# ASQ02: Lecture Summary

## Overview

This document summarises the ASQ-02 lecture on building time-series models. This lecture introduces various time-series modelling techniques and their implementation in Python. After this lecture, you should be able to:

- Understand various time-series models
- Build a time-series model in Python
- Interpret the results of the model

The lecture covers the following topics:

- Anatomy of a time series process
- Modelling time series using decomposition
    - Method I: *statsmodels* library
    - Method II: *fbprophet* library
- Testing for stationarity
- Modelling time series using
    - Method III: Exponential smoothing
    - Method IV: ARIMA
- A glance at modelling volatility using ARCH/GARCH

## Anatomy of a time series process

A time-series process can be considered as a combination of the following components:

- Systematic - Features that can be explained using a model.
- Non-systematic - Features that are random and cannot be modelled.

The systematic component of a time-series process can be further divided as follows:

- Level: The average value of the process
- Trend: The direction and rate of change of the process
- Seasonality: Deviations in the process due to recurring short-term cycles
- Noise: The random variation observed in the process.

A time-series can be modelled using the components mentioned above in 2 different ways:

- Additive model: This process is used when:
    - The process change remains constant over time.
    - It shows linear seasonality.

The process can be represented as -

X(t) = Level + Trend + Seasonality + Noise

- Multiplicative model: This process is used when:
    - The process changes overtime
    - The process shows non-linear seasonality.

The process can be represented as -

X(t) = Level x Trend x Seasonality x Noise

**Modelling time series using decomposition**

There are several available libraries (statsmodels,fbprophet, scikit-learn, PyFlux Fecon236, PM-Prophet) in Python to develop sophisticated time series models for forecasting. *statsmodels* and *fbprophet* are used in this lecture.

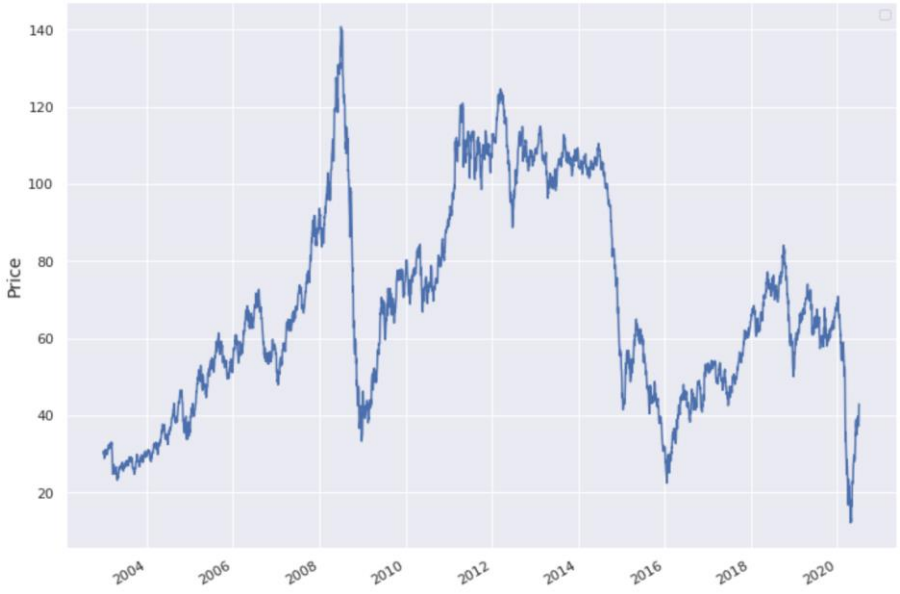
The following datasets are used as an example in the lecture:

1. Daily historical prices of crude oil (from 2003 to 2020) - OPEC benchmark prices
2. Daily historical soybean (from 2000 to 2020).

**Method-1 Using *statsmodel* library**

The seasonal decomposition is performed using *seasonal_decompose* in the statsmodel package.

*The following snapshot shows the OPEC crude oil benchmark prices.*

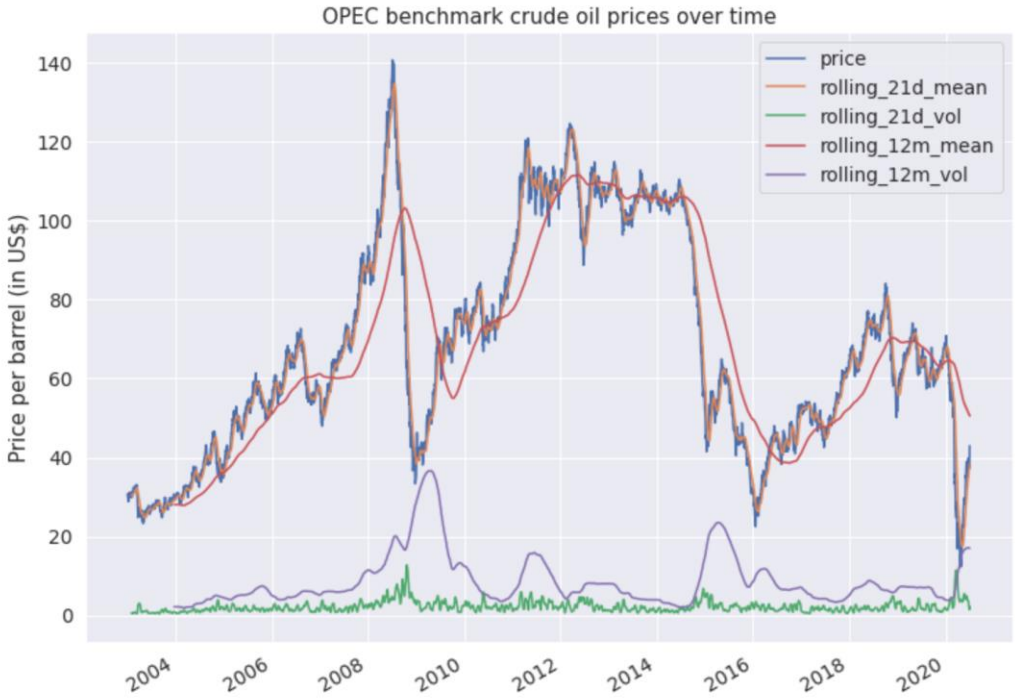



Observations:

- There are upward and downward trends in the prices. The plot looks linear but needs further probing.

- There seems to be seasonality, and we can investigate further by looking at some moving averages.

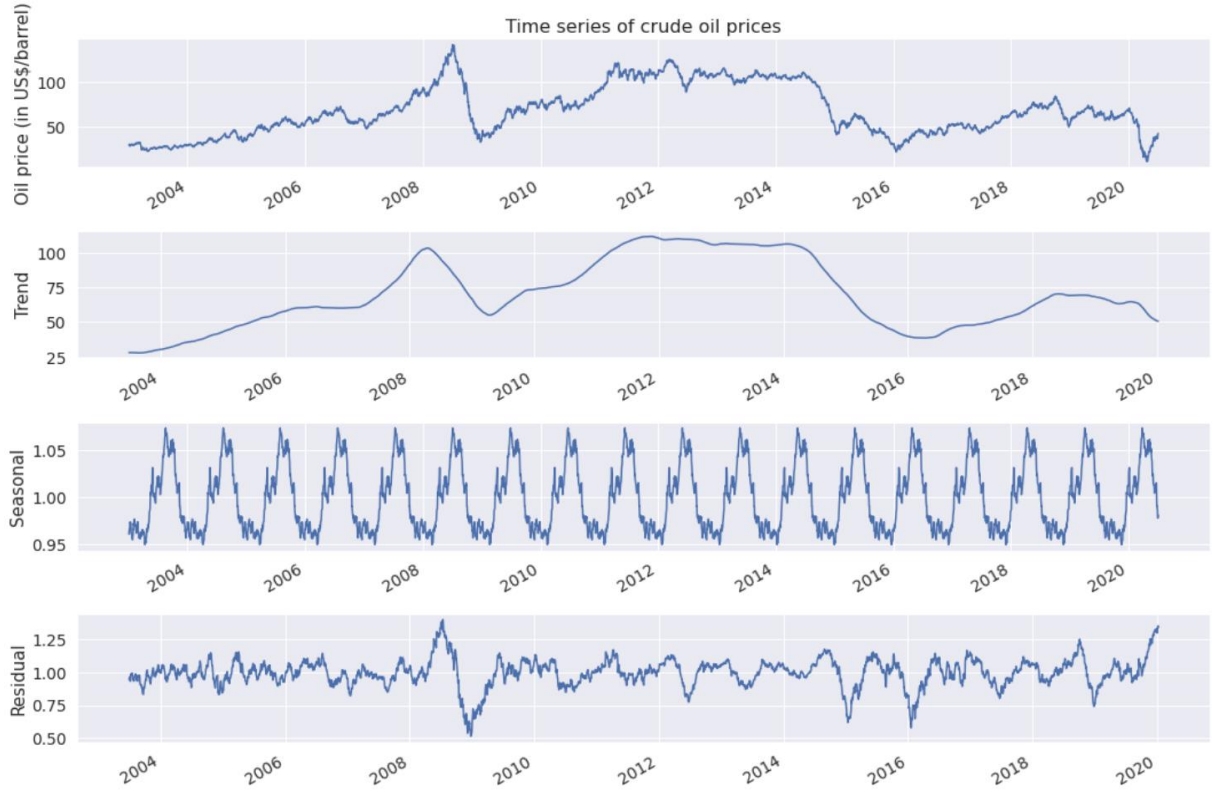
  *The following snapshot shows the OPEC crude oil benchmark prices with rolling mean and standard deviation values for 21-day and 12 months.*




Observations:

- The yearly moving average of the prices shows a linear trend (which changes roughly every couple of years).
- The monthly moving price average shows seasonality.
- The rolling volatility is time-varying in both (monthly and annual) cases.

*The following snapshot shows the seasonal decomposition of the crude oil prices calculated using the multiplicative model.*




Observations:

- The residual plot has non-constant volatility.

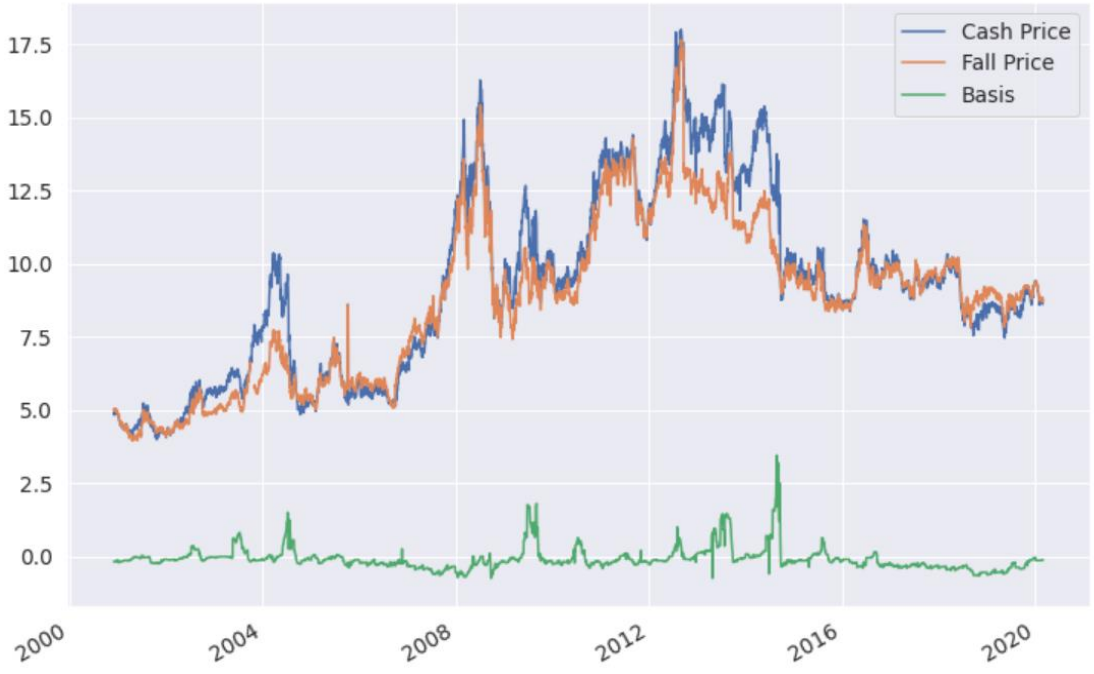
**Method-2 fbprophet library**

This library uses the Prophet() function to decompose the series.

**Soybean prices**

*The following snapshot shows different soybean prices.*


*The following snapshot shows various soybean prices (removing Fall basis price)*




Note:

- We will work with the “Cash price.”

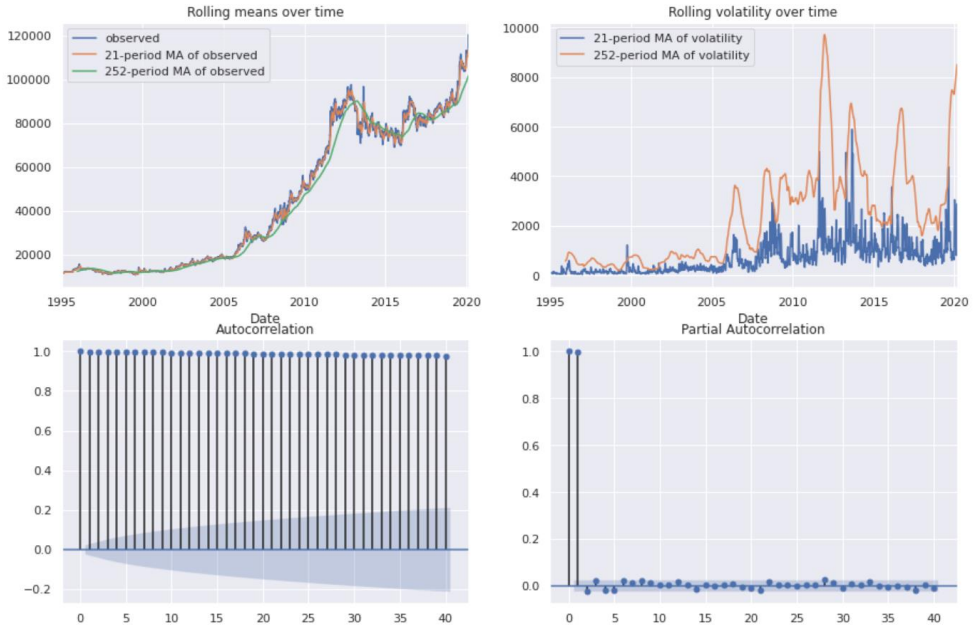
**Testing for stationarity**

There are three ways of checking for stationarity in a time series-

1. Visual inspection
2. Statistical tests
3. ACF/PACF plots

For this section, we work with the last 25 years of daily gold prices in India. The prices shown are denominated in INR per ounce.

*The following snapshot shows the rolling mean, volatility, ACF, PACF and ADF test results for gold prices.*




```
------------------------------------------------------------------
--------- The augmented Dickey-Fuller test results -----------
------------------------------------------------------------------
Test Statistic           1.10185
p-value                  0.995232
# of Lags                     32
# of Observations          6532
Critical Value (1%)       -3.431
Critical Value (5%)       -2.862
Critical Value (10%)      -2.567
dtype: object
------------------------------------------------------------------
```

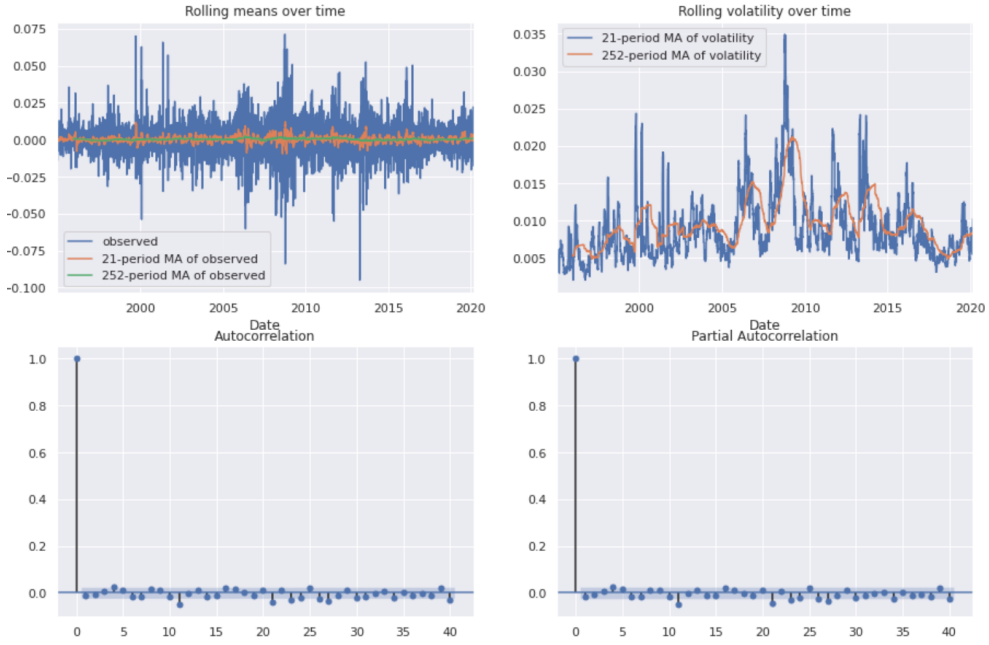
Observations:

- For the gold prices, we have a p-value of nearly 1 (and equivalently, the test statistic is greater than the critical values at all three significance levels), so we conclude that the price series is not stationary.
- The rolling means and volatility plots are time-varying. So we also conclude visually that gold prices in India are non-stationary.

- From the ACF, there are significant autocorrelations above the 95% confidence interval at all lags. From the PACF, we have significance in autocorrelations at lags 1, 2, 3, 6, and 8.

**Checking stationarity for log returns**

*The following snapshot shows the rolling mean, volatility, ACF, PACF and ADF test results for log-returns of gold prices.*




```
------------------------------------------------------------
--------- The augmented Dickey-Fuller test results ----------
------------------------------------------------------------
Test Statistic             -17.4701
p-value                 4.54757e-30
# of Lags                       26
# of Observations             6537
Critical Value (1%)          -3.431
Critical Value (5%)          -2.862
Critical Value (10%)         -2.567
dtype: object
------------------------------------------------------------
```

Observations:

- As per the ADF test results, the returns of gold are stationary since the p-value is almost 0 and the test statistic is less than all the critical values.
- The volatility is time-varying at both the faster and slower rolling levels.
- There are little spikes in the ACF plot at lags 3, 11, and 21.

**Method-3 Modelling time-series using exponential smoothing**

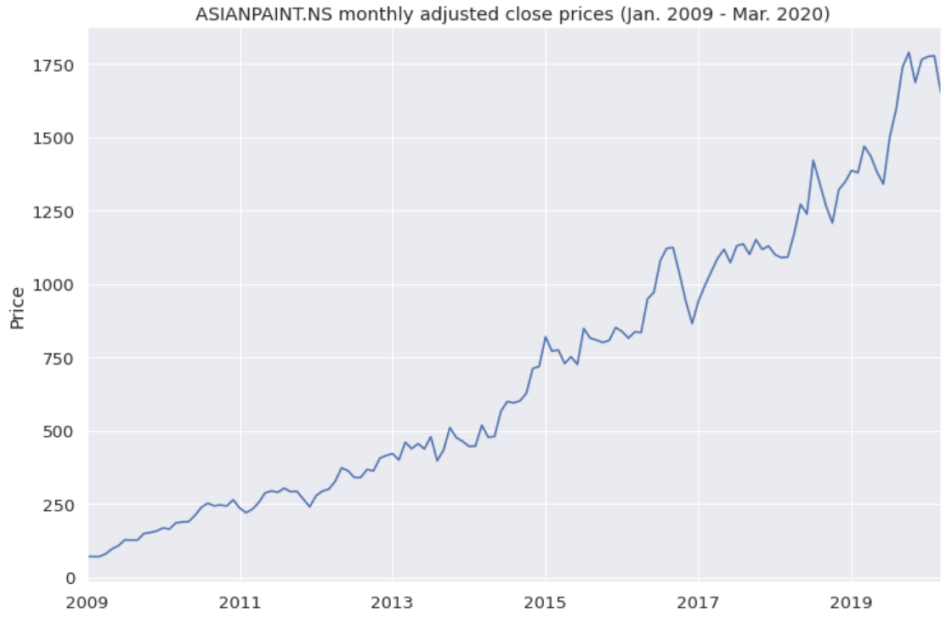
In exponential smoothing, the forecasts are a weighted sum of past observations wherein the weights decrease exponentially as we move further into the past.

**Simple Exponential Smoothing**

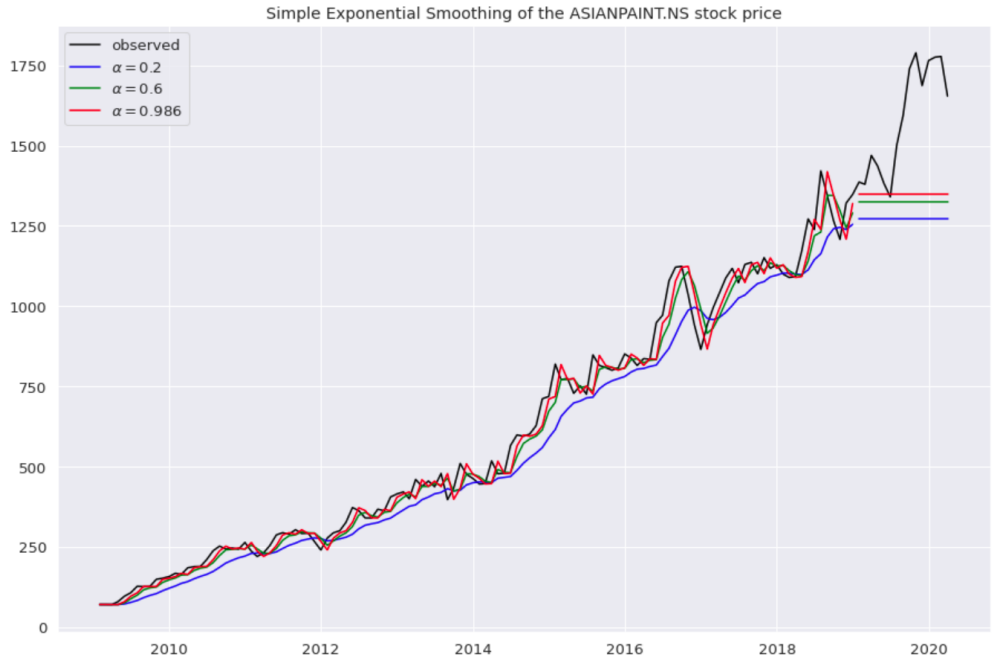
- Most suited for a series that does not exhibit any clear trend or seasonality.
- We employ the data from 2009 to 2018 as part of our training set and 2019 & early 2020 as our test set.

**Example: Asian Paints**

*The following snapshots show the monthly adjusted close prices for Asian Paints.*



*The following snapshot shows the simple exponential smoothing on Asian Paints stock price for various alpha values.*



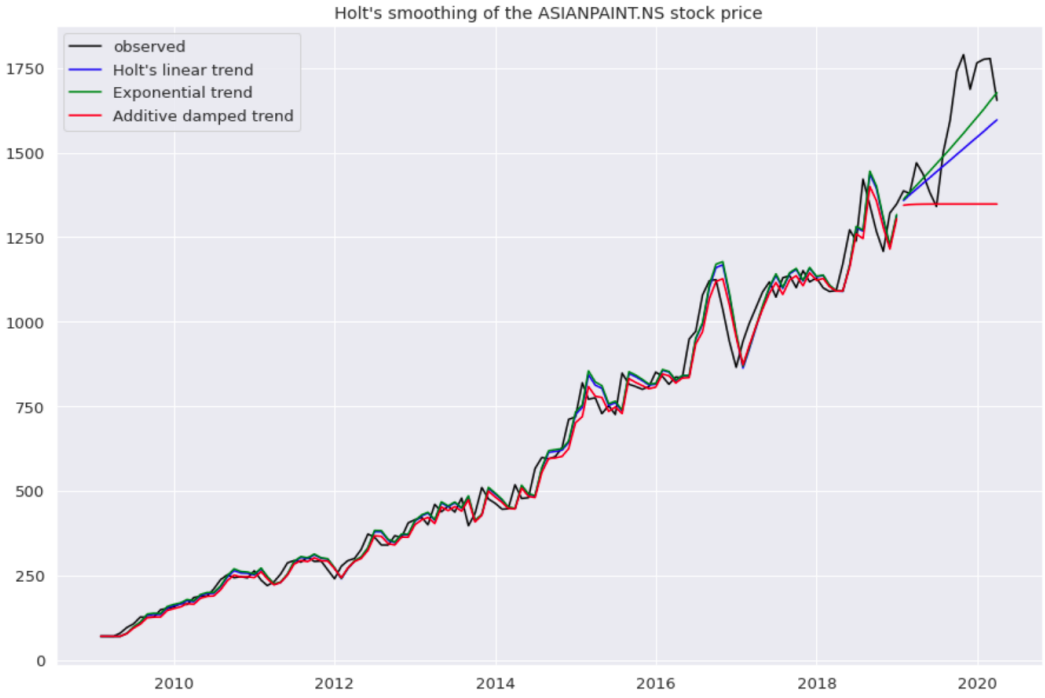

Observations:

- All the forecasts are flat lines.
- We expect to see such a plot since, by definition, SES assumes no trend.
- We manually selected the `smoothing_level` for the first two models.
- In the third model, we let the statsmodel automatically find the optimised value of the smoothing factor.
- Notice that it eventually calculated a value almost equal to 1. That is to say, the best forecast for the price in the next period is the last observed price.

**Holt's Method**

Extension of the SES method to account for trend (but not seasonality) in a time series

*The following snapshot shows Holt's linear trend, Exponential trend and Additive damped trend for Asian Paint stock price.*




Observations:

- All the forecasts are improvements over the SES methods.
- We manually set the values of alpha and beta in all three models.
- In `Holt's linear trend`, we chose the trend to be additive (i.e. linear) by default.
- In `Exponential trend`, we explicitly set the trend as exponential by setting `exponential=True`.
- In `Additive damped trend`, we used a damped version of Holt's linear model by setting `damped=True`.

**Method 4: Modeling time series using ARIMA models**

The **ARIMA (AutoRegressive Integrated Moving Average)** class of models is a popular statistical technique in time series forecasting. It exploits different standard temporal structures seen in time series processes.

That is, the classification ARIMA(p, d, q) process can be thought of as AR(p) I(d) MA(q)
Here,

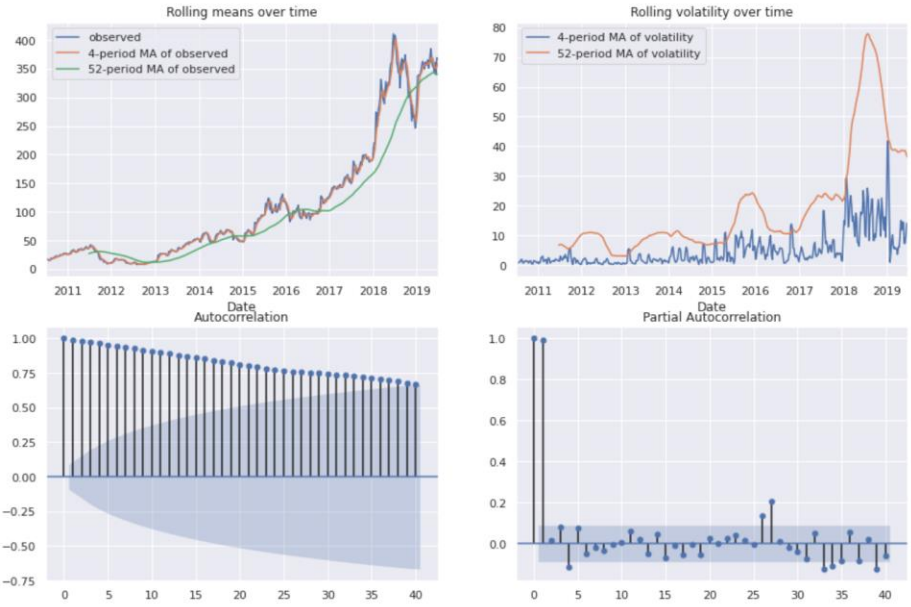
p: The number of past observations of the process included in the model.
d: The number of times we difference the original process to make it stationary.
q: The number of past error terms of the process included in the model.

A well-known deficiency of ARIMA applications on financial time series is its failure to capture the phenomenon of volatility clustering.

We now fit an ARIMA model to Netflix weekly stock prices (from mid-2010 to mid-2019) and learn to evaluate it.

**Checking stationarity**

*The following snapshot shows the rolling mean, volatility, ACF, PACF, and ADF test results for Netflix adjusted close prices.*



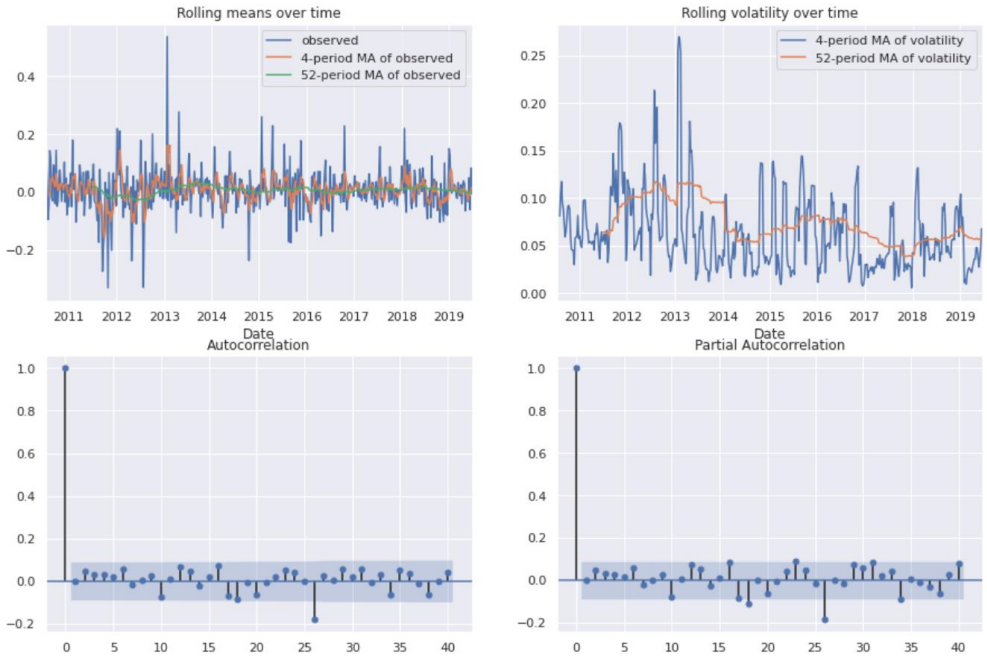

```
------------------------------------------------------------
--------- The augmented Dickey-Fuller test results ----------
------------------------------------------------------------
Test Statistic          0.507267
p-value                 0.985092
# of Lags                    16
# of Observations           453
Critical Value (1%)      -3.445
Critical Value (5%)      -2.868
Critical Value (10%)     -2.570
dtype: object
------------------------------------------------------------
CPU times: user 197 ms, sys: 193 ms, total: 389 ms
Wall time: 160 ms
```

Observations:

- The p-value is nearly 1 (and equivalently, the test statistic is greater than the critical values at all three significance levels). So the ADF test result is that the price series is non-stationary.
- The rolling means and volatility plots are time-varying. So we arrive at the same conclusion by examining the plots.
- From the ACF, there are significant autocorrelations above the 95% confidence interval at all lags. From the PACF, we have spikes at lags 1, 8, 9, 13, 18, 23 and 38.

**Checking stationarity**

*The following snapshot shows the rolling mean, volatility, ACF, PACF, and ADF test results for log-returns of Netflix adjusted close prices.*




```
------------------------------------------------------------------
--------- The augmented Dickey-Fuller test results -----------
------------------------------------------------------------------
Test Statistic         -21.6952
p-value                      0
# of Lags                   0
# of Observations          468
Critical Value (1%)      -3.444
Critical Value (5%)      -2.868
Critical Value (10%)     -2.570
dtype: object
------------------------------------------------------------------
CPU times: user 206 ms, sys: 116 ms, total: 323 ms
Wall time: 154 ms
```

Observations:

- As per the ADF test results, the Netflix returns are stationary since the p-value is almost 0 and the test statistic is less than all the critical values.
- The returns and rolling means of the returns are all centred around 0.
- The volatility is time-varying at both the faster and slower rolling levels.
- We can see bristles near or beyond the blue shadow at lags 17 and 26 in the ACF plot and lags 12, 16, 17, 18 and 26 in the PACF plot.

**Implementing ARIMA model**

*The following snapshot shows the ARMIA result for Netflix adjusted close prices.*

ARIMA Model Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | D.adj_close | **No. Observations:** | 468 |
| **Model:** | ARIMA(3, 1, 2) | **Log Likelihood** | -1669.422 |
| **Method:** | css-mle | **S.D. of innovations** | 8.567 |
| **Date:** | Thu, 16 Sep 2021 | **AIC** | 3352.843 |
| **Time:** | 16:27:35 | **BIC** | 3381.883 |
| **Sample:** | 07-18-2010 | **HQIC** | 3364.270 |
| | - 06-30-2019 | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | 0.7526 | 0.396 | 1.902 | 0.057 | -0.023 | 1.528 |
| **ar.L1.D.adj_close** | -0.4227 | 0.079 | -5.331 | 0.000 | -0.578 | -0.267 |
| **ar.L2.D.adj_close** | -0.8401 | 0.134 | -6.280 | 0.000 | -1.102 | -0.578 |
| **ar.L3.D.adj_close** | 0.0338 | 0.060 | 0.559 | 0.576 | -0.085 | 0.152 |
| **ma.L1.D.adj_close** | 0.3660 | 0.065 | 5.617 | 0.000 | 0.238 | 0.494 |
| **ma.L2.D.adj_close** | 0.8618 | 0.128 | 6.717 | 0.000 | 0.610 | 1.113 |

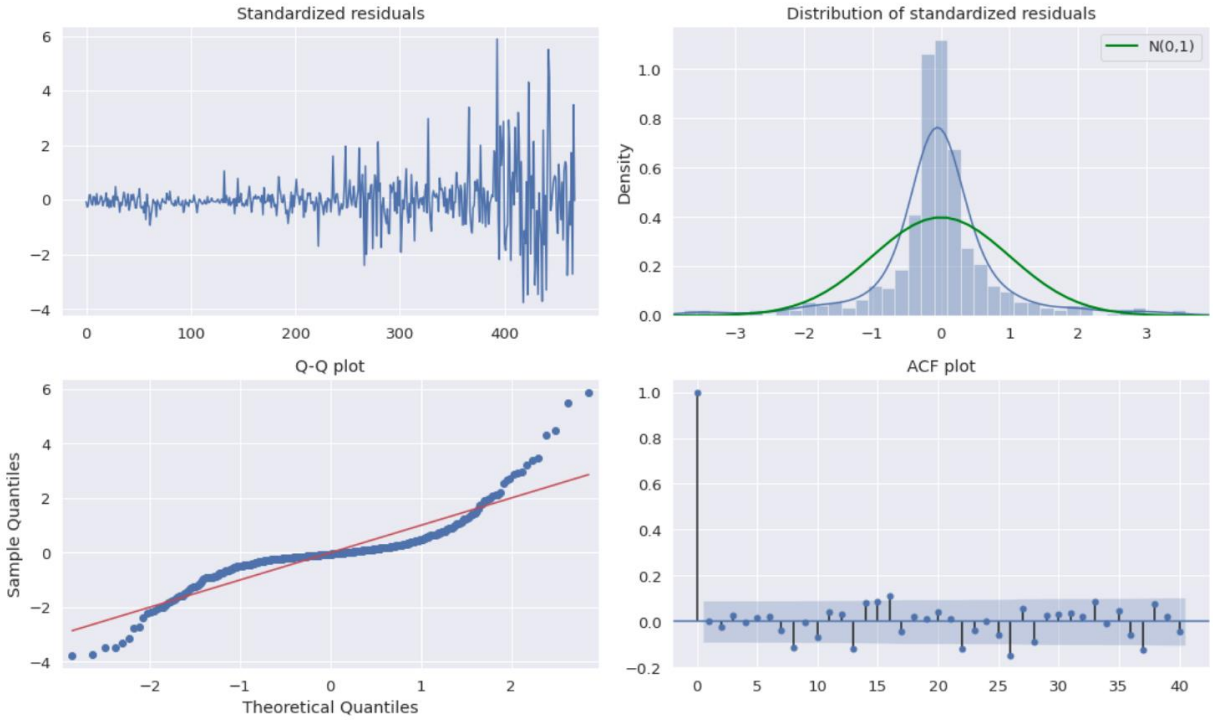
Roots

| | Real | Imaginary | Modulus | Frequency |
|---|---|---|---|---|
| **AR.1** | -0.2692 | -1.0453j | 1.0794 | -0.2901 |
| **AR.2** | -0.2692 | +1.0453j | 1.0794 | 0.2901 |
| **AR.3** | 25.3902 | -0.0000j | 25.3902 | -0.0000 |
| **MA.1** | -0.2123 | -1.0561j | 1.0772 | -0.2816 |
| **MA.2** | -0.2123 | +1.0561j | 1.0772 | 0.2816 |

Observations:

- We chose an ARIMA(3, 1, 2) model to fit the price series of Netflix. Equivalently, we could have fit an ARIMA(3, 0, 2) to the returns instead.
- The most important part is the table at the centre, which has the coefficient values, 95% confidence intervals, and corresponding p-values.
- However, we also need to run model diagnostics by examining the residual errors closely. This will tell us if our model was a good fit for the underlying data.

*The following snapshot shows the residuals, distribution of residuals, quantiles and the ACF plot for residuals from the ARIMA model.*




Observations:

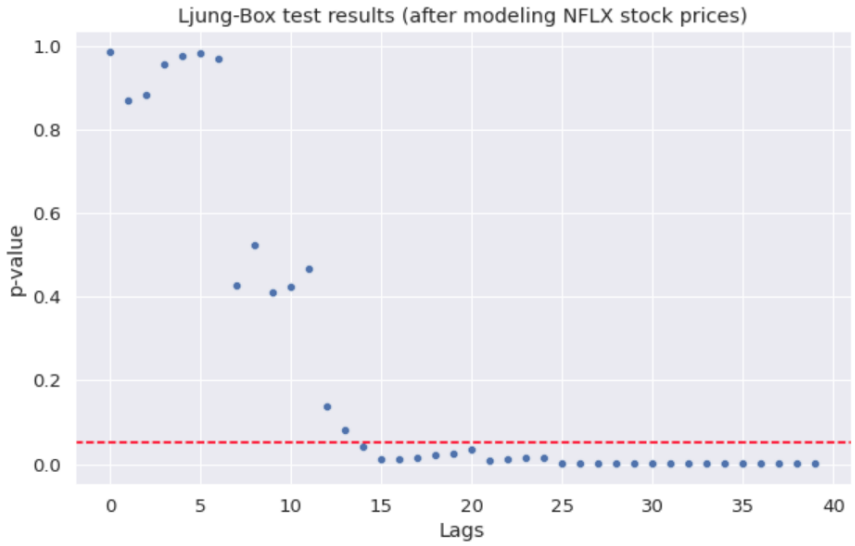
- The mean of the residuals is approximately zero. However, its variance is much higher in the second half of the series.
- The distribution of standardised residuals and Q-Q plot indicate fatter tails than a normal distribution.
- There seem to be serial correlations at lags 8, 13, 14, 22 and a few more.
- If the fit is good, we should see residuals similar to Gaussian white noise. It's not so here.

**Statistical tests**

- To check for autocorrelations in residuals: The Ljung-Box test. The null hypothesis is that the serial correlations of the time series are zero. We use it in addition to a visual interpretation of ACF/PACF plots.

- To check for normality in residuals: The Jarque-Bera test. The null hypothesis is that the time series is normally distributed. We use it for a visual interpretation of plots like the residual distribution and the Q-Q plots.

**The Ljung-Box test**




Observations:

- There are no significant serial correlations until lag 12.
- However, many of the correlations from lag 13 are below the red line, so our model is not a good fit.

**The Jarque-Bera test**

- Jarque-Bera statistic: 1287.88 with p-value: 0.00
- Our residuals are likely not normally distributed.

**Automatically finding the best ARIMA fit using pmdarima library**

*The following snapshot shows the ARIMA results generated using the pmdarima library. ARIMA fit in the left image is using the default parameters while it's using the tuned parameters in the right image*

SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | y | **No. Observations:** | 469 |
| **Model:** | SARIMAX(2, 1, 2) | **Log Likelihood** | -1669.602 |
| **Date:** | Thu, 16 Sep 2021 | **AIC** | 3351.204 |
| **Time:** | 16:32:07 | **BIC** | 3376.095 |
| **Sample:** | 0 | **HQIC** | 3360.999 |
| | - 469 | | |
| **Covariance Type:** | opg | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **intercept** | 1.7575 | 1.010 | 1.740 | 0.082 | -0.222 | 3.737 |
| **ar.L1** | -0.4484 | 0.050 | -8.969 | 0.000 | -0.546 | -0.350 |
| **ar.L2** | -0.8823 | 0.043 | -20.422 | 0.000 | -0.967 | -0.798 |
| **ma.L1** | 0.3709 | 0.052 | 7.153 | 0.000 | 0.269 | 0.472 |
| **ma.L2** | 0.8921 | 0.041 | 21.774 | 0.000 | 0.812 | 0.972 |
| **sigma2** | 73.4617 | 2.311 | 31.794 | 0.000 | 68.933 | 77.990 |

| | | | |
|---|---|---|---|
| **Ljung-Box (Q):** | 79.25 | **Jarque-Bera (JB):** | 1310.00 |
| **Prob(Q):** | 0.00 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 43.56 | **Skew:** | 0.98 |
| **Prob(H) (two-sided):** | 0.00 | **Kurtosis:** | 10.96 |

SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | y | **No. Observations:** | 469 |
| **Model:** | SARIMAX(2, 1, 2) | **Log Likelihood** | -1669.602 |
| **Date:** | Thu, 16 Sep 2021 | **AIC** | 3351.204 |
| **Time:** | 16:40:49 | **BIC** | 3376.095 |
| **Sample:** | 0 | **HQIC** | 3360.999 |
| | - 469 | | |
| **Covariance Type:** | opg | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **intercept** | 1.7575 | 1.010 | 1.740 | 0.082 | -0.222 | 3.737 |
| **ar.L1** | -0.4484 | 0.050 | -8.969 | 0.000 | -0.546 | -0.350 |
| **ar.L2** | -0.8823 | 0.043 | -20.422 | 0.000 | -0.967 | -0.798 |
| **ma.L1** | 0.3709 | 0.052 | 7.153 | 0.000 | 0.269 | 0.472 |
| **ma.L2** | 0.8921 | 0.041 | 21.774 | 0.000 | 0.812 | 0.972 |
| **sigma2** | 73.4617 | 2.311 | 31.794 | 0.000 | 68.933 | 77.990 |

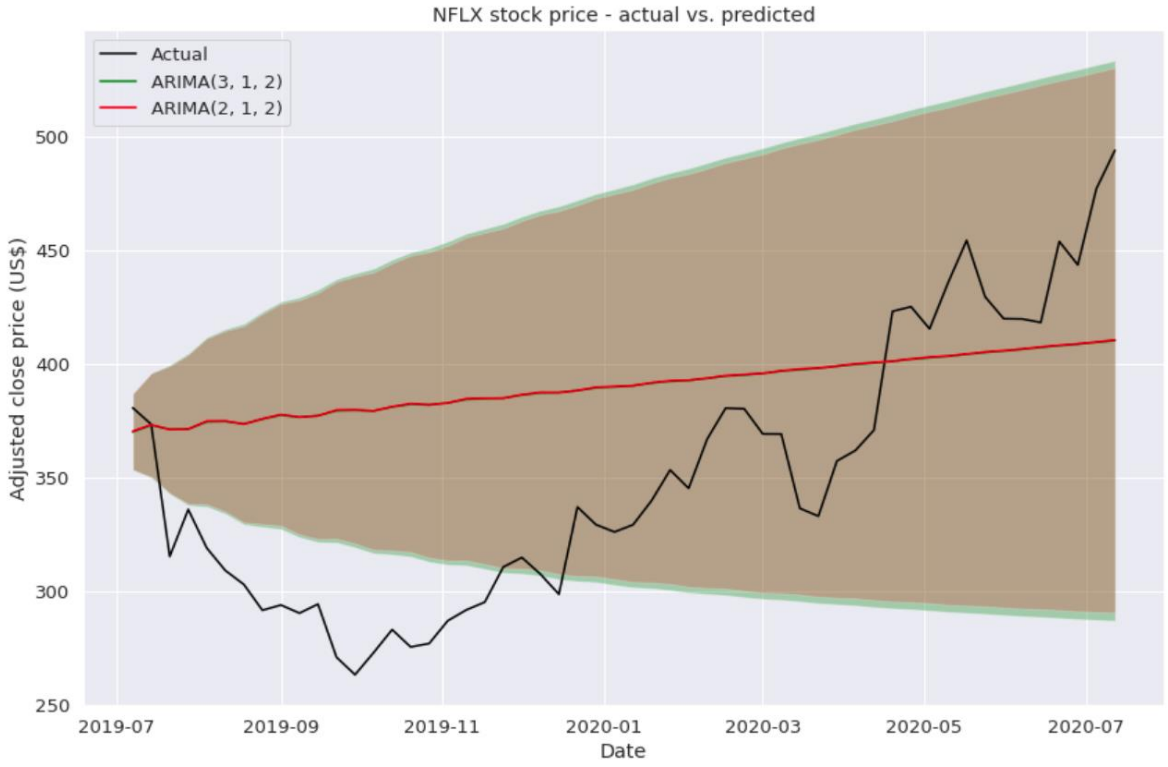
| | | | |
|---|---|---|---|
| **Ljung-Box (Q):** | 79.25 | **Jarque-Bera (JB):** | 1310.00 |
| **Prob(Q):** | 0.00 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 43.56 | **Skew:** | 0.98 |
| **Prob(H) (two-sided):** | 0.00 | **Kurtosis:** | 10.96 |

Observations:

- The most suitable model is ARIMA(2, 1, 2)
- We use AIC to build a parsimonious model
- When choosing from multiple competing models, we choose the one which has the smallest AIC.
- The idea is to find the right balance between underfitting and overfitting. AIC helps us see that.

**Forecasting using the ARIMA class**

*The following snapshot shows the ARIMA (3,1,2) and ARIMA (2,1,2) forecasted results for Netflix adjusted close prices.*




**A brief look at modelling volatility using the ARCH/GARCH family of models**

The ARIMA class of models is widely used in asset price forecasting. However, the ARMA/ARIMA models do not account for volatility clustering. ARCH/GARCH method allows us to model the time-dependent change in the volatility of a time series.

The ARIMA + GARCH combination is used to improve forecasts. In practice, we jointly estimate the mean returns and the volatility associated with the returns.

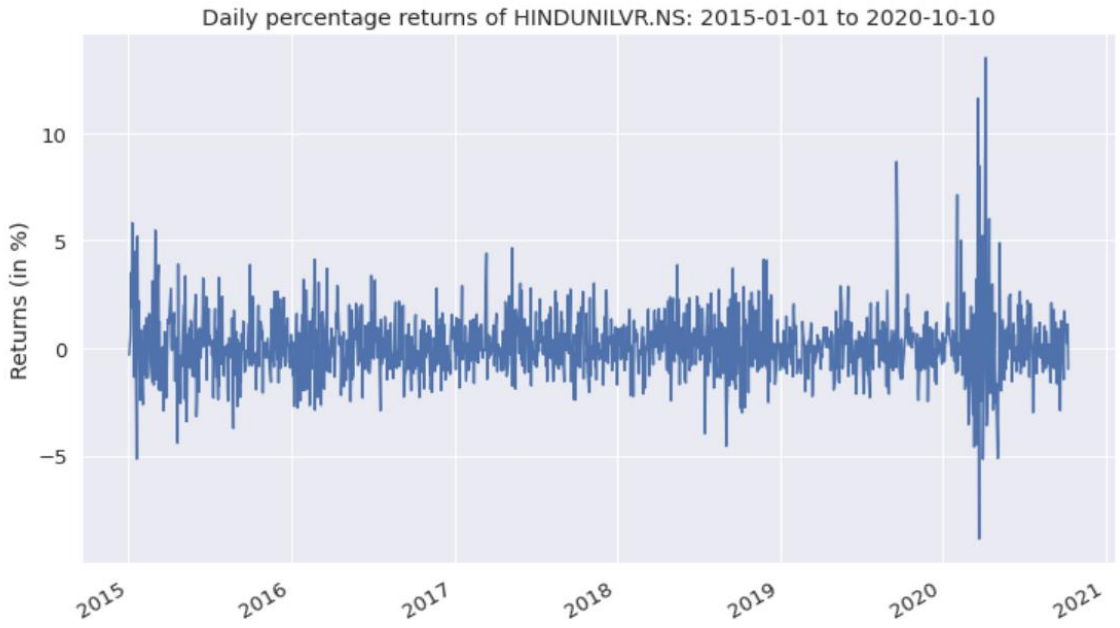
We model conditional variance in Python with the *arch* library.

**Autoregressive Conditionally Heteroskedastic Models (ARCH)**

ARCH estimates the conditional variance over time based on the past values of the variance (hence the name autoregressive).

It is helpful to think of ARCH(q) as the application of AR(p) to the variance of a time series process.

*The following snapshot presents the daily returns of HUL.*




*The following snapshot presents the summary ARCH (1) model fitted on HUL returns.*

```
Iteration:      1,   Func. Count:      4,   Neg. LLF: 2547.926984112656
Iteration:      2,   Func. Count:     10,   Neg. LLF: 2547.9078937181885
Iteration:      3,   Func. Count:     17,   Neg. LLF: 2547.906937303316
Iteration:      4,   Func. Count:     21,   Neg. LLF: 2547.906910929086
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 2547.9069109291895
            Iterations: 4
            Function evaluations: 21
            Gradient evaluations: 4
                      Zero Mean - ARCH Model Results
==============================================================================
Dep. Variable:              adj_close   R-squared:                       0.000
Mean Model:                 Zero Mean   Adj. R-squared:                  0.001
Vol Model:                       ARCH   Log-Likelihood:               -2547.91
Distribution:                  Normal   AIC:                           5099.81
Method:            Maximum Likelihood   BIC:                           5110.33
                                        No. Observations:                 1421
Date:                Thu, Sep 16 2021   Df Residuals:                     1421
Time:                        16:46:05   Df Model:                            0
                          Volatility Model
============================================================================
                 coef    std err          t      P>|t|    95.0% Conf. Int.
----------------------------------------------------------------------------
omega          1.7327      0.165     10.503  8.366e-26   [  1.409,  2.056]
alpha[1]       0.2409  7.530e-02      3.200  1.376e-03 [9.335e-02,  0.389]
============================================================================

Covariance estimator: robust
```

*The following snapshot presents the summary GARCH (1, 1) model fitted on HUL returns.*

```
Iteration:      1,   Func. Count:      5,   Neg. LLF: 2470.179246506139
Iteration:      2,   Func. Count:     13,   Neg. LLF: 2468.9943805013695
Iteration:      3,   Func. Count:     22,   Neg. LLF: 2464.9681047891227
Iteration:      4,   Func. Count:     30,   Neg. LLF: 2464.3820212017654
Iteration:      5,   Func. Count:     35,   Neg. LLF: 2464.324401116124
Iteration:      6,   Func. Count:     40,   Neg. LLF: 2464.316407550874
Iteration:      7,   Func. Count:     45,   Neg. LLF: 2464.3156251368873
Iteration:      8,   Func. Count:     50,   Neg. LLF: 2464.31555584943
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 2464.31555500775
            Iterations: 8
            Function evaluations: 51
            Gradient evaluations: 8
                      Zero Mean - GARCH Model Results
==============================================================================
Dep. Variable:              adj_close   R-squared:                       0.000
Mean Model:                 Zero Mean   Adj. R-squared:                  0.001
Vol Model:                      GARCH   Log-Likelihood:               -2464.32
Distribution:                  Normal   AIC:                           4934.63
Method:            Maximum Likelihood   BIC:                           4950.41
                                        No. Observations:                 1421
Date:                Thu, Sep 16 2021   Df Residuals:                     1421
Time:                        16:47:26   Df Model:                            0
                            Volatility Model
==============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
------------------------------------------------------------------------------
omega          0.1067  4.172e-02      2.556  1.058e-02   [2.488e-02,  0.188]
alpha[1]       0.0603  1.740e-02      3.463  5.335e-04 [2.616e-02,9.437e-02]
beta[1]        0.8840  2.968e-02     29.787 5.688e-195     [  0.826,  0.942]
==============================================================================

Covariance estimator: robust
```

*The following snapshot presents the predicted price range using (ARIMA + GARCH) approach.*

```
5 1 3
Iteration:      1,   Func. Count:      6,   Neg. LLF: 6104.026039469407
Iteration:      2,   Func. Count:     15,   Neg. LLF: 6103.907267778623
Iteration:      3,   Func. Count:     23,   Neg. LLF: 6103.353625835673
Iteration:      4,   Func. Count:     30,   Neg. LLF: 6102.826181653838
Iteration:      5,   Func. Count:     36,   Neg. LLF: 6101.991655893475
Iteration:      6,   Func. Count:     42,   Neg. LLF: 6100.862533230206
Iteration:      7,   Func. Count:     48,   Neg. LLF: 6100.673592385145
Iteration:      8,   Func. Count:     54,   Neg. LLF: 6100.317091784095
Iteration:      9,   Func. Count:     60,   Neg. LLF: 6100.294047587601
Iteration:     10,   Func. Count:     66,   Neg. LLF: 6100.292602499462
Iteration:     11,   Func. Count:     72,   Neg. LLF: 6100.2918558376705
Iteration:     12,   Func. Count:     78,   Neg. LLF: 6100.291852250388
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 6100.291851535754
            Iterations: 12
            Function evaluations: 79
            Gradient evaluations: 12
(2117.249670591576, 2117.374859813964)
CPU times: user 31.1 s, sys: 39.7 s, total: 1min 10s
Wall time: 15.8 s
```

**Additional resources**

- https://towardsdatascience.com/@eryk.lewinson
- https://pyflux.readthedocs.io/en/latest/getting_started.html

- https://tomaugspurger.github.io/modern-7-timeseries
- https://arch.readthedocs.io/en/latest/univariate/univariate_volatility_modeling.html
  https://www.statsmodels.org/devel/examples/notebooks/generated/exponential_smoothing.html
- http://www.blackarbs.com/blog/time-series-analysis-in-Python-linear-models-to-garch/11/1/2016