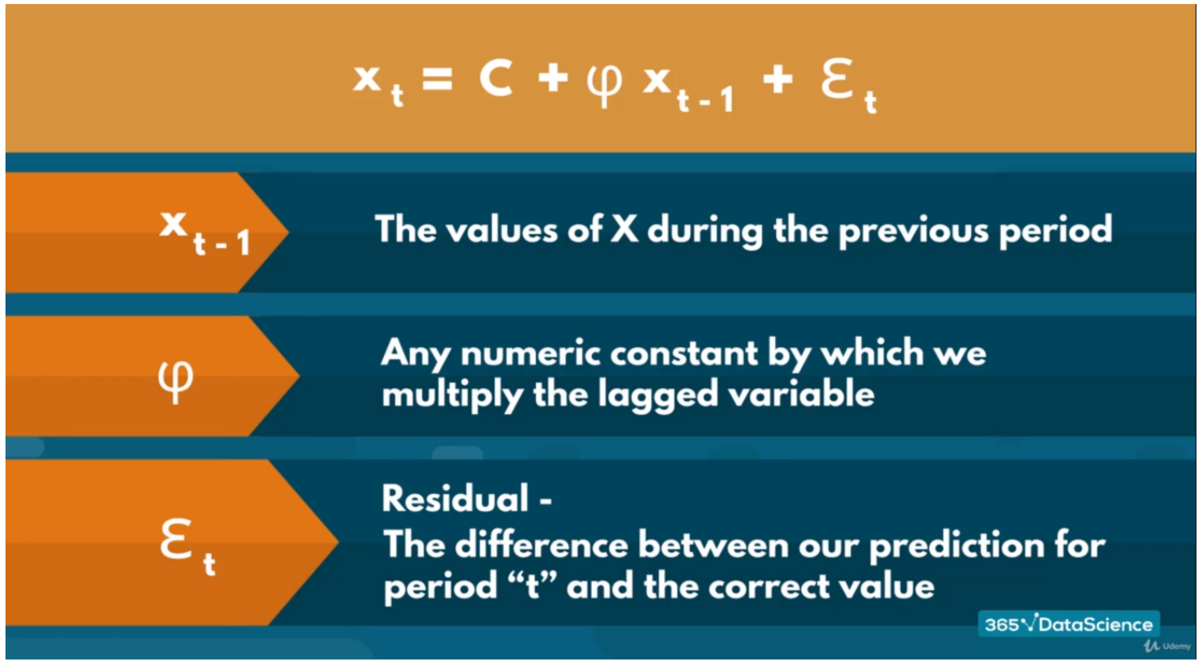
The AR Model

Full Name:

The **A**uto**r**egressive Model

Mathematical Notation:



$$x_t = C + \varphi x_{t-1} + \varepsilon_t$$

x_{t-1}	The values of X during the previous period
φ	Any numeric constant by which we multiply the lagged variable
ε_t	Residual - The difference between our prediction for period "t" and the correct value

365√DataScience

Description:

The name of the model comes from autoregression. This means that the model uses values of the same variable (auto) to estimate the current one (regression).

We rely on autoregressive models when there is clear autocorrelation within the data. The term (autocorrelation) suggests that the variable is correlated with itself. More precisely, values from consecutive periods are related.

Since time series assumes that patters found in the past translate to the future, if autocorrelation is present in the data, we need to us some form of AR model to capture this relationship if we wish to make good estimates.

The AR Model

Implementation of the Simple Model in Python:

The library the *ARMA* method comes from

The method we are importing

```
from statsmodels.tsa.arima_model import ARMA
```

```
model_ar = ARMA(df.market_value, order=(1,0))
```

The variable storing the model characteristics that we will fit later

The time series we wish to analyse

The order of the model **(we use (1,0), since AR(1) = ARMA (1,0))*

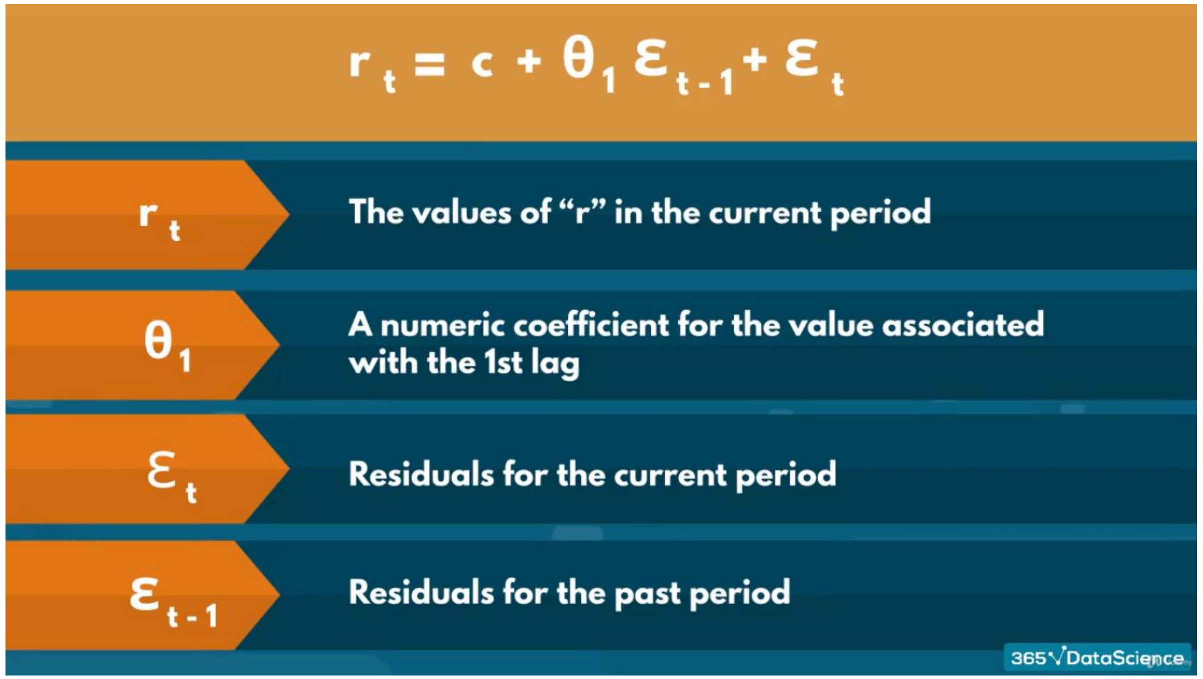
*For an AR(p) model, simply change the order from (1,0) to (p,0).

The MA Model

Full Name:

The **M**oving **A**verage Model

Mathematical Notation:



$$r_t = c + \theta_1 \varepsilon_{t-1} + \varepsilon_t$$

r_t	The values of "r" in the current period
θ_1	A numeric coefficient for the value associated with the 1st lag
ε_t	Residuals for the current period
ε_{t-1}	Residuals for the past period

365√DataScience

Description:

The name of the model comes from the moving averages of fixed period intervals as we cruise move through the data set.

This implies that there are other factors apart from the previous values of the variable that need to be accounted for. Therefore, if we know how far off our predictions were last time, then we have a better chance of estimating the prices better this time.

This is why, these models incorporate past residuals (also known as error terms) to help us improve our estimations. These make sure our model handles unexpected shocks well, which is why it's also known as a smoothing model.

The MA Model

Implementation of the Simple Model in Python:

The library the *ARMA* method comes from

The method we are importing

```python
from statsmodels.tsa.arima_model import ARMA
```

```python
model_ma = ARMA(df.market_value, order=(0,1))
```

The variable storing the model characteristics that we will fit later

The time series we wish to analyse

The order of the model **(we use (0,1), since MA(1) = ARMA (0,1))*

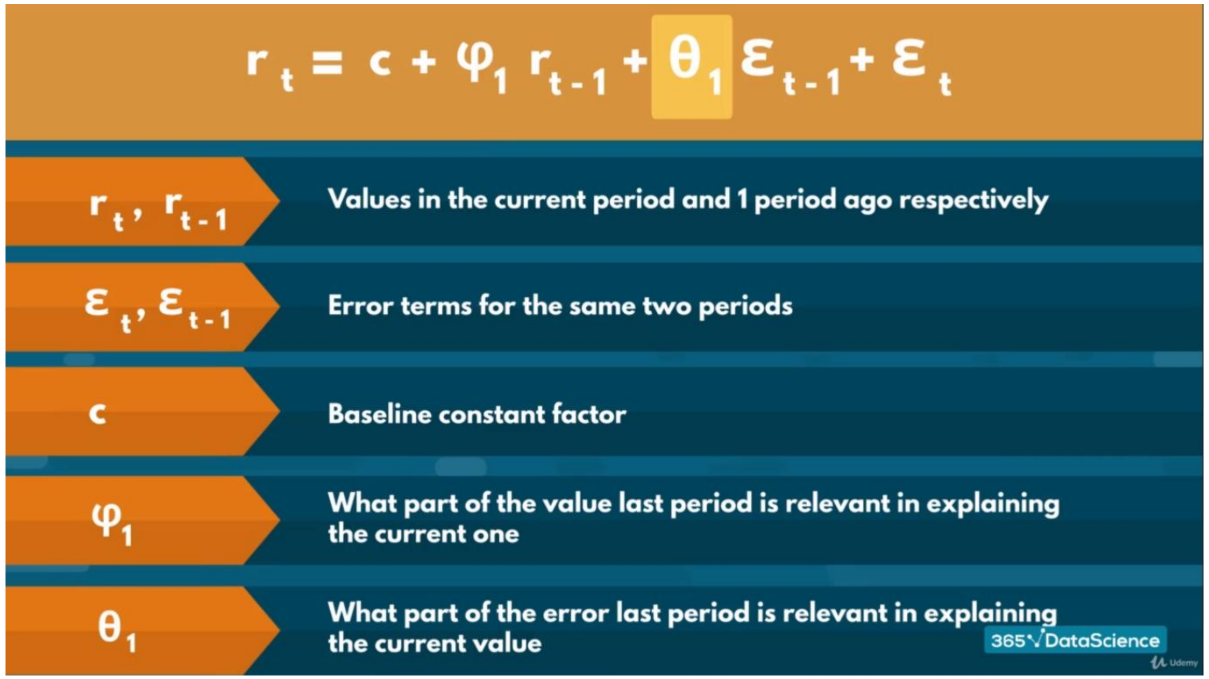
*For an MA(q) model, simply change the order from (0,1) to (0,q).

The ARMA Model

Full Name:

The **A**uto**r**egressive **M**oving **A**verage Model

Mathematical Notation:



$$r_t = c + \varphi_1 r_{t-1} + \theta_1 \varepsilon_{t-1} + \varepsilon_t$$

r_t, r_{t-1}	Values in the current period and 1 period ago respectively
ε_t, ε_{t-1}	Error terms for the same two periods
c	Baseline constant factor
φ_1	What part of the value last period is relevant in explaining the current one
θ_1	What part of the error last period is relevant in explaining the current value

365√DataScience
Udemy

Description:

The name comes from combining the names of the two simpler models it incorporates – the AR and the MA.

The ARMA incorporates both past values (like the AR) and past errors (like the MA). By including both, we should improve our estimates. This is because we are enabling our AR model to calibrate (by including how far off our predictions were) and also giving a benchmark (different from the constant) to the MA model, which should severely decrease the variation in the residuals.

Picking the correct order for such a model could be tricky, since including or removing AR and MA orders can have wildly different effects on the accuracy.

365√DataScience

The ARMA Model

Implementation of the Simple Model in Python:

The library the *ARMA* method comes from

The method we are importing

```
from statsmodels.tsa.arima_model import ARMA
```

```
model_ar_1_ma_1 = ARMA(df.market_value, order=(1,1))
```

The variable storing the model characteristics that we will fit later

The time series we wish to analyse

The order of the model

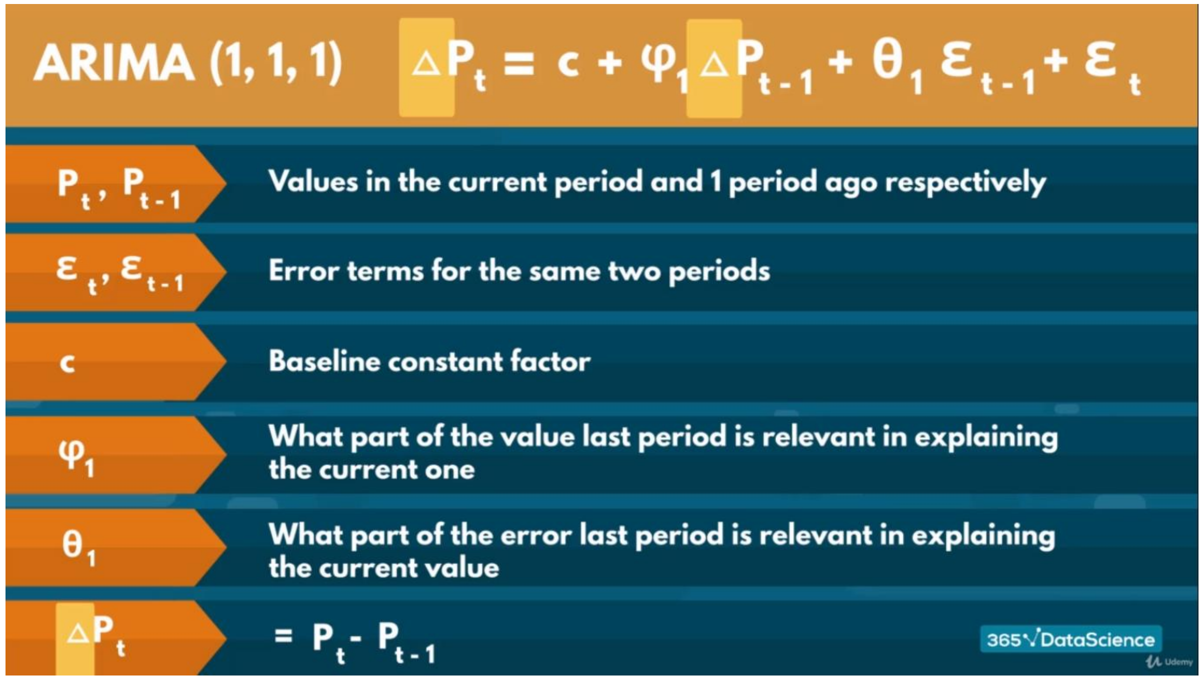
*For an ARMA(p,q) model, simply change the order from (1,1) to (p,q).

The ARIMA Model

Full Name:

The **A**uto**r**egressive **I**ntegrated **M**oving **A**verage Model

Mathematical Notation:



ARIMA (1, 1, 1) $\Delta P_t = c + \varphi_1 \Delta P_{t-1} + \theta_1 \varepsilon_{t-1} + \varepsilon_t$

P_t, P_{t-1}	Values in the current period and 1 period ago respectively
ε_t, ε_{t-1}	Error terms for the same two periods
c	Baseline constant factor
φ_1	What part of the value last period is relevant in explaining the current one
θ_1	What part of the error last period is relevant in explaining the current value
ΔP_t	$= P_t - P_{t-1}$

Description:

The ARIMA is just an integrated version of the ARMA model. What that means is, we simply integrate the data (however many times is needed) to get a stationary set.

Then, we fit a normal ARMA model like we already learned to.

An ARIMA model with 0 degrees of integration is simply an ARMA model, and so any ARIMA (p, 0, q) model is equivalent to an ARMA (p,q).

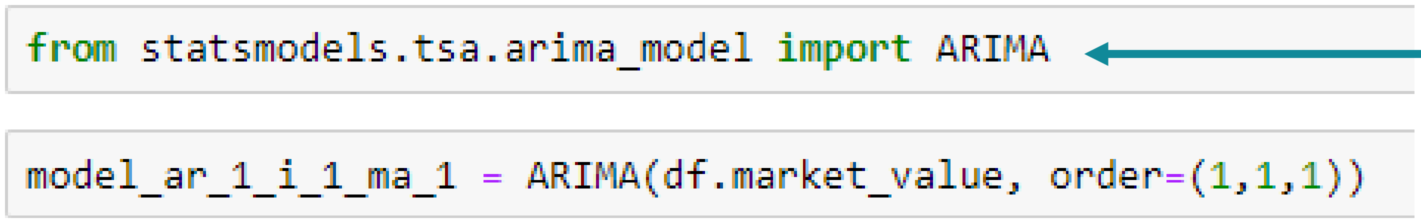
The order of Integration (d) tells us exactly how many times we need to compute the non-seasonal differences between the values to reach stationarity and including more is discouraged (due to data attrition and interpretability of the results).

The ARIMA Model

Implementation of the Simple Model in Python:

The library the *ARIMA* method comes from

The method we are importing



```python
from statsmodels.tsa.arima_model import ARIMA
```

```python
model_ar_1_i_1_ma_1 = ARIMA(df.market_value, order=(1,1,1))
```

The variable storing the model characteristics that we will fit later

The time we wish to analyse

The order of the model

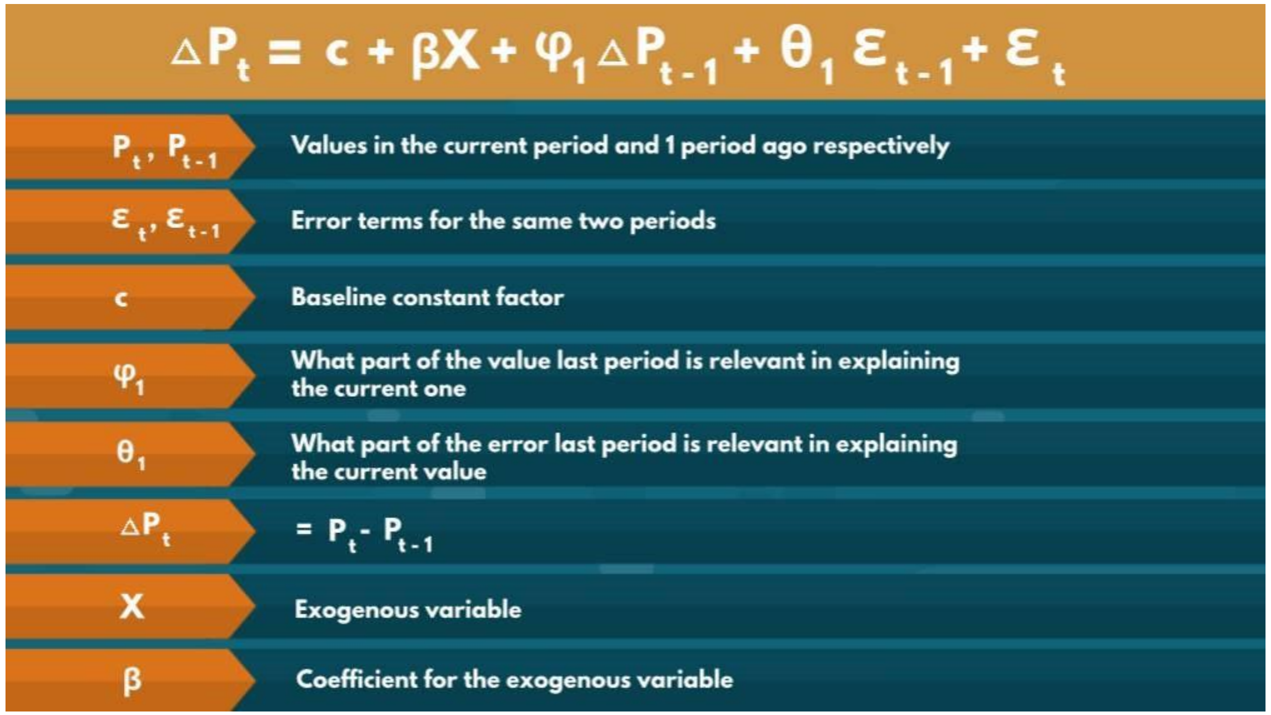
*For an ARIMA(p,d,q) model, simply change the order from (1,1,1) to (p,d,q).

The ARIMAX Model

Full Name:

The **A**uto**r**egressive **I**ntegrated **M**oving **A**verage e**X**ogenous Model

Mathematical Notation:



$$\Delta P_t = c + \beta X + \varphi_1 \Delta P_{t-1} + \theta_1 \varepsilon_{t-1} + \varepsilon_t$$

P_t, P_{t-1}	Values in the current period and 1 period ago respectively
ε_t, ε_{t-1}	Error terms for the same two periods
c	Baseline constant factor
φ_1	What part of the value last period is relevant in explaining the current one
θ_1	What part of the error last period is relevant in explaining the current value
ΔP_t	$= P_t - P_{t-1}$
X	Exogenous variable
β	Coefficient for the exogenous variable

Description:

The ARIMA is just an integrated version of the ARMA model. What that means is, we simply integrate the data (however many times is needed) to get a stationary set.

Then, we fit a normal ARMA model like we already learned to.

An ARIMA model with 0 degrees of integration is simply an ARMA model, and so any ARIMA (p, 0, q) model is equivalent to an ARMA (p,q).

The order of Integration (d) tells us exactly how many times we need to compute the non-seasonal differences between the values to reach stationarity and including more is discouraged (due to data attrition and interpretability of the results).

The ARIMAX Model

Implementation of the Simple Model in Python:

The library the *ARIMA* method comes from

The method we are importing

```
from statsmodels.tsa.arima_model import ARIMA
```

```
model_ar_1_i_1_ma_1_X_spx = ARIMA(df.market_value, exog = df.spx, order=(1,1,1))
```

The variable storing the model characteristics that we will fit later

The time series we wish to analyse

The exogenous variable we are adding to the ARIMA model

The order of the model

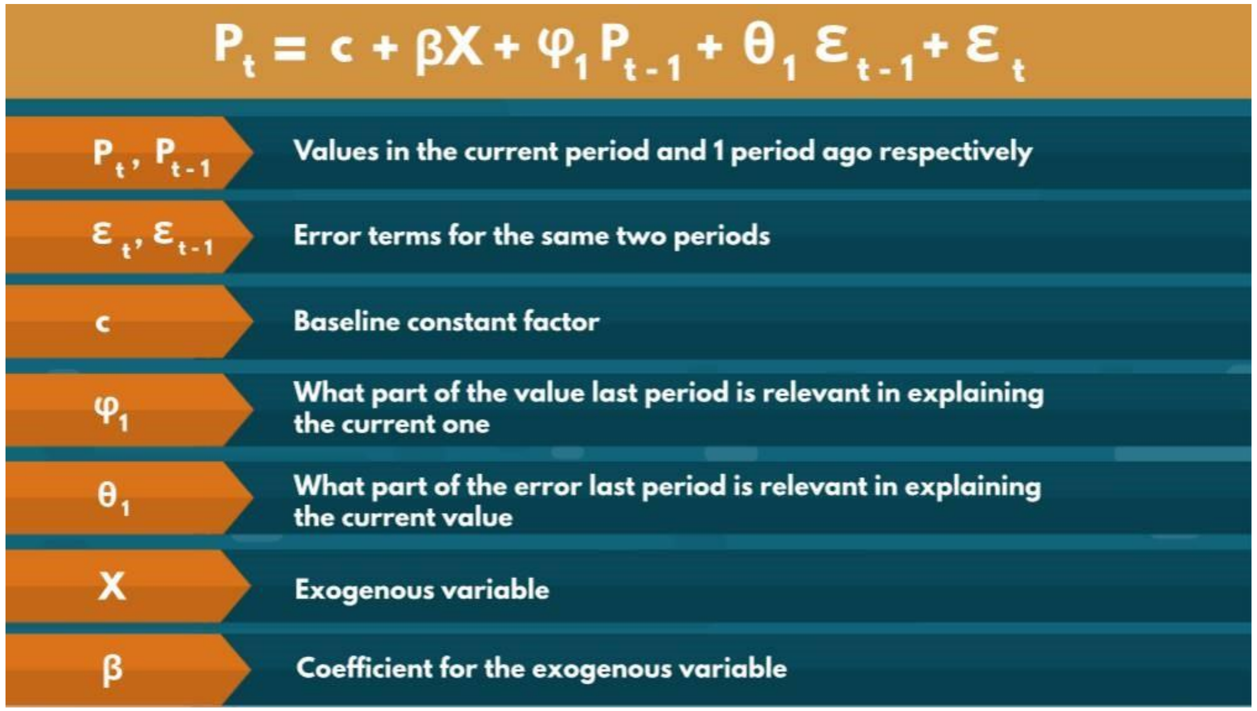
*For an ARIMAX(p,d,q) model, simply change the order from (1,1,1) to (p,d,q).

The ARMAX Model

Full Name:

The **A**uto**r**egressive **M**oving **A**verage e**X**ogenous Model

Mathematical Notation:



$$P_t = c + \beta X + \varphi_1 P_{t-1} + \theta_1 \varepsilon_{t-1} + \varepsilon_t$$

P_t, P_{t-1}	Values in the current period and 1 period ago respectively
ε_t, ε_{t-1}	Error terms for the same two periods
c	Baseline constant factor
φ_1	What part of the value last period is relevant in explaining the current one
θ_1	What part of the error last period is relevant in explaining the current value
X	Exogenous variable
β	Coefficient for the exogenous variable

Description:

The ARMAX is an extension of the ARIMA model, which incorporates other **exogenous** variables.

These variables can be pretty much anything that can have an affect on the values we are trying to estimate. The only requirement is that we have data available for every time period we are interested in. Thus, we often rely on other time series as the exogenous components in the regression.

These models are great, when a big part of the change period to period cannot be explained by past values and past errors alone, so including other relevant values might be of great help (like the prices for an index of a market of a neighbouring country).

365√DataScience

The ARMAX Model

Implementation of the Simple Model in Python:

The library the *ARMA* method comes from

The method we are importing

```
from statsmodels.tsa.arima_model import ARMA
```

```
model_ar_1_ma_1_X_spx = ARMA(df.returns[1:], exog = df.returns_spx[1:], order=(1,1))
```

The variable storing the model characteristics that we will fit later

The time series we wish to analyse

The exogenous variable we are adding to the ARMA model

The order of the model

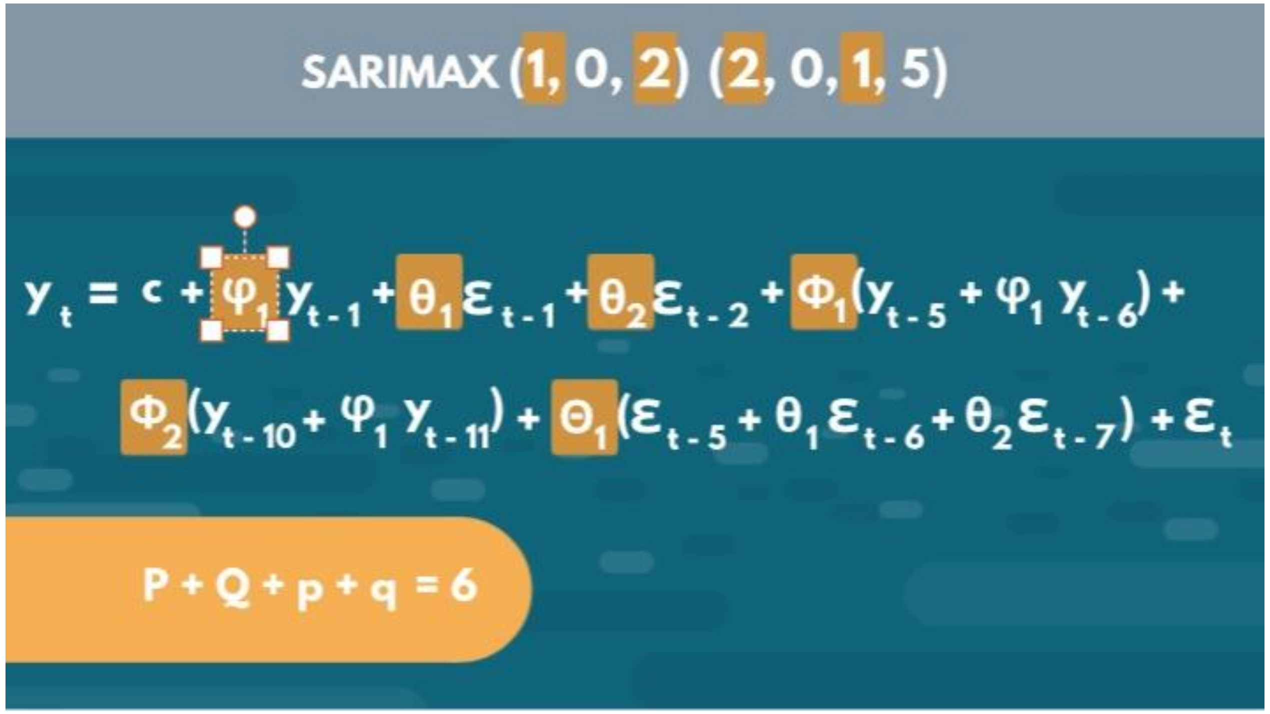
*For an ARMAX(p,q) model, simply change the order from (1,1) to (p,q).

The SARIMAX Model

Full Name:

The **S**easonal **A**uto**r**egressive **I**ntegrated **M**oving **A**verage e**X**ogenous Model

Mathematical Notation:



SARIMAX (1, 0, 2) (2, 0, 1, 5)

$$y_t = c + \varphi_1 y_{t-1} + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \Phi_1 (y_{t-5} + \varphi_1 y_{t-6}) + \Phi_2 (y_{t-10} + \varphi_1 y_{t-11}) + \Theta_1 (\varepsilon_{t-5} + \theta_1 \varepsilon_{t-6} + \theta_2 \varepsilon_{t-7}) + \varepsilon_t$$

$$P + Q + p + q = 6$$

Short Description:

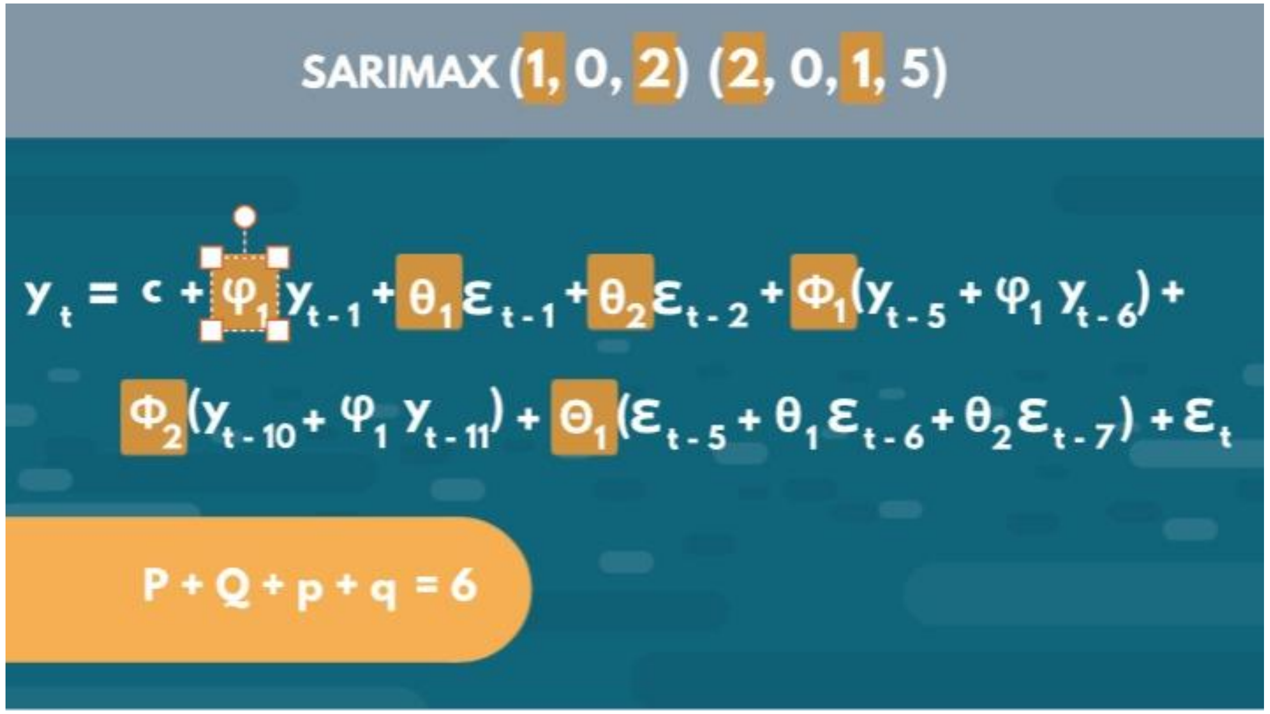
The SARIMAX is the **seasonal** equivalent of the ARIMAX model. Of course, there exist seasonal versions of the other models as well (SARMA, SARIMA, SARMAX, etc.).

Seasonal models help capture patterns which aren't ever-present but appear periodically. For example, the amount of flights leaving an international hub like JFK Airport in NYC are far larger in December compared to October.

That is mainly due to the festive period for many countries in December. Thus, October is far less busy. Therefore, we need a way to account for this expected influx of demand in December and we can do so by checking the values in December of the previous year.

The SARIMAX Model

Equivalents of the SARIMAX:



SARIMAX (1, 0, 2) (2, 0, 1, 5)

$$y_t = c + \varphi_1 y_{t-1} + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \Phi_1(y_{t-5} + \varphi_1 y_{t-6}) + \Phi_2(y_{t-10} + \varphi_1 y_{t-11}) + \Theta_1(\varepsilon_{t-5} + \theta_1 \varepsilon_{t-6} + \theta_2 \varepsilon_{t-7}) + \varepsilon_t$$

$$P + Q + p + q = 6$$

The SARIMAX is among the most-complicated models we can have, since it **can** incorporate seasonality, integration and/or exogenous variables.

However, it doesn’t **have** to.

By setting the values of certain orders to 0, or by not providing certain information, the model can be simplified.

For instance, by not including exogenous variables and having no integration, the model automatically becomes equivalent to a SARMA.

The equation on the left is exactly that - a SARIMAX equivalent of a SARMA.

The SARIMAX Model

The Original Equation:

So, a seasonal model has 7 orders split into two parts – seasonal vs nonseasonal: SARIMAX (p, d, q) (P, D, Q, s)

The nonseasonal ones are the ARIMA lags we're already used to: **p**, **d** and **q**. The rest are the seasonal ones – **P**, **D**, **Q** and **s**. The first 3 are obviously the seasonal equivalents of the **p**, **d** and **q**, while **s** is the only new one. It represents the length of the **season**, hence the name – '**s**'.

Now, the seasonal order (P, Q) determines the number of seasons we're going back. For instance, if P = 2 and s = 10, then we're including the values from 1 and 2 seasons ago, which is the same as 10 and 20 periods ago.

Then, if p = 1, we'd be including $X_{t-1}, X_{t-10}, X_{t-11}, X_{t-20}$ and X_{t-21}. That is because for each of the two seasons, we also need to include p-many past values relevant to it. Thus, for each seasons (X_{t-10}, X_{t-20}), we also include 1 additional past value (X_{t-11}, X_{t-21}).

To make it easier, let's see what a SARIMAX (1,0,0) (2,0,0,10) model looks like:

$$X_t = C + \phi_1 X_{t-1} + \phi_{10} X_{t-10} + \phi_{11} X_{t-11} + \phi_{20} X_{t-20} + \phi_{21} X_{t-21} + \epsilon_t$$

The SARIMAX Model

The Modified Equation:

However, the values for ϕ_{11} and ϕ_{21} are restricted. They must be equal to $\phi_1\ \phi_{10}$ and $\phi_1\ \phi_{20}$ respectively.

$$X_t = C + \phi_1 X_{t-1} + \phi_{10} X_{t-10} + \phi_1 \phi_{10} X_{t-11} + \phi_{20} X_{t-20} + \phi_1 \phi_{20} X_{t-21} + \epsilon_t$$

Thus, we can rewrite the equation to get the following:

$$X_t = C + \phi_1 X_{t-1} + \phi_{10}(X_{t-10} + \phi_1 X_{t-11}) + \phi_{20}(X_{t-20} + \phi_2 X_{t-21}) + \epsilon_t$$

For consistency, we like to use distinct notation for the seasonal coefficients as well, so we plug in Φ_1 and Φ_2 for ϕ_{10} and ϕ_{20}.

$$X_t = C + \phi_1 X_{t-1} + \Phi_1(X_{t-10} + \phi_1 X_{t-11}) + \Phi_2(X_{t-20} + \phi_2 X_{t-21}) + \epsilon_t$$

Now, this is the actual model that gets regressed. In other words, Python only complies a constant and 3 coefficients: ϕ_1, Φ_1 and Φ_2. Thus, even though the model uses very many past variables, it only needs to compute (p + P) – many values.

The SARIMAX Model

Past Seasons and Past Residuals:

Now, if we decide to include residuals, you need to know that the seasonal orders don't directly affect one another. To see what we mean, here is what a SARIMA (1,0,2)(2,0,1,10) looks like:

$$X_t = C + \phi_1 X_{t-1} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \phi_{10} X_{t-10} + \phi_{11} X_{t-11} + \phi_{20} X_{t-20} + \phi_{21} X_{t-21} \\ + \theta_{10} \epsilon_{t-10} + \theta_{11} \epsilon_{t-11} + \theta_{12} \epsilon_{t-12} + \epsilon_t$$

We've highlighted the new additions in red. We se that simply because we're adding lags, doesn't mean we're expanding the coefficients we're including. In other words, we're not including the value for t-12 only because we're adding the residual for that period.

Additionally, the coefficients θ_{11} and θ_{12} are restricted too and equal $\theta_{10}\theta_1$ and $\theta_{10}\theta_2$ respectively. We can once again plug in and substitute a few things. We don't plan on going over each step once more, so we eventually reach the following:

$$X_t = C + \phi_1 X_{t-1} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \Phi_1 (X_{t-10} + \phi_1 X_{t-11}) + \Phi_2 (X_{t-20} + \phi_1 X_{t-21}) + \Theta_1 (\epsilon_{t-10} + \theta_1 \epsilon_{t-11} + \theta_2 \epsilon_{t-12}) + \epsilon_t$$

The SARIMAX Model

A Quick Look at the Coefficients:

Now that we know what the actual equation of a SARIMAX (1,0,2) (2,0,1,10) looks like, let's make a few remarks:

$$X_t = C + \phi_1 X_{t-1} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \Phi_1 (X_{t-10} + \phi_1 X_{t-11}) + \Phi_2 (X_{t-20} + \phi_1 X_{t-21}) + \Theta_1 (\epsilon_{t-10} + \theta_1 \epsilon_{t-11} + \theta_2 \epsilon_{t-12}) + \epsilon_t$$

Even though we are using values from 10 different past values and/or residuals, we're only estimating 6 coefficients (excluding the constant). Therefore, when we fit a model, we only get a coefficient for each order, rather than one for each value we're using.

Additionally, we have to include more past values because if the value from yesterday affects the value today, then the value from 11 days ago affects the one from 10 days ago. This is the entire reason we're not only including X_{t-10}, X_{t-20} and ϵ_{t-10} in the model, but also the values that shape them.

You can think of the values we're including as a time series with a different frequency. Notice how $X_{t-10} + \phi_1 X_{t-11}$ and $X_{t-20} + \phi_1 X_{t-21}$ are essentially the same thing 1 season (10 periods) apart. Then, we just think of seasonal patterns as trends with a different frequency we need to include in order to make good estimations.

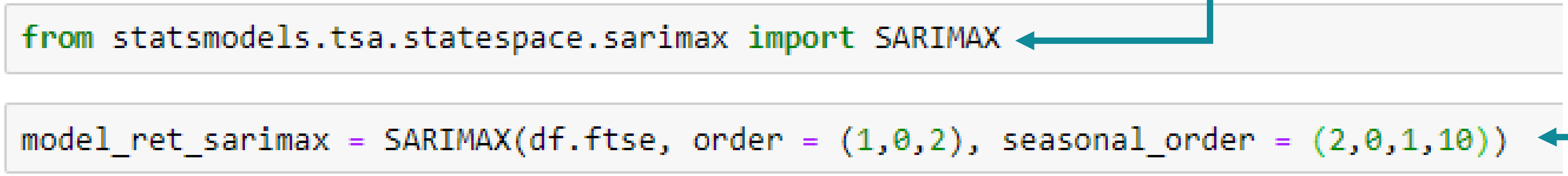
The SARIMAX Model

Implementation of the Model in Python:

The library the *SARIMAX* method comes from

The method we are importing

The seasonal order of the model



```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

```
model_ret_sarimax = SARIMAX(df.ftse, order = (1,0,2), seasonal_order = (2,0,1,10))
```

The variable storing the model characteristics that we will fit later

The time series we wish to analyse

The non-seasonal order of the model

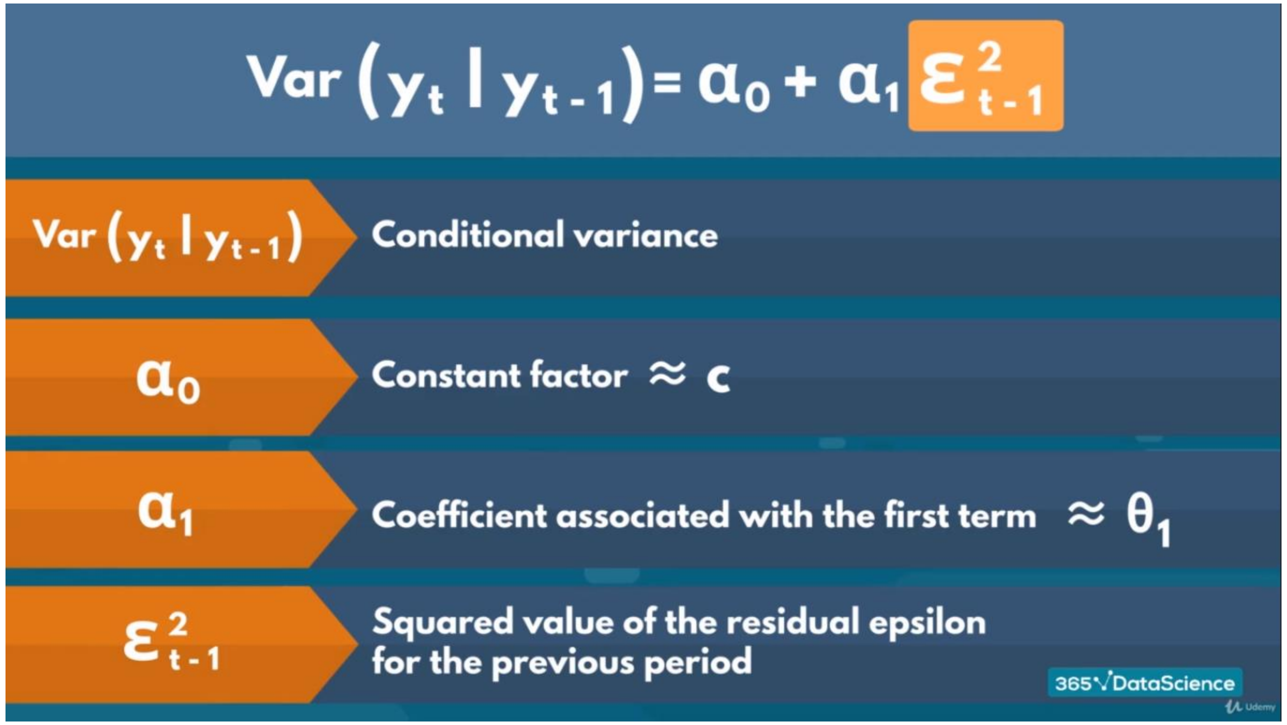
*For an SARIMAX(p,d,q)(P,D,Q,s) model, simply change the order from (1,0,2) to (p,d,q), and the seasonal order from (2,0,1,10) to (P,D,Q,s).

The ARCH Model

Full Name:

The **A**uto**r**egressive **C**onditional **H**eteroskedasticity Model

Mathematical Notation:



$$\mathrm{Var}(y_t \mid y_{t-1}) = \alpha_0 + \alpha_1 \varepsilon^2_{t-1}$$

$\mathrm{Var}(y_t \mid y_{t-1})$	Conditional variance
α_0	Constant factor $\approx c$
α_1	Coefficient associated with the first term $\approx \theta_1$
ε^2_{t-1}	Squared value of the residual epsilon for the previous period

365√DataScience

Description:

Unlike the previous models, the ARCH measures volatility of the results, rather than the results themselves. Thus, the purpose of it is entirely different and focused on predicting turbulence in the data, regardless of whether it's an increase or decrease in the values.

As you can see on the left side of the equation, the endogenous variable is the variance, rather than the time series variable.

Thus, this is only the variance equation of the model. The simplest ARCH model assumes a 0 or constant mean, so this is the only equation we are interested in.

The ARCH Model

Implementation of the Simple Model in Python:

The library the *arch_model* method comes from

The method we are importing

The type of volatility model we are assuming for the volatility equation

```
from arch import arch_model
```

```
model_arch_1 = arch_model(df.returns[1:], mean = "Constant", vol = "ARCH", p = 1)
```

The variable storing the model characteristics that we will fit later

The time series, whose volatility we wish to analyse

The type of model we are assuming for the mean equation

The order of the model

*For an ARCH(q) model, simply change p from 1 to q.

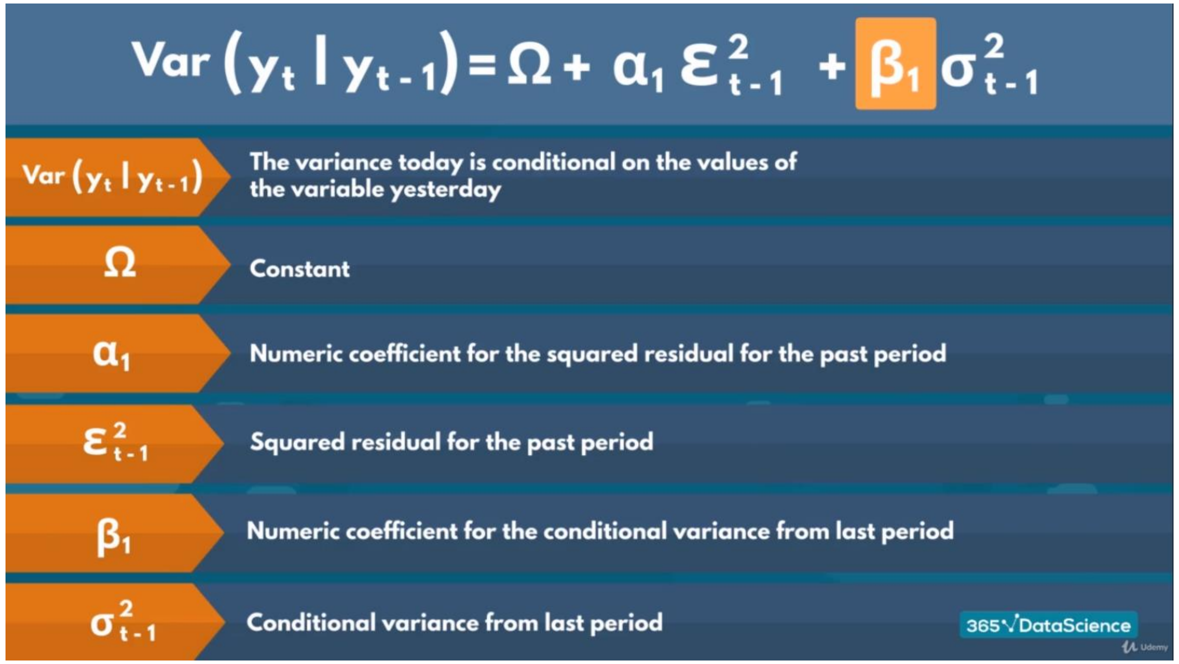
365√DataScience

The GARCH Model

Full Name:

The **G**eneralized **A**uto**r**egressive **C**onditional **H**eteroskedasticity Model

Mathematical Notation:



$$Var(y_t \mid y_{t-1}) = \Omega + \alpha_1 \varepsilon^2_{t-1} + \beta_1 \sigma^2_{t-1}$$

$Var(y_t \mid y_{t-1})$	The variance today is conditional on the values of the variable yesterday
Ω	Constant
α_1	Numeric coefficient for the squared residual for the past period
ε^2_{t-1}	Squared residual for the past period
β_1	Numeric coefficient for the conditional variance from last period
σ^2_{t-1}	Conditional variance from last period

Description:

As the name suggests, the GARCH is just the generalized version of the ARCH model.

This generalization is expressed in including past variances as well as past squared residuals to estimate current (and subsequent) variances.

The generalization comes from the fact that including a single past variance would (in theory) contain in itself the explanatory power of all other previous squared error terms.

It serves as a sort of ARMA equivalent to the ARCH, where we're including both past values and past errors (albeit squared).

The GARCH Model

Implementation of the Simple Model in Python:

The library the *arch_model* method comes from

The method we are importing

The type of volatility model we are assuming for the volatility equation

```python
from arch import arch_model
```

```python
model_garch_1_1 = arch_model(df.returns[1:], mean = "Constant", vol = "GARCH", p = 1, q = 1)
```

The variable storing the model characteristics that we will fit later

The time series, whose volatility we wish to analyse

The type of model we are assuming for the mean equation

The orders of the model

*For an GARCH(p,q) model, simply change p from 1 to q and the q from 1 to p.