# Forecasting Tesla's Stock Price Using Autoregression

Learn how to apply a fundamental time series modelling technique to Tesla's stock price using Python.

Nathan Thomas Aug 7 · 7 min read ★

Source: Bram Van Oost via Unsplash.

Tesla has been making waves in financial markets over the last few months. Previously named the most shorted stock in the US [1], Tesla's stock price has since catapulted the electric carmaker to a market capitalization of $278 billion [2]. Its latest quarterly results suggest that it is now available to be added to the S&P 500, which it is currently not a member of, despite being the 12th largest company in the US [3].

Amid market volatility, various trading strategies and a sense of "FOMO" (fear of missing out), predicting the returns of Tesla's stock is a difficult task. However, we are going to use Python to forecast Tesla's stock price returns using autoregression.

**Exploring the data**

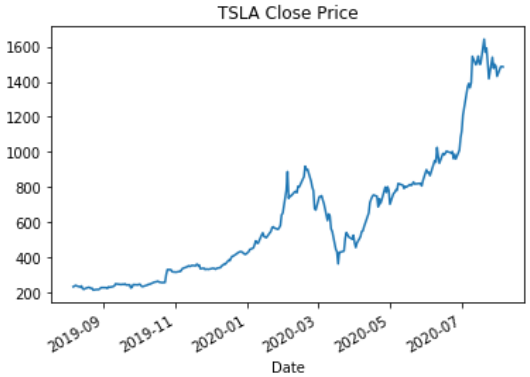
First, we need to import the data. We may use historical stock price data downloaded from Yahoo Finance. We're going to use the "Close" price for this analysis.

```
import pandas as pd
```

```
df = pd.read_csv("TSLA.csv", index_col=0, parse_dates=[0])
df.head()
```

| Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|
| **2019-08-06** | 231.880005 | 232.500000 | 225.750000 | 230.750000 | 230.750000 | 5564200 |
| **2019-08-07** | 226.500000 | 233.570007 | 225.800003 | 233.419998 | 233.419998 | 4776500 |
| **2019-08-08** | 234.449997 | 239.800003 | 232.649994 | 238.300003 | 238.300003 | 5274300 |
| **2019-08-09** | 236.050003 | 238.960007 | 233.809998 | 235.009995 | 235.009995 | 3898200 |
| **2019-08-12** | 232.990005 | 235.770004 | 228.750000 | 229.009995 | 229.009995 | 4663900 |

Source: Yahoo Finance.

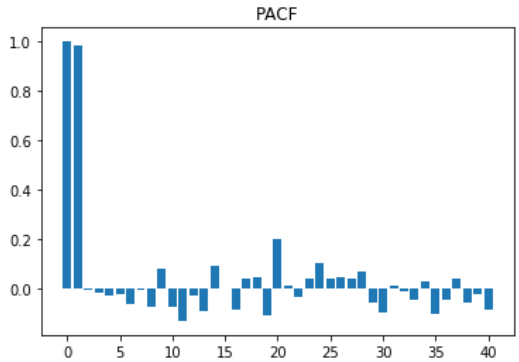



Source: Yahoo Finance.

To determine the order for the ARMA model, we can firstly plot a *partial autocorrelation function*. This gives a graphical interpretation of the amount of correlation between the dependent variable and the lags of itself, *which is not explained* by correlations at *all* lower-order lags.

From the PACF below, we can see that the significance of the lags cuts off after lag 1, which suggests we should use an autoregressive (AR) model [4].

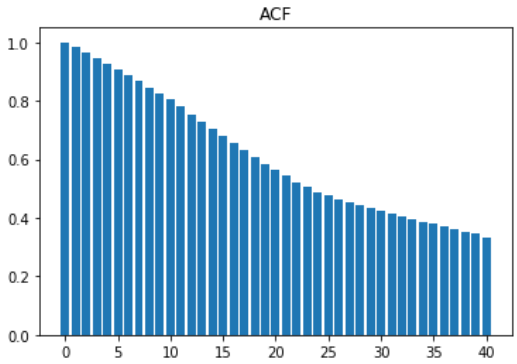
```
# Plot PACF
from statsmodels.tsa.stattools import acf, pacf
plt.bar(x=np.arange(0,41), height=pacf(df.Close))
plt.title("PACF")
```




Finite & cuts off after lag 1, so AR.

When plotting the autocorrelation function, we get a slightly different result. The series is infinite and slowly damps out, which suggests an AR or ARMA model [4]. Taking both the PACF and the ACF into account, we are going to use an AR model.

```
#Plot ACF
plt.bar(x=np.arange(0,41), height=acf(df.Close))
plt.title("ACF")
```




Infinite and damps out so AR/ARMA.

**Pre-processing the data**

Before we run the model we must make sure we are using *stationary data*. Stationarity refers to a characteristic in which the way the data moves doesn't change over time. Looking at the raw stock price seen earlier in the article, it is clear that the series is not stationary. We can see this as the stock price increases over time in a seemingly exponential manner.

Therefore, to make the series stationary we *difference* the series, which essentially means to **subtract today's value from tomorrow's value.** This results in the series revolving around a constant mean (0), giving us the *stock returns* instead of the stock price.

We are also going to lag the differenced series by 1, which **brings yesterday's value forward to today.** This is so we can obtain our AR term (Yt-1).

After putting these values into the same DataFrame, we split the data into training and testing sets. In the code, the data is split roughly into 80:20 respectively.

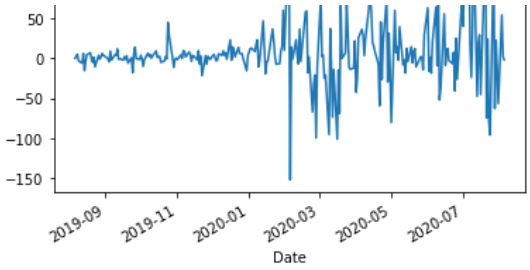
```
# Make the data stationary by differencing
tsla = df.Close.diff().fillna(0)

# Create lag
tsla_lag_1 = tsla.shift(1).fillna(0)

# Put all into one DataFrame
df_regression = pd.DataFrame(tsla)
df_regression["Lag1"] = tsla_lag_1

# Split into train and test data
df_regression_train = df_regression.iloc[0:200]
df_regression_test = df_regression.iloc[200:]

tsla.plot()
```




Differenced Series. Source: Yahoo Finance.

**Forming the AR model**

Now, how many values should we use to predict the next observation? Using all the past 200 values may not give a good estimate as intuitively, stock price activity from 200 days ago is unlikely to have a significant effect on today's value as numerous factors may have changed since then. This could include earnings, competition, season and more. Therefore, to find the optimal window of observations to use in the regression, one method we can use is to run a regression with an *expanding window.* This method, detailed in the code below, runs a regression with one past observation, recording the r-squared value (goodness-of-fit), and then repeats this process, **expanding past observations by 1 each time.** For economic interpretation, I've set the limit on the size of the window at 30 days.

```
# Run expanding window regression to find optimal window

n = 0
rsquared = []

while n<=30:

    y = df_regression_train["Close"].iloc[-n:]
    x = df_regression_train["Lag1"].iloc[-n:]
    x = sm.add_constant(x)

model = sm.OLS(y,x)
    results = model.fit()

rsquared.append(results.rsquared)

n +=1
```

Looking at the r-squared plot of each iteration, we can see than it is high around 1–5 iterations, and also has a peak at 13 past values. It may seem tempting to choose one of the values between 1 and 5, however, the very small sample size will likely mean that out regression is *statistically biased*, so wouldn't give us the best result. Therefore let's choose the second peak at 13 observations as this is a more sufficient sample size, which gives an r-squared of around 0.437 (i.e. model explains 43% of the variation in the data).

Observation

R-squared plot.

### Running the AR model on the training data

The next step is to use our window of 13 past observations to fit the AR(1) model. We may do this using the OLS function in statsmodels. Code below:

```
# AR(1) model with static coefficients

import statsmodels.api as sm
y = df_regression_train["Close"].iloc[-13:]
x = df_regression_train["Lag1"].iloc[-13:]
x = sm.add_constant(x)

model = sm.OLS(y,x)
results = model.fit()
results.summary()
```

OLS Regression Results

| | | | |
|---|---|---|---|
| Dep. Variable: | Close | R-squared: | 0.437 |
| Model: | OLS | Adj. R-squared: | 0.386 |
| Method: | Least Squares | F-statistic: | 8.551 |
| Date: | Thu, 06 Aug 2020 | Prob (F-statistic): | 0.0138 |
| Time: | 18:09:01 | Log-Likelihood: | -53.741 |
| No. Observations: | 13 | AIC: | 111.5 |
| Df Residuals: | 11 | BIC: | 112.6 |
| Df Model: | 1 | | |
| Covariance Type: | nonrobust | | |

| | coef | std err | t | P>\|t\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| const | 9.6456 | 4.564 | 2.114 | 0.058 | -0.399 | 19.690 |
| Lag1 | -0.4268 | 0.146 | -2.924 | 0.014 | -0.748 | -0.106 |

| | | | |
|---|---|---|---|
| Omnibus: | 0.302 | Durbin-Watson: | 1.288 |
| Prob(Omnibus): | 0.860 | Jarque-Bera (JB): | 0.198 |
| Skew: | 0.239 | Prob(JB): | 0.906 |
| Kurtosis: | 2.629 | Cond. No. | 31.3 |

Regression output from the AR(1) model (training data).

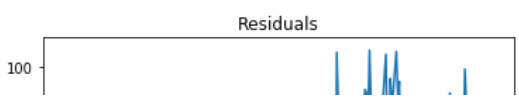
As we can see in the statistical summary, the p-value of both the constant and the first lag is significant at the 10% significance level. Looking at the sign of the coefficients, the positive sign on the constant suggests that, all else being equal, stock price returns should be positive. Also, the negative sign on the first lag suggests that the past value of the stock return is lower than today's value, ceteris paribus, which also **maintains the narrative that stock returns increase over time.**

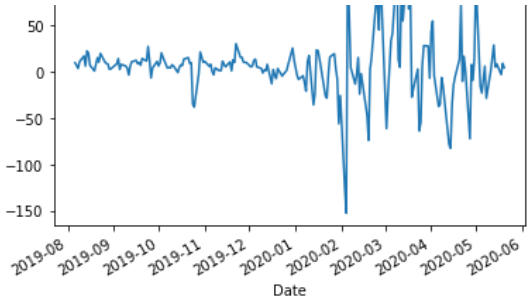
Great, now let's use those coefficients to find the fitted value for Tesla's stock returns so we can plot the model against the original data. Our model may now be specified as:

$$Y_t = 9.6456 - 0.4268Y_{t-1} + \varepsilon$$

Our AR(1) equation.

### Plot Residuals (Actual — Fitted)





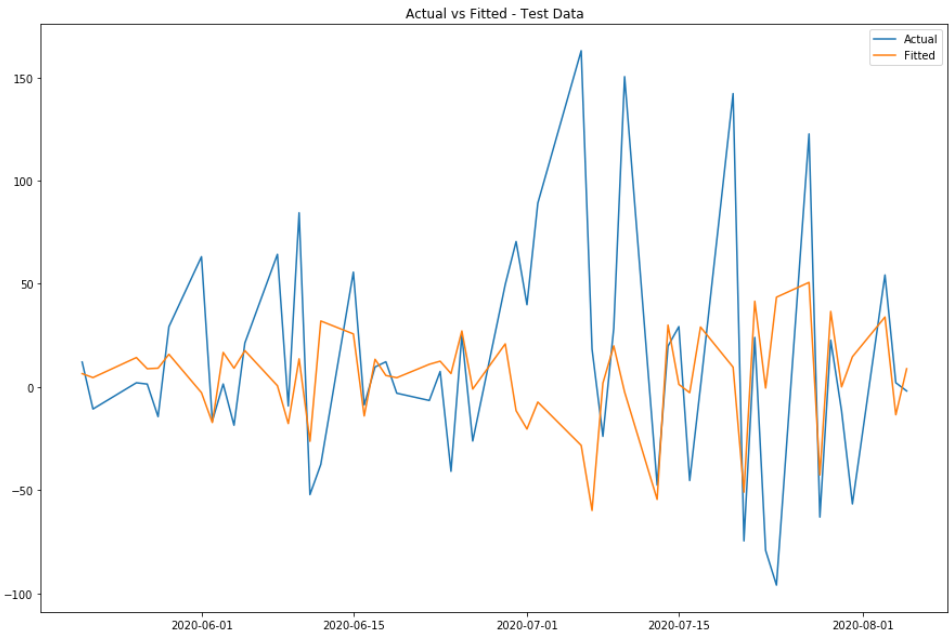

Residuals (training data)

The residuals suggest that the model performs better in 2019, but in 2020 as volatility increased, the model performed considerable worse (residuals are larger). This is intuitive as the volatility experienced in the March 2020 selloff had a large impact on US stocks, while the quick and sizeable rebound was particularly felt by tech stocks. This, along with the increased betting on Tesla stock by retail traders on platforms such as Robinhood has increased price volatility, thus making it harder to predict.

Given these factors, along with our previous r-squared of around 43%, we would not expect our AR(1) model to predict the exact stock return. Instead, we can test the model's accuracy by calculating its "hit rate", **i.e. when our model predicted a positive value and the actual value was also positive**, and vice versa. Summing up instances of true positives and true negatives, the accuracy of our mode 55%, which is fairly good for this simple model.

**Fit model to the test data**

Now, let's apply the same methodology to the test data to see how our model performs *out-of-sample.*




Actual vs Fitted (test data).

```
# Calculate hit rate
true_neg_test = np.sum((df_2_test["Fitted Value"] <0) &
(df_2_test["Actual"] <0))
true_pos_test = np.sum((df_2_test["Fitted Value"] >0) &
(df_2_test["Actual"] >0))

accuracy = (true_neg_test + true_pos_test)/len(df_2_test)
print(accuracy)
```

```
# Output: 0.6415
```

Our hit rate has improved to 64% when applying the model to the test data, which is a promising improvement! Next steps to improve its accuracy may include running a rolling regression, where coefficients change with each iteration, or perhaps incorporating a moving average (MA) element to the model.

***Thanks for reading!*** *Please feel free to leave any comments for any insights you may have. The full Jupyter Notebook which contains the source code I used to do this project can be found on my Github Repository.*

References

[1] Reinicke, Carmen (2020). *"Tesla just became the most shorted stock in the US, again (TSLA)".* Markets Insider. Available at: https://markets.businessinsider.com/news/stocks/tesla-stock-most-shorted-us-beats-apple-highest-short-interest-2020-1-1028823046

[2] Yahoo Finance, as of 6 August 2020.

[3] Stevens, Pippa (2020). *"Tesla could soon join the S&P 500 — but inclusion isn't automatic, even with a full year of profitability".* Available at: https://www.cnbc.com/2020/07/21/tesla-isnt-a-gurantee-for-the-sp-500-even-with-year-of-profits.html

[4] Johnson J., Dinardo J. (1997). *"Econometric Methods, Fourth Edition".*

*Disclaimer: All views expressed in this article are my own, and are* ***not*** *in any way associated with any financial entity. I am not a trader and am not making any money from the methods used in this article. This is not financial advice.*


Download on the
App Store


GET IT ON
Google Play