I think that they are fantastic. I attended Yale and Stanford and have worked at Honeywell,Oracle, and Arthur Andersen(Accenture) in the US. I have taken Big Data and Hadoop,NoSQL, Spark, Hadoop...

Ray han

Tech Leader | Stanford / Yale University

ProjectPro is a unique platform and helps many people in the industry to solve real-life problems with a step-by-step walkthrough of projects. A platform with some fantastic resources to gain hands-...

Anand Kumpatla

Sr Data Scientist @ Doubleslash Software Solutions Pvt Ltd

Not sure what you are looking for? View All Projects

# How to Justify the Use of the ARIMA Model?

Suppose the past values in your data affect the current or future values or can foretell future trends based on recent fluctuations. In that case, time-series forecasting is the solution for such a regression problem. Several other time-series forecasting models rely on incorporating successive changes or more recent developments in the data to predict future trends. On the other hand, some other models use purely statistical quantities that often incorporate trends from historical data that might not be as relevant in the present or future values. These assumptions and approaches have a valid rationale but often fail in real life.

ARIMA incorporates these ideas in its combined autoregressive and moving-average approach to model stationary time-series data. This approach figures out the importance of past fluctuations, includes overall trends and deals with smoothening the effect of outliers or temporary abnormal changes in the data. ARIMA is, thus, a perfect match to capture historical trends, seasonality, randomness, and other non-static behavior that humans miss.

**Get Closer To Your Dream of Becoming a Data Scientist with 150+ Solved End-to-End ML Projects**

# How to Build an ARIMA Model in Python

# Example Implementation)

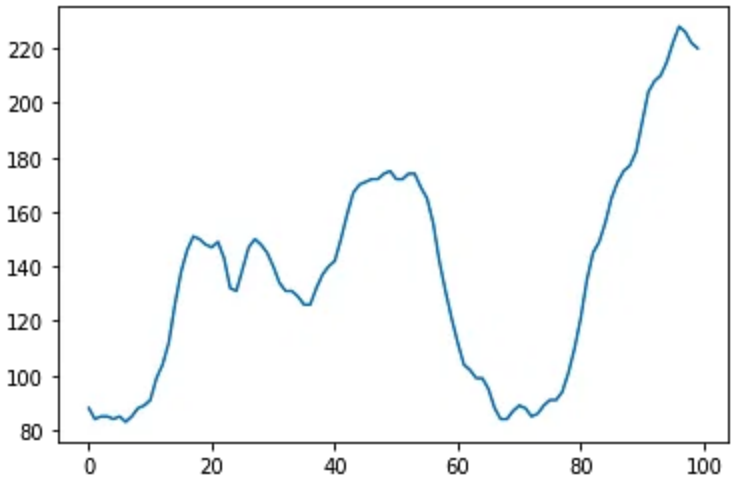
There are several ways to implement ARIMA in Python on any time series dataset. In this article, we work with the *statsmodel* library for implementing the ARIMA model Python code. The TSA sub-module of *statsmodel* provides an implementation of the ARIMA model as *statsmodel.tsa.arima_model.ARIMA*

We will work with the WWWUsage time-series dataset to keep things simple and visually intuitive. WWWUsage.csv contains 100 minutes' worth of information, with each row representing the number of users connected to the server in that minute.

```
1 df = pd.read_csv('wwwusage.csv', names=['value'], header=0)
2 print(f"Total samples: {len(df)}")
3 print(df.head())
```

```
Total samples: 100
   value
0     88
1     84
2     85
3     85
4     84
```

A simple line plot can show how users fluctuate with each passing minute but an overall upward trend.



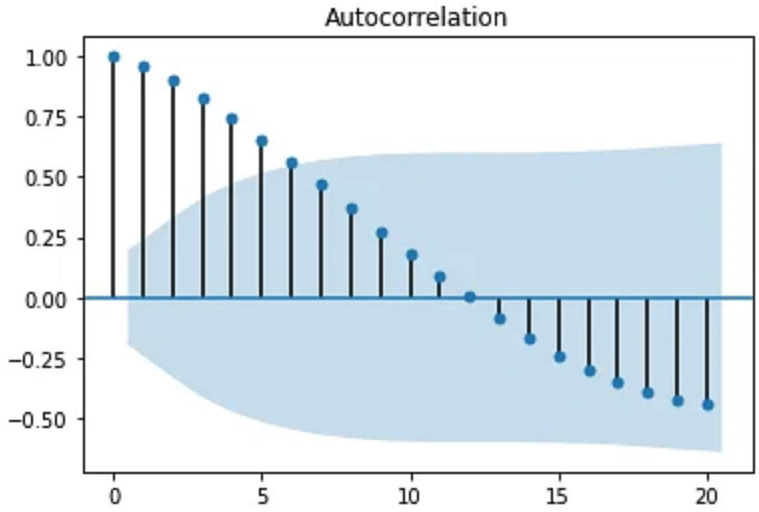

Before building the model, first, we need to take care of its assumptions about the data and determine the parameters of the *ARIMA(p, d, q)* model.

There are multiple approaches to set ARIMA parameters as they can be determined by either looking at the data properties or empirically by fitting the model and evaluating the performance. We will further look at the Autocorrelation function (ACF) plots and the Partial Autocorrelation function (ACF) plots to identify the overall correlation in the data. Moreover, computing the rolling mean and standard deviation or existing tests such as the Augmented Dickey Fuller or ADF test can be employed to determine the properties of the time series. We will look at these techniques further ahead. However, we still need to know how to set the three fundamental parameters of the ARIMA model.

## How to choose p, d, and q for the ARIMA Python Model?

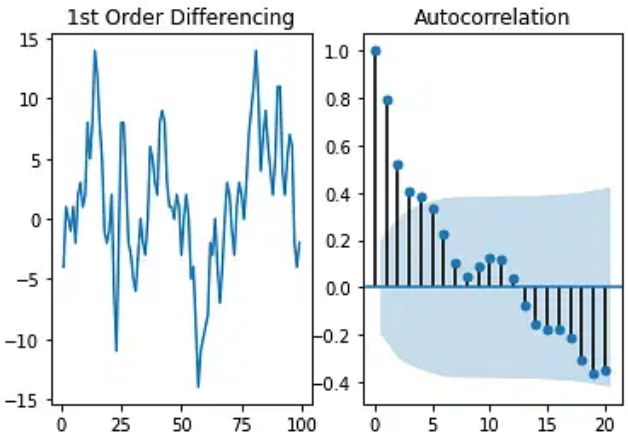
The first step would be to take care of the assumptions discussed above. For that, we need to determine the order of differencing "d." Let's first check the autocorrelation plot. The statsmodel package can help us with this-

```
1 from statsmodels.graphics.tsaplots import plot_acf
2 plot_acf(df.value)
```



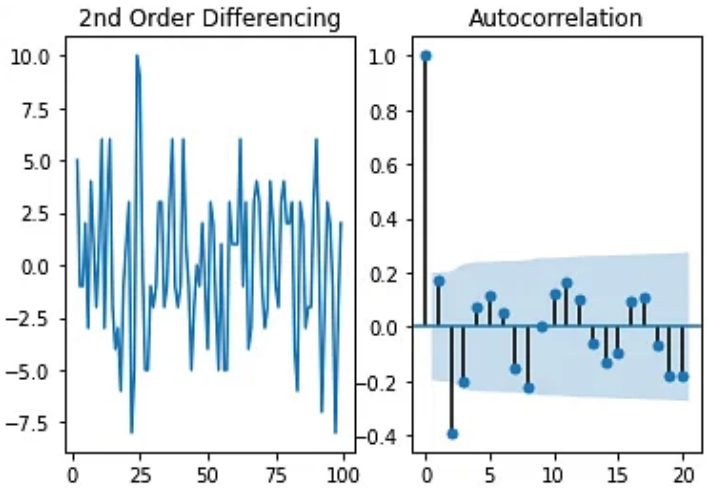

Clearly, the data is not ideal for the Python ARIMA model to directly start autoregressive training. So let's see how the differencing segment of ARIMA model Python makes the data stationary.

```
f = plt.figure()
ax1 = f.add_subplot(121)
ax1.set_title('1st Order Differencing')
ax1.plot(df.value.diff())

ax2 = f.add_subplot(122)
plot_acf(df.value.diff().dropna(), ax=ax2)
plt.show()
```




As seen above, first-order differencing shakes up autocorrelation considerably. We can also try 2nd order differencing to enhance the stationary nature.

```
f = plt.figure()
ax1 = f.add_subplot(121)
ax1.set_title('2nd Order Differencing')
ax1.plot(df.value.diff().diff())

ax2 = f.add_subplot(122)
plot_acf(df.value.diff().diff().dropna(), ax=ax2)
plt.show()
```




From the autocorrelation graph, we can decide if more differencing is needed. If collectively the

consecutive lags, more differencing might be needed. Conversely, if more data points are negative, the series is over-differenced.

### 1. What are the three main components of an ARIMA model?

- Autoregression, Integration, Moving Average
- Integration, Differencing, Smoothing
- Moving Average, Differencing, Smoothing
- Autoregression, Smoothing, Seasonal Adjustment

Previous 1 / 7 Next



However, a more mathematical test can be employed to determine which order is best for the data in question. The Augmented Dickey-Fuller test is one such measure that *statsmodel* readily provides. The ADF test aims to reject the null hypothesis that the given time-series data is non-stationary. It calculates the p-value and compares it with a threshold value or significance level of 0.05. If the p-value is less than this level, then the data is stationary; else, the differencing order is incremented by one.

```
from statsmodels.tsa.stattools import adfuller
result = adfuller(df.value.dropna())
print('p-value: ', result[1])

result = adfuller(df.value.diff().dropna())
print('p-value: ', result[1])

result = adfuller(df.value.diff().diff().dropna())
print('p-value: ', result[1])
```

```
p-value:  0.12441935447109487
p-value:  0.07026846015272707
p-value:  2.843428755547158e-17
```

As we see above, after the 2nd order differencing, the p-value drops beyond the acceptable threshold. Thus, we can consider the order of differencing (“d”) as 2. This corresponds well with the autocorrelation line graph seen above. However, the p-value for the 1st order is much closer to the threshold, so to be conservative, we will consider “d” as 1 and see how the model performs.

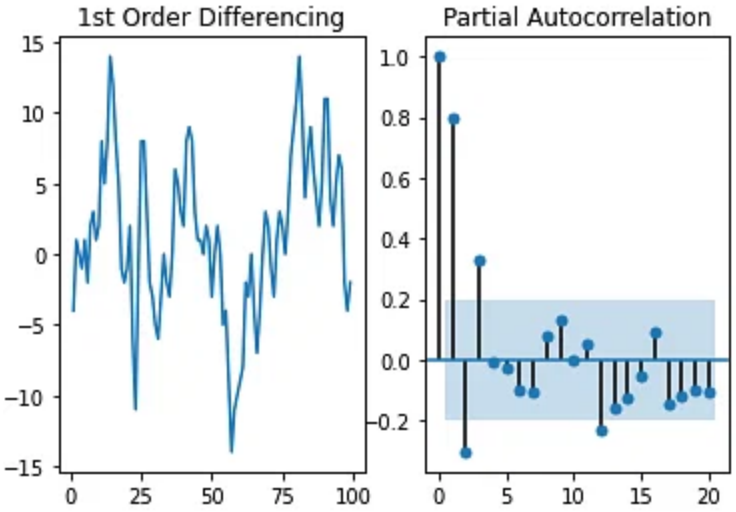
The next step in the ARIMA model is computing “p,” or the order for the autoregressive model. We can inspect the partial autocorrelation plot, which measures the correlation between the time-series data and a certain lag. Based on the presence or absence of correlation, we can determine whether the lag or order is needed or not.

Mathematically in the equation below, the partial autocorrelation of series and a lag would be that lag's coefficient (alpha) -

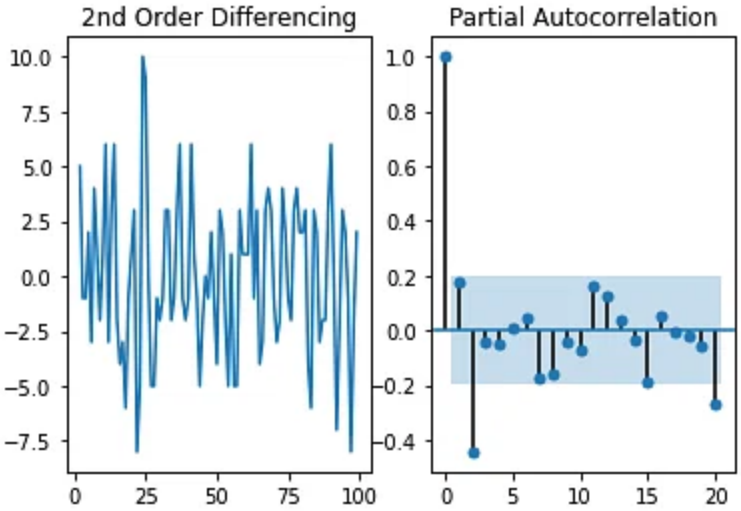
$$Y_t = \alpha_0 + \alpha_1 Y_{t-1} + \alpha_2 Y_{t-2} + \alpha_3 Y_{t-3}$$

Thus, we determine “p” based on the most significant lag in the partial autocorrelation plot. We can check the plot up to 2nd order difference to be sure.

```
f = plt.figure()
ax1 = f.add_subplot(121)
ax1.set_title('1st Order Differencing')
ax1.plot(df.value.diff())

ax2 = f.add_subplot(122)
plot_pacf(df.value.diff().dropna(), ax=ax2)
plt.show()
```




```
f = plt.figure()
ax1 = f.add_subplot(121)
ax1.set_title('2nd Order Differencing')
ax1.plot(df.value.diff().diff())

ax2 = f.add_subplot(122)
plot_pacf(df.value.diff().diff().dropna(), ax=ax2)
plt.show()
```



Finally, “q” can be estimated similarly by looking at the ACF plot instead of the PACF plot. Looking at the number of lags crossing the threshold, we can determine how much of the past would be significant enough to consider for the future. The ones with high correlation contribute more and would be enough to predict future values. From the plots above, the moving average (MA) parameter can be set to 2.

Thus, our final Python ARIMA model can be defined as *ARIMA(p=1, d=1,q= 2)*.



## How to Fit ARIMA in Python?

Using the ARIMA class from the statsmodels.tsa.arima_model module, we can feed the data and

```
from statsmodels.tsa.arima_model import ARIMA

arima_model = ARIMA(df.value, order=(1,1,2))
model = arima_model.fit()
print(model.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:                D.value   No. Observations:                   99
Model:                 ARIMA(1, 1, 2)   Log Likelihood                -253.790
Method:                       css-mle   S.D. of innovations              3.119
Date:                Sun, 09 Jan 2022   AIC                            517.579
Time:                        13:53:01   BIC                            530.555
Sample:                             1   HQIC                           522.829

==================================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------
const              1.1202      1.290      0.868      0.387      -1.409       3.649
ar.L1.D.value      0.6351      0.257      2.469      0.015       0.131       1.139
ma.L1.D.value      0.5287      0.355      1.489      0.140      -0.167       1.224
ma.L2.D.value     -0.0010      0.321     -0.003      0.998      -0.631       0.629
```

# How to Interpret ARIMA Python Example Results?

As seen above, the model summary provides several statistical measures to evaluate the performance of ARIMA model in Python. Moreover, we also know the coefficient values for each of the parameters. As we kept the value of the MA parameter or “q” as 2, we have two trained coefficients for MA and one for AR.

Other than that we see the scores such as Akaike Information Critera (AIC), Bayesian Information Criterion (BIC), Hannan-Quinn Information Criterion (HQIC), and the standard deviation of innovations (innovations are the difference of the real value at time t and the predicted value at that time).



# How to Get AIC for the ARIMA Forecast Python Example?

The Akaike Information Critera or AIC is a good measure for testing the goodness of how fit the

training and generalizing the ARIMA model. The AIC must be as low as possible. By displaying the model summary of the trained ARIMA in Python from statsmodel, we can check the AIC scores and other statistical performance measures.

To reduce AIC, we can try changing the p, q, and d values or using training techniques like k-cross-validation. For example, changing the value of “d” to 2 instead of 1 reduces the AIC to around 514.

```
from statsmodels.tsa.arima_model import ARIMA

arima_model = ARIMA(df.value, order=(1,2,2))
model = arima_model.fit()
print(model.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:               D2.value   No. Observations:                   98
Model:                 ARIMA(1, 2, 2)   Log Likelihood                -252.446
Method:                       css-mle   S.D. of innovations              3.130
Date:                Sun, 09 Jan 2022   AIC                            514.893
Time:                        16:54:19   BIC                            527.818
Sample:                             2   HQIC                           520.121

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0245      0.045      0.547      0.586      -0.063       0.112
ar.L1.D2.value 0.6487      0.089      7.301      0.000       0.475       0.823
ma.L1.D2.value -0.4739     0.096     -4.944      0.000      -0.662      -0.286
ma.L2.D2.value -0.5260     0.091     -5.757      0.000      -0.705      -0.347
```

However, measures including AIC, BIC, and HQIC rely significantly on the learned likelihood of the data. By changing “d” (among 0, 1, and 2), we effectively change the data distribution and, thus, the likelihood computation. It would not be a suitable method of determining what value of “d” we should use. So, the different AICs we obtained by changing “d” are not directly comparable. But still, we can see from the parameter mentioned in the above selection method that d=2 would be a better choice overall.

However, Information Criteria scores are excellent for determining the optimal values of “p” and “q.” For example, changing the value of q to 3 reduced the AIC by a unit.