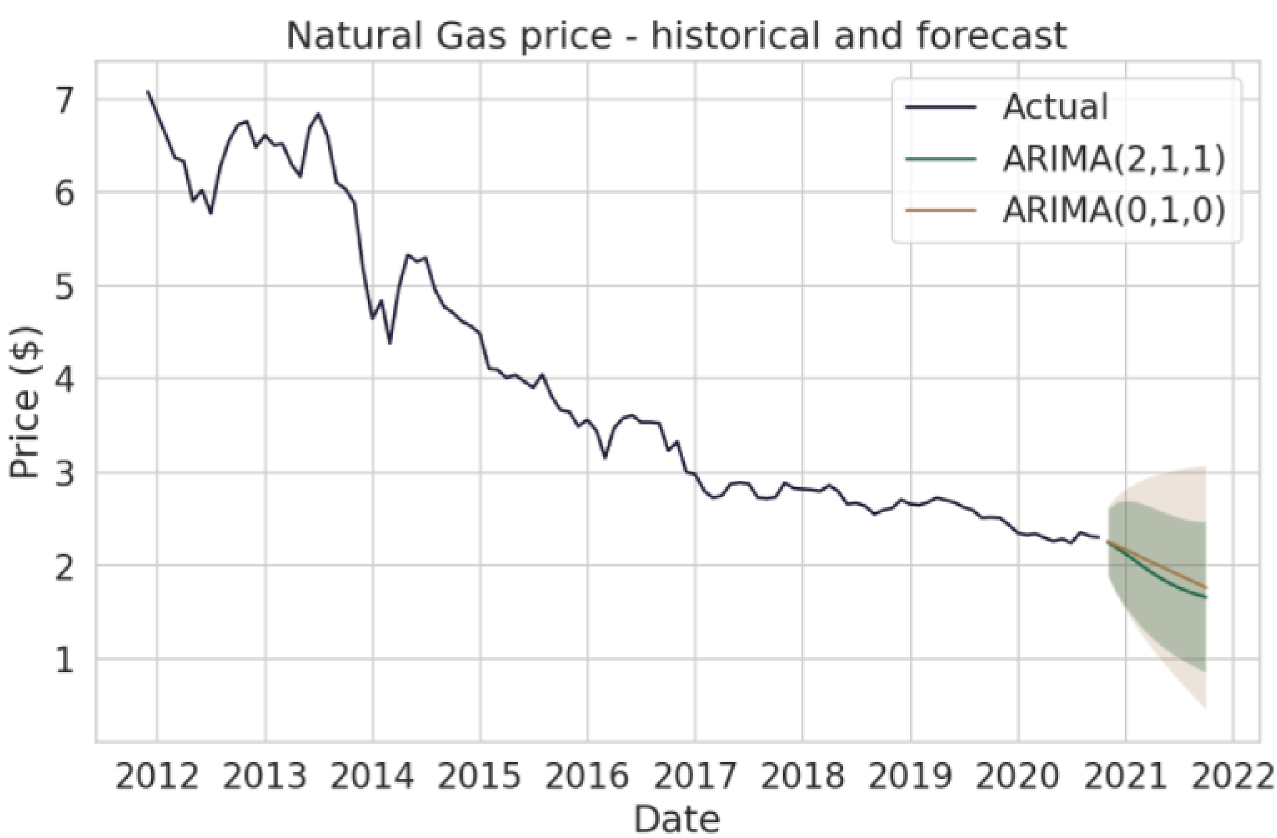


Image by Author

case here is to forecast future time steps using the univariate data. The time series is stochastic/ random walk price series.

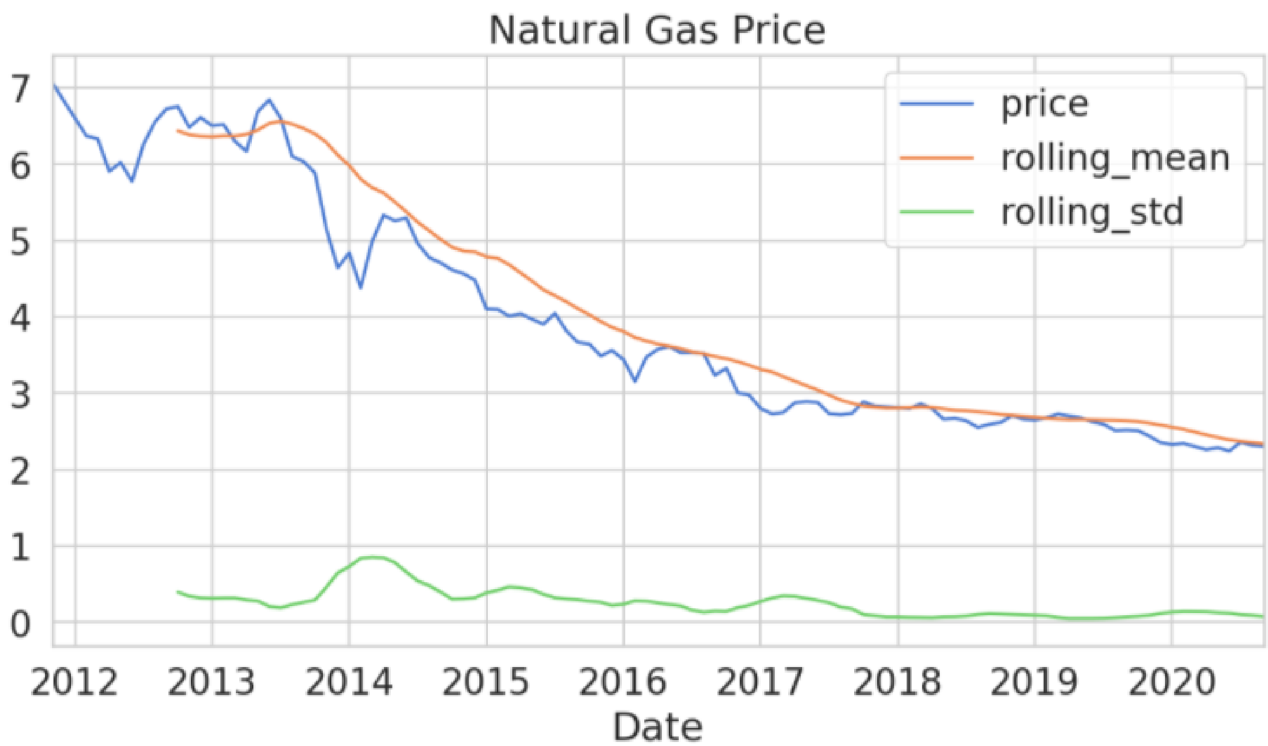
Let us load the check the data we have and resample to monthly frequency for the ease of computation.

```
print("....Data Loading...."); print();
print('\033[4mHenry Hub Natural Gas Price\033[0m');
data = web.DataReader('NNJ24.NYM', data_source = 'yahoo', start =
'2000-01-01');
data.rename(columns={'Close': 'price'}, inplace=True);
df = data.resample('M').last(); df = DataFrame(df.price.copy());
df;
```

```
....Data Loading....

Henry Hub Natural Gas Price
```

	price
Date	
2011-11-30	7.074
2011-12-31	6.834
2012-01-31	6.602
2012-02-29	6.369
2012-03-31	6.330
...	...
2020-05-31	2.285
2020-06-30	2.243
2020-07-31	2.356
2020-08-31	2.317
2020-09-30	2.305

```
window = 12
df['rolling_mean'] = df.price.rolling(window=window).mean();
df['rolling_std'] = df.price.rolling(window=window).std();
df.plot(title='Natural Gas Price', figsize = (10,5)); plt.show();
```



Non-linear pattern can be observed in the 12-month moving average and that the rolling standard deviation which are on decreasing trend. We will use multiplicative model. Multiplicative model assumes that as the data increase, so does the seasonal pattern. Most time series plots exhibit such a pattern. In this model, the trend and seasonal components are multiplied and then added to the error component.

Decomposition

Decomposition will be performed by breaking down the series into multiple components. Objective is to have deeper understanding of the series. It provides insight in terms of modeling complexity and which approaches to follow in order to accurately capture each of the components.

The components are:

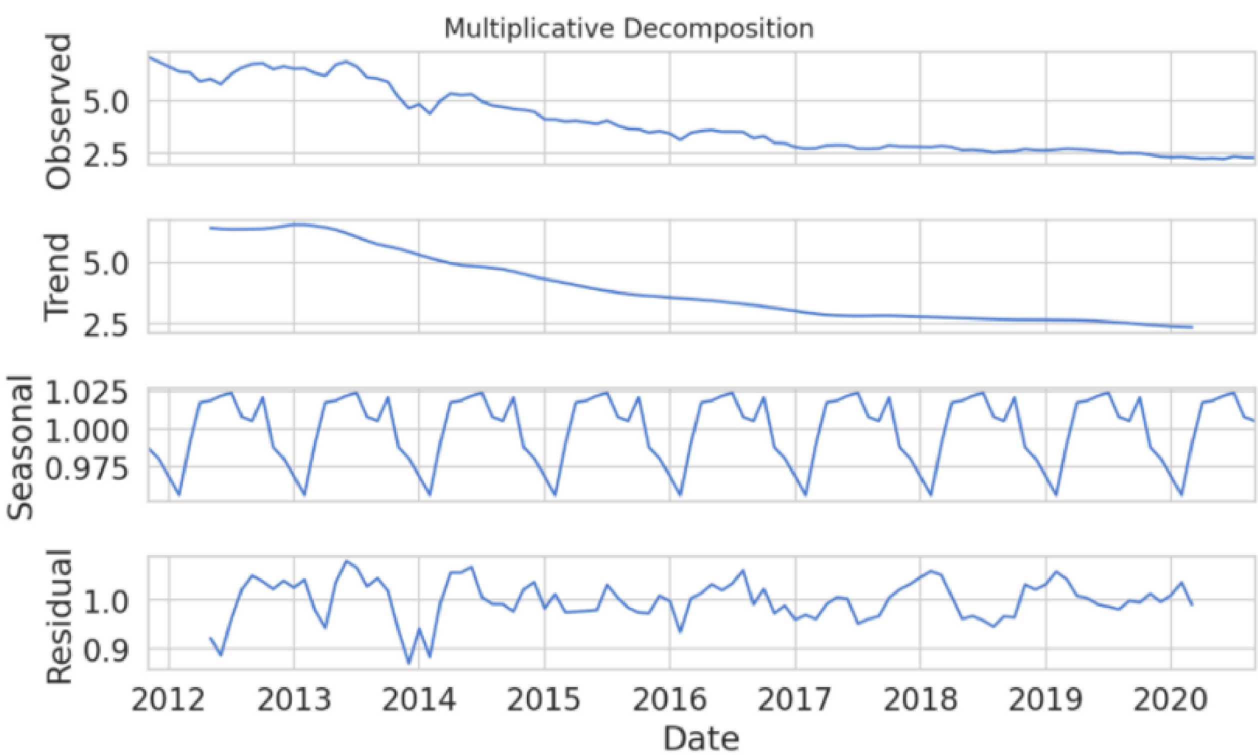
2. trend is associated with the slope (increasing/decreasing) of the series.

3. seasonality which is the deviations from the mean caused by repeating short-term cycles and

4. noise is the random variation in the series

In multiplicative model, all the abovecomponents are multiplied with each other like y(t) = level * trend * seasonality * noise to develop a non-linear model. If we do not want to work with the multiplicative model, we can apply transformations e.g. log transformation to make the trend/seasonality linear.

```
decomp = seasonal_decompose(df.price, model='multiplicative')
rcParams['figure.figsize'] = 10, 6
decomp.plot().suptitle('Multiplicative Decomposition', fontsize=14);
```



It looks like the variance in the residuals is slightly higher in the 1st half of the data set. In case of additive model, the residuals display an increasing pattern over time.

Stationarity test

- The Augmented Dickey-Fuller (ADF) test

Plots of the (partial) auto-correlation function (PACF, ACF)

```
print('Results Dickey-Fuller Test:')
test = adfuller(df.price, autolag = 'AIC')
output = pd.Series(test[0:4], index = ['Test Statistic', 'p-value','# of Lags Used', '# of Observations Used'])
for key, value in test[4].items():
    output[f'Critical Value ({key})'] = value

print(output)
```

```
Results Dickey-Fuller Test:
Test Statistic              -1.784942
p-value                      0.387963
# of Lags Used               0.000000
# of Observations Used     106.000000
Critical Value (1%)         -3.493602
Critical Value (5%)         -2.889217
Critical Value (10%)        -2.581533
dtype: float64
```

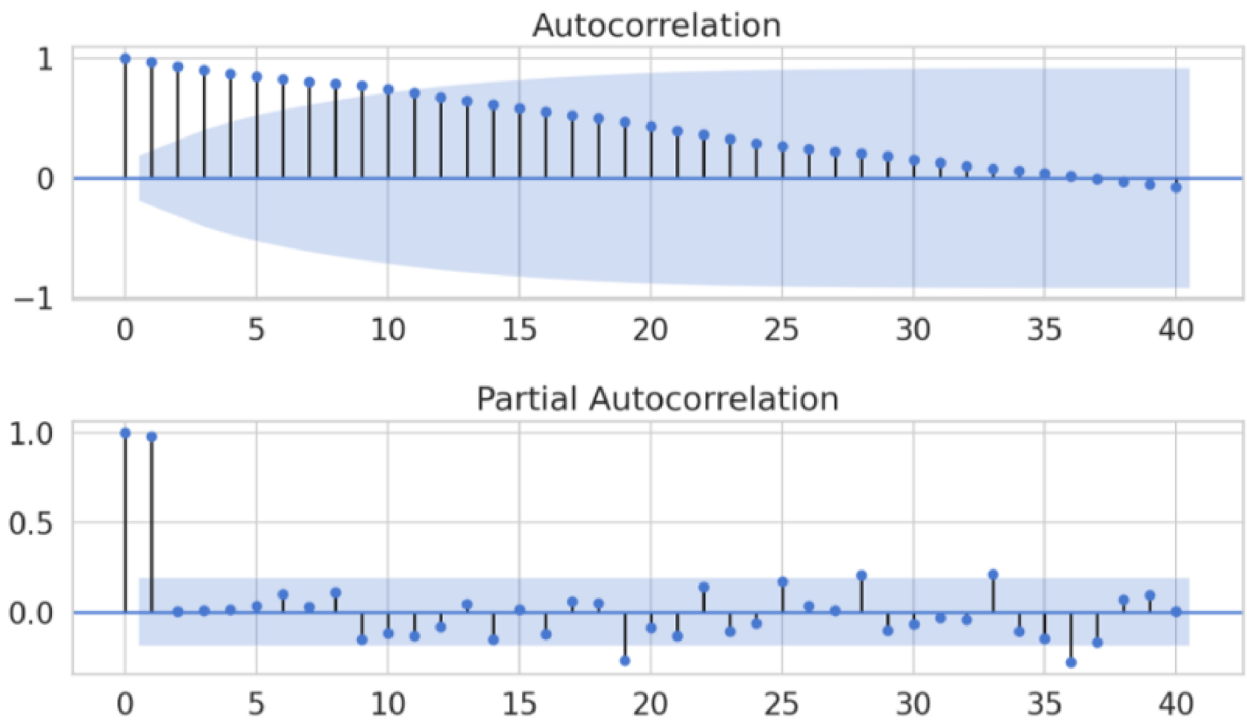
ADF test

```
def kpss_test(x, h0_type='c'):
  indices = ['Test Statistic', 'p-value', '# of Lags']
  kpss_test = kpss(x, regression=h0_type)
  results = pd.Series(kpss_test[0:3], index=indices)
  for key, value in kpss_test[3].items():
    results[f'Critical Value ({key})'] = value
    return results

kpss_test(df.price)
```

```
p-value is smaller than the indicated p-value

Test Statistic            0.810465
p-value                   0.010000
# of Lags                13.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
dtype: float64
```

KPSS test

```
lags = 40
sig_level = 0.05
```



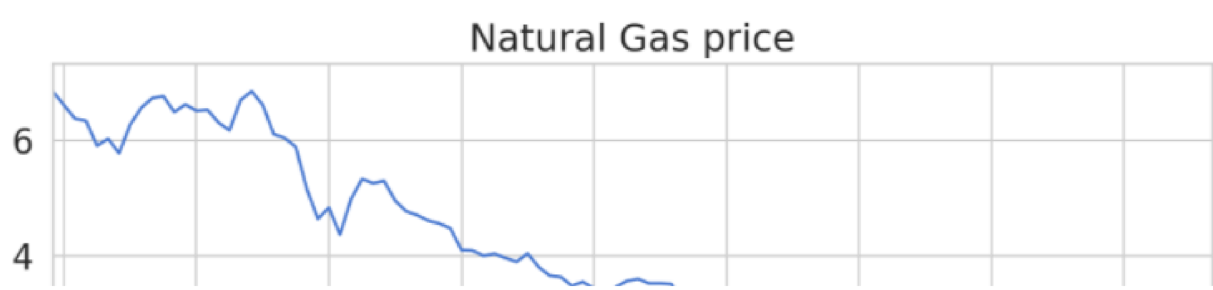
ACF/PACF on raw data

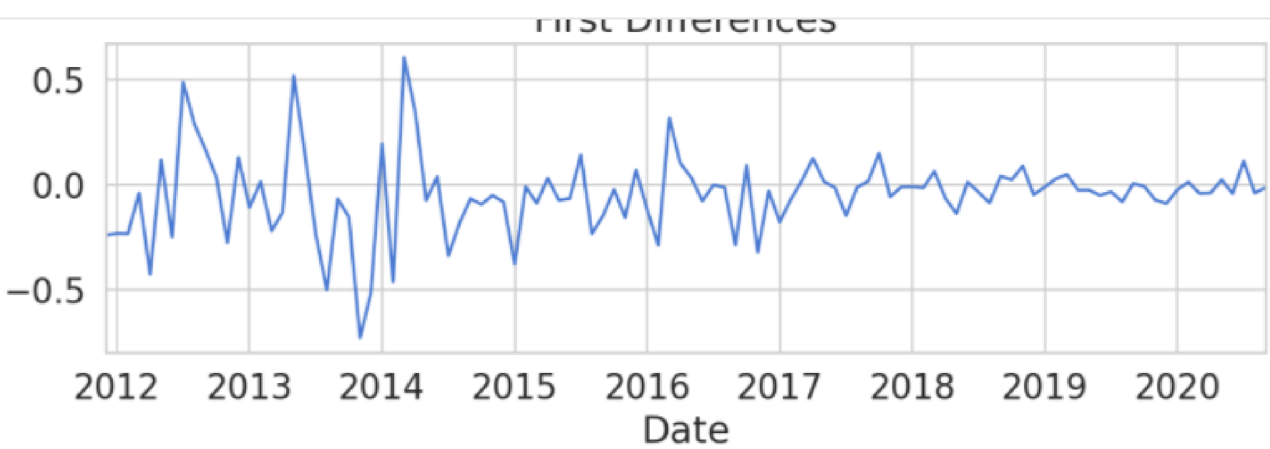
Significant auto-correlations can be seen in ACF plot. There are also some significant auto-correlations at lags 1 and 19 in the PACF plot.

Let us correct the stationarity by differencing i.e. taking the 1st orxer difference between the current observation and a lagged value.

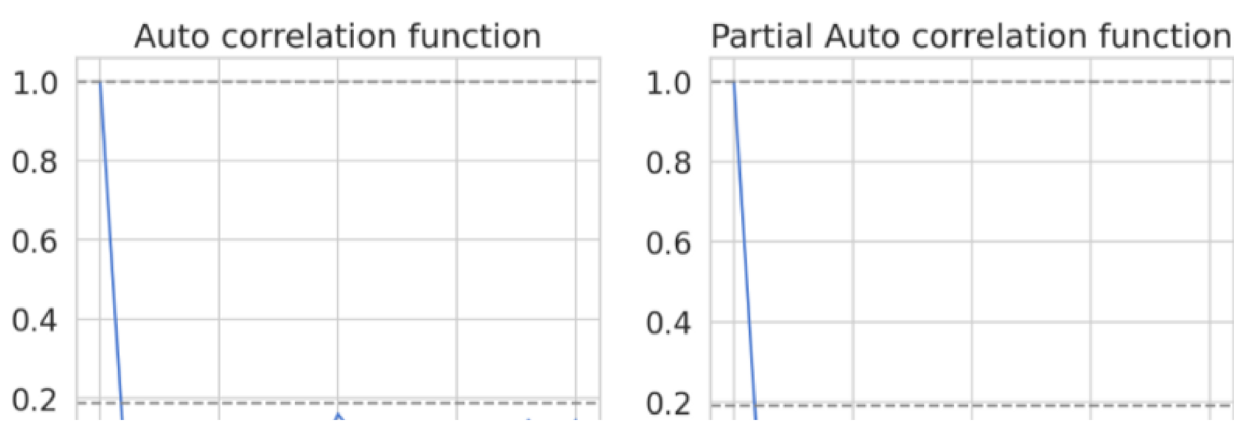
Differenced series

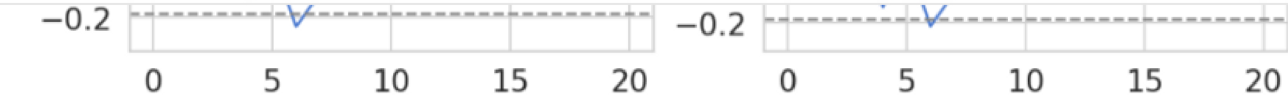
```
price_diff = df.price.diff().dropna()
fig, ax = plt.subplots(2, sharex=True)
df.price.plot(title = "Natural Gas price", ax=ax[0])
price_diff.plot(ax=ax[1], title='First Differences')
plt.show()
```






```
diff = df.price.diff().dropna();
lag_acf = acf(diff, nlags=20); lag_pacf = pacf(diff, nlags=20, method
= 'ols');

# plot acf
plt.subplot(121); plt.plot(lag_acf);
plt.axhline(y=1, linestyle='--', color = 'gray');
plt.axhline(y=-1.96/np.sqrt(len(diff)), linestyle='--', color =
'gray');
plt.axhline(y=1.96/np.sqrt(len(diff)), linestyle='--', color =
'gray'); plt.title('Auto correlation function');

# plot pacf
plt.subplot(122); plt.plot(lag_pacf);
plt.axhline(y=1, linestyle='--', color = 'gray');
plt.axhline(y=-1.96/np.sqrt(len(diff)), linestyle='--', color =
'gray')'
plt.axhline(y=1.96/np.sqrt(len(diff)), linestyle='--', color =
'gray');
plt.title('Partial Auto correlation function'); plt.tight_layout();
```

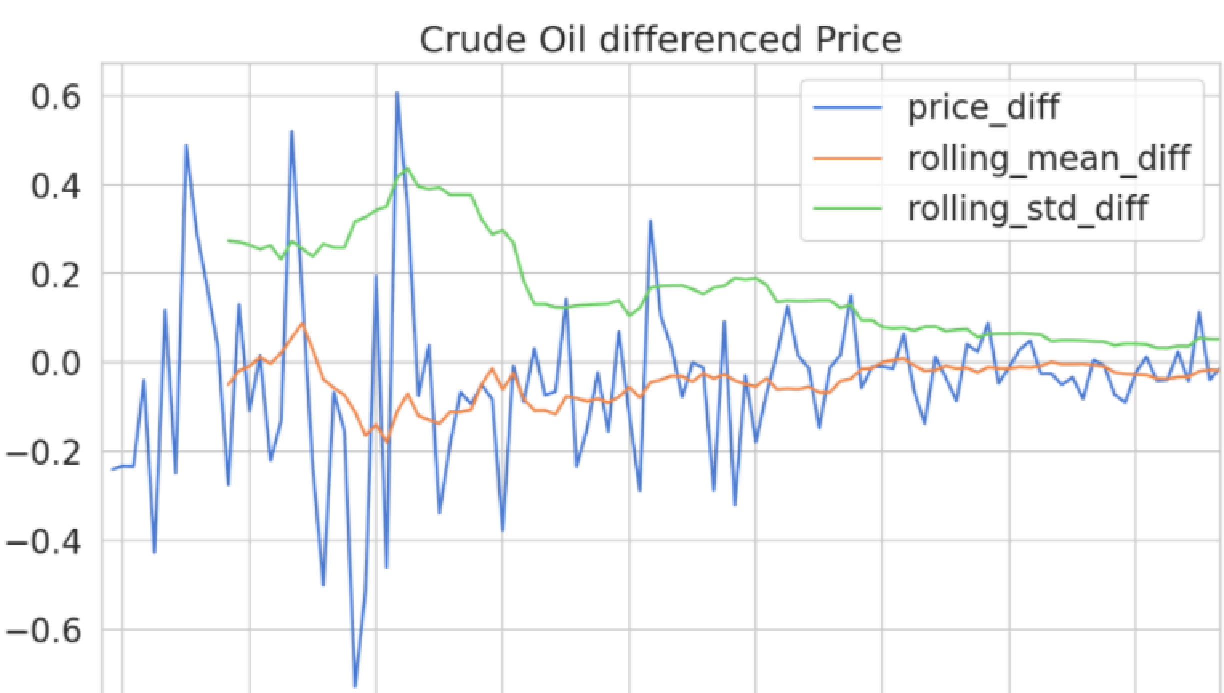





```
print(adf_test(price_diff).dropna()); print(); print(kpss_test(price_diff).dropna());
```

```
Test Statistic                -9.521530e+00
p-value                        3.055624e-16
# of Lags Used                 0.000000e+00
# of Observations Used         1.050000e+02
Critical Value (1%)           -3.494220e+00
Critical Value (5%)           -2.889485e+00
Critical Value (10%)          -2.581676e+00
dtype: float64

Test Statistic                 0.288121
p-value                        0.100000
# of Lags                     13.000000
Critical Value (10%)           0.347000
Critical Value (5%)            0.463000
Critical Value (2.5%)          0.574000
Critical Value (1%)            0.739000
```

```
rcParams['figure.figsize'] = 10, 6
columns = ['price_diff', 'rolling_mean_diff', 'rolling_std_diff']
df['price_diff'] = df['price'].diff()
df['rolling_mean_diff'] = df['price'].diff().rolling(window=window).mean()
df['rolling_std_diff'] = df['price'].diff().rolling(window=window).std()
df[columns].plot(title='Crude Oil differenced Price')
plt.show()
```



Here, we can see that differenced series making the trend linear. The series moves with constant mean or less constant variance. At least there is no visible trend. Let us fit ARIMA model.

ARIMA model

```
from statsmodels.tsa.arima_model import ARIMA
arima = ARIMA(df.price, order=(2, 1, 1)).fit(disp=0)
arima.summary()
```

ARIMA Model Results

Dep. Variable:	D.price	**No. Observations:**	106
Model:	ARIMA(2, 1, 1)	**Log Likelihood**	24.940
Method:	css-mle	**S.D. of innovations**	0.191
Date:	Fri, 18 Sep 2020	**AIC**	-39.880
Time:	14:36:09	**BIC**	-26.563
Sample:	12-31-2011 - 09-30-2020	**HQIC**	-34.482

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	-0.0447	0.012	-3.601	0.000	-0.069	-0.020
ar.L1.D.price	0.9411	0.142	6.634	0.000	0.663	1.219
ar.L2.D.price	-0.1209	0.098	-1.233	0.220	-0.313	0.071
ma.L1.D.price	-0.8844	0.108	-8.217	0.000	-1.095	-0.673

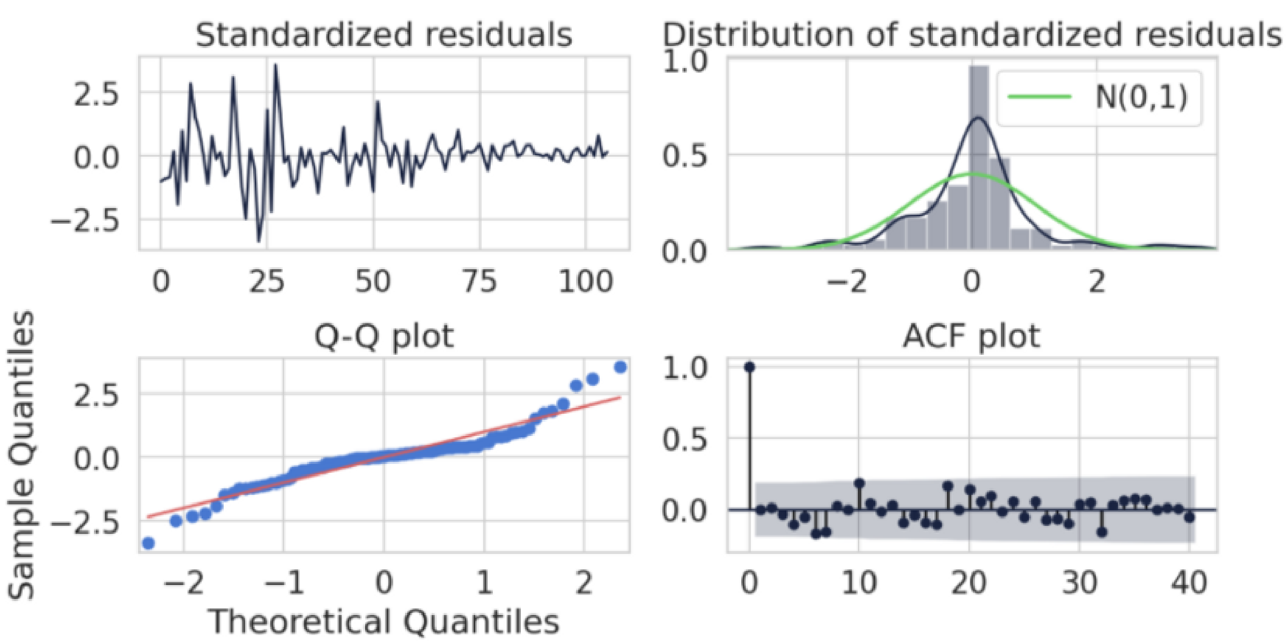
Roots

	Real	**Imaginary**	**Modulus**	**Frequency**
AR.1	1.2697	+0.0000j	1.2697	0.0000
AR.2	6.5171	+0.0000j	6.5171	0.0000
MA.1	1.1306	+0.0000j	1.1306	0.0000

The AR & MA (p & q) orders can be taken based on acf/pacf plots as shown above.

```
def arima_diagnostics(resids, n_lags=40):
  fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2);
```

```
  sns.lineplot(x=np.arange(len(resids)), y=resids, ax=ax1);
  ax1.set_title('Standardized residuals');
  x_lim = (-1.96 * 2, 1.96 * 2);  r_range = np.linspace(x_lim[0],
x_lim[1]);  norm_pdf = scs.norm.pdf(r_range);
  sns.distplot(resids_nonmissing, hist=True, kde=True,   norm_hist
=True, ax=ax2);
  ax2.plot(r_range, norm_pdf, 'g', lw=2, label='N(0,1)');
  ax2.set_title('Distribution of standardized residuals');
  ax2.set_xlim(x_lim); ax2.legend();

  # Q-Q plot
  qq = sm.qqplot(resids_nonmissing, line='s', ax=ax3);
  ax3.set_title('Q-Q plot');

  # ACF plot
  plot_acf(resids, ax=ax4, lags=n_lags, alpha=0.05);
  ax4.set_title('ACF plot');
  return fig

arima_diagnostics(arima.resid, 40); plt.tight_layout();
```

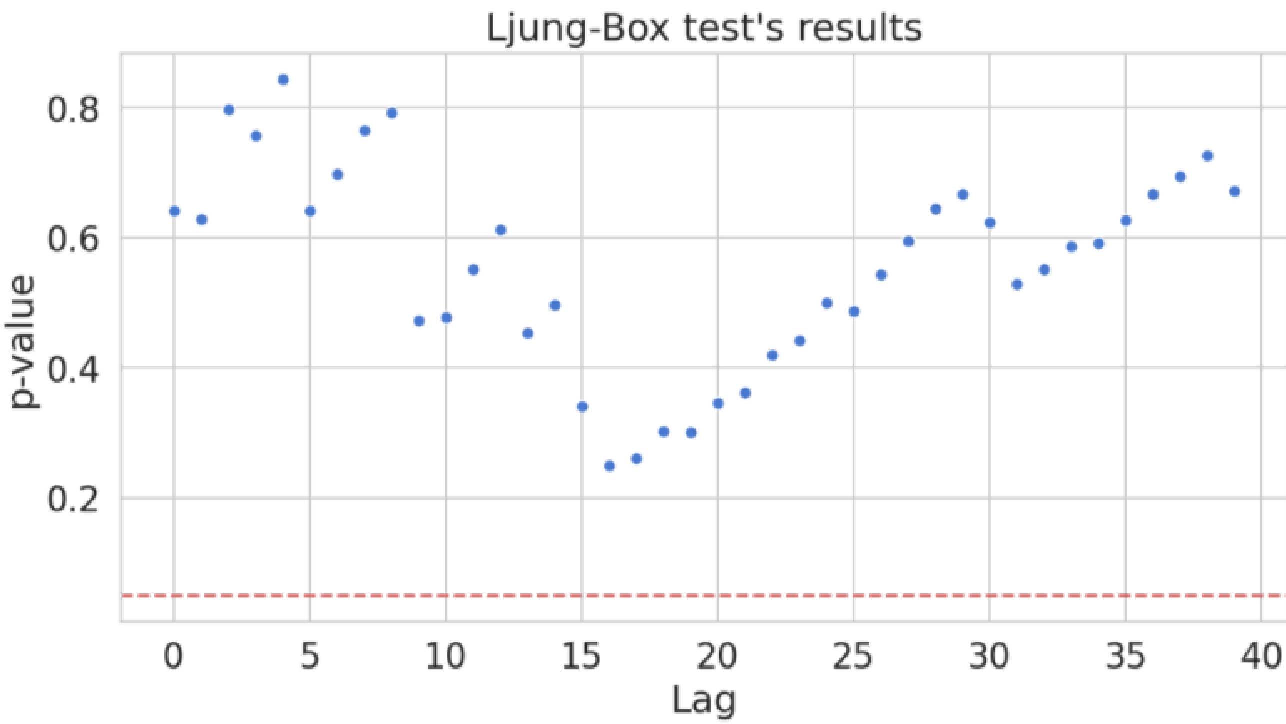


Looking at the diagnostics plots, the residuals look like normal distribution. The average of the residuals is close to 0 (-0.05), and ACF plot says that the residuals are not correlated.

Ljung-Box test

Open in app

```
sns.scatterplot(x=range(len(ljung_box_results[1])), y = results[1],
ax=ax); ax.axhline(0.05, ls='--', c='r');
ax.set(title="Ljung-Box test's results", xlabel='Lag', ylabel='p-
value'); plt.show();
```



Ljung-Box test satisfies the goodness-of-fit by showing no significant autocorrelation for any of the selected lags.

Prediction based on ARIMA model

```
forecast = int(12)
arima_pred, std, ci = (arima.forecast(forecast))
arima_pred = DataFrame(arima_pred)
d = DataFrame(df.price.tail(len(arima_pred))); d.reset_index(inplace
= True)
d = d.append(DataFrame({'Date': pd.date_range(start =
d.Date.iloc[-1], periods = (len(d)+1), freq = 'm', closed =
'right')}))
d = d.tail(forecast); d.set_index('Date', inplace = True)
arima_pred.index = d.index
arima_pred.rename(columns = {0: 'arima_fcast'}, inplace=True)
```

Open in app

```
ci.index = arima_pred.index;
ARIMA = concat([arima_pred, ci], axis=1); ARIMA
```

Date	arima_fcast	lower95	upper95
2020-10-31	2.251660	1.891692	2.611627
2020-11-30	2.191202	1.700045	2.682359
2020-12-31	2.125722	1.549243	2.702200
2021-01-31	2.057690	1.422696	2.692685
2021-02-28	1.989711	1.314025	2.665396
2021-03-31	1.924266	1.219842	2.628691
2021-04-30	1.863495	1.137964	2.589026
2021-05-31	1.809005	1.066736	2.551275
2021-06-30	1.761747	1.004585	2.518909
2021-07-31	1.721953	0.949676	2.494230
2021-08-31	1.689150	0.899670	2.478629
2021-09-30	1.662228	0.851702	2.472754

Auto-ARIMA

To re-validate the manual selection of ARIMA parameters, let us run through auto-arima.

```
model = pm.auto_arima(df.price, error_action='ignore', suppress_warnings=True,
                      seasonal=False)
model.summary()
```

Statespace Model Results

Dep. Variable:	y	**No. Observations:**	107
Model:	SARIMAX(0, 1, 0)	**Log Likelihood**	23.828
Date:	Fri, 18 Sep 2020	**AIC**	-43.656

Covariance Type: opg

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	-0.0450	0.019	-2.397	0.017	-0.082	-0.008
sigma2	0.0373	0.003	11.087	0.000	0.031	0.044

Ljung-Box (Q):	41.57	**Jarque-Bera (JB):**	30.75
Prob(Q):	0.40	**Prob(JB):**	0.00
Heteroskedasticity (H):	0.04	**Skew:**	0.01
Prob(H) (two-sided):	0.00	**Kurtosis:**	5.64

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Prediction (auto-arima)

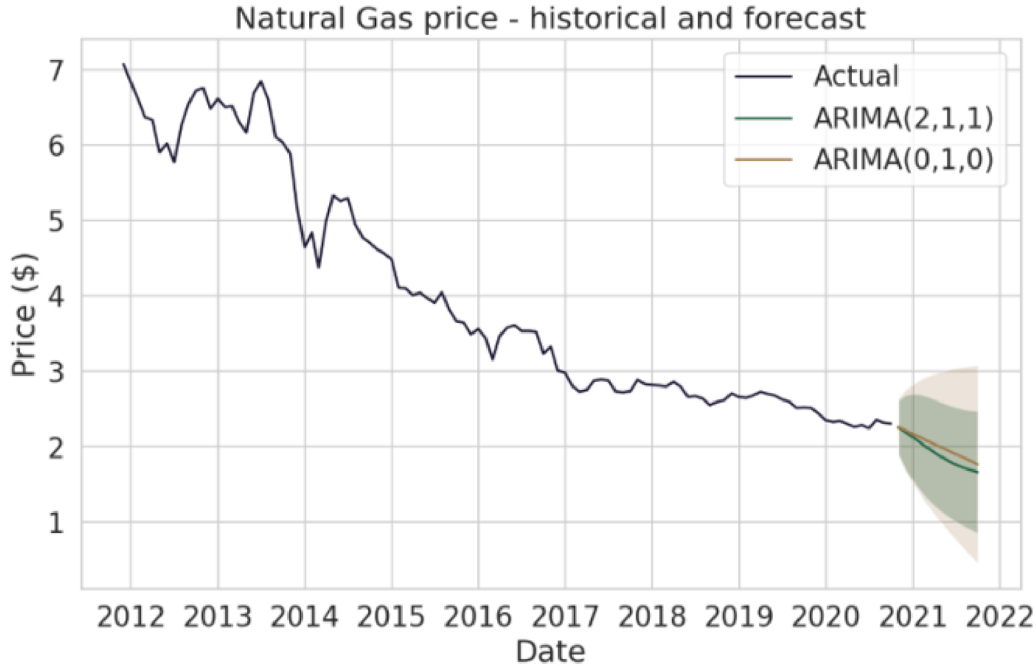
```
auto_arima_pred = model.predict(n_periods=forecast, return_conf_int=True,alpha=0.05)
auto_arima_pred = [DataFrame(auto_arima_pred[0],columns=['prediction']),DataFrame(auto_arima_pred[1],
                                                                                      columns=['ci_lower', 'ci_upper'])]
auto_arima_pred = concat(auto_arima_pred,axis=1).set_index(ARIMA.index)
auto_arima_pred
```

Date	prediction	ci_lower	ci_upper
2020-10-31	2.260009	1.881231	2.638788
2020-11-30	2.215019	1.679345	2.750693
2020-12-31	2.170028	1.513965	2.826092
2021-01-31	2.125038	1.367481	2.882595
2021-02-28	2.080047	1.233073	2.927022
2021-03-31	2.035057	1.107243	2.962871
2021-04-30	1.990066	0.987912	2.992220
2021-05-31	1.945076	0.873728	3.016423
2021-06-30	1.900085	0.763750	3.036420
2021-07-31	1.855094	0.657292	3.052897
2021-08-31	1.810104	0.553838	3.066370
2021-09-30	1.765113	0.452986	3.077240

Visualization

```
plt.set_cmap('cubehelix'); sns.set_palette('cubehelix')
COLORS = [plt.cm.cubehelix(x) for x in [0.1, 0.3, 0.5, 0.7]]; fig, ax = plt.subplots(1)
ax = sns.lineplot(data=df.price, color=COLORS[0], label='Actual')
ax.plot(ARIMA.arima_fcast, c=COLORS[1], label='ARIMA(2,1,1)')
ax.fill_between(ARIMA.index, ARIMA.lower95, ARIMA.upper95, alpha=0.3, facecolor=COLORS[1])
ax.plot(auto_arima_pred.prediction, c=COLORS[2], label='ARIMA(0,1,0)')
ax.fill_between(auto_arima_pred.index, auto_arima_pred.ci_lower, auto_arima_pred.ci_upper,
```

```
<matplotlib.legend.Legend at 0x7fe4dbafa828>
<Figure size 720x432 with 0 Axes>
```



Conclusion

The approach applied here to forecast the future trends of price movements based on its past behavior using stochastic time-series modeling . ARIMA model is designated by ARIMA (p, d, q), where 'p' is the auto regressive process order, 'd' corresponds to stationary data order and 'q' denotes the moving average process order. Validity of the developed ARIMA models can be testified via statistical parameters such as RMSE, MAPE, AIC, AICc and BIC which was not shown here. ARIMA with GARCH volatility model can be tried too to capture the volatility in the series.

Connect me 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.

Timeseries Forecasting | Arima | Predictive Modeling | Stochastic Modelling