Stock Price Prediction – Capstone 3

By Sebastian Ivan

Final project Report

In this project, the stock price of three Nasdaq listed companies was analyzed. Three models and various scenarios were considered:

- ARIMA (AutoRegressive Integrated Moving Average models are powerful for understanding and predicting future points in the series.)
- Random Forest
- Decision Tree

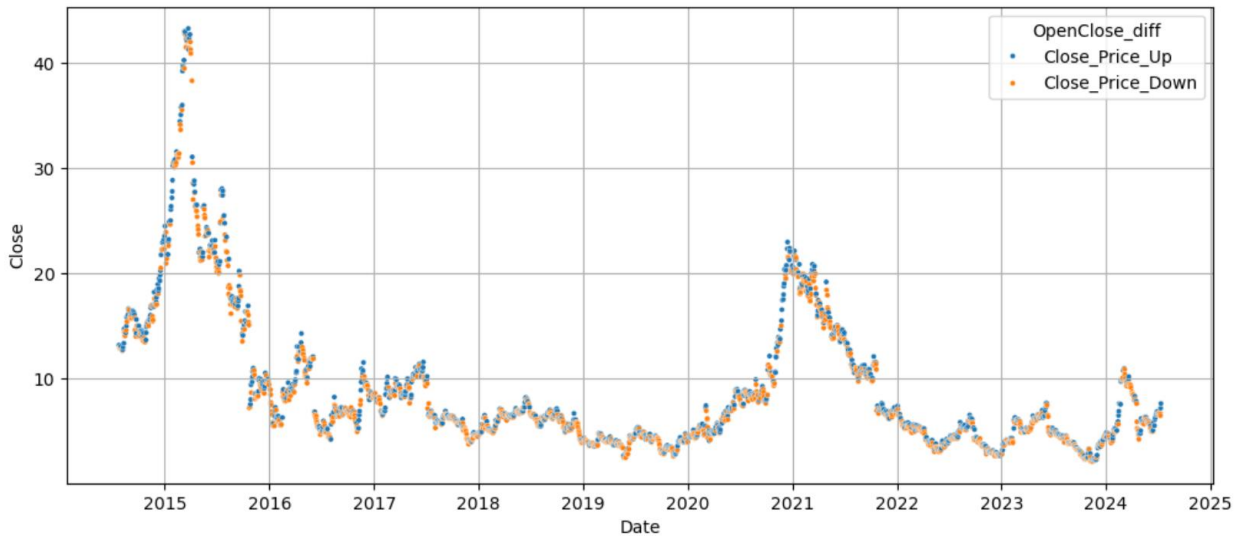
The time series data was sourced from Yahoo finance website. In the preliminary data wrangling stage, the data was converted into a dataframe, cleaned and saved for further processing.

The three companies analysed are:

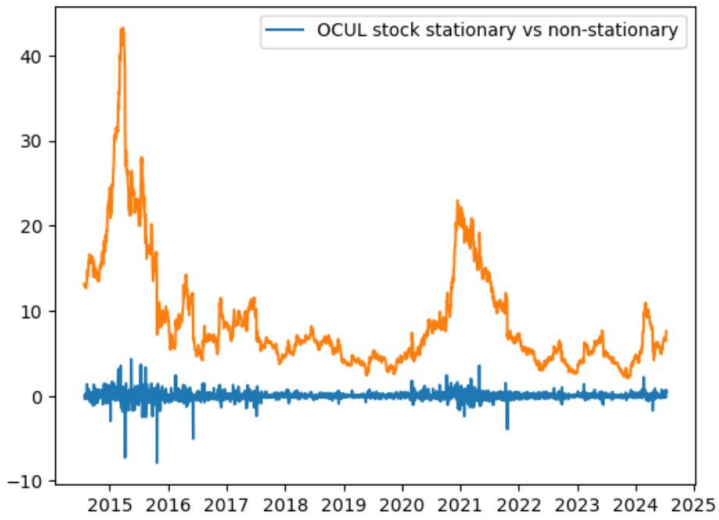
- OCUL Therapeutics, sticker symbol OCUL
- Solid Biosciences, sticker symbol SLDB
- Urogen Pharma, sticker symbol URGN

When close price is higher than open price the overall trend rises and a price momentum is created.

The blue markers in plot below are clustered together on upper trends, while the red markers are clustered together in down trends



Data needs to be stationary to be able to build models to predict prices. This is done by taking the first difference of the Close price of stocks. Below is a visualization of transformed data.



Computing percent changes and differences in time series is a common task in data analysis, especially for financial data, sales data, or any other sequential data. These method can help analyze trends and fluctuations in the time series data effectively. The percent change method converts prices to returns.

```
# Use pct_change() method to convert prices to returns

df['Close'].pct_change()

```

```
Date
2022-07-25         NaN
2022-07-26    0.011737
2022-07-27    0.032483
2022-07-28   -0.033708
2022-07-29    0.041860
                ...
2024-07-16   -0.039063
2024-07-17   -0.019164
2024-07-18   -0.014802
2024-07-19   -0.051082
2024-07-22    0.025966
Name: Close, Length: 1503, dtype: float64
```

To avoid spurious correlation, when looking at the correlation of say, two stocks, one should look at the correlation of their returns, not their levels.

Simple Linear Regressions - time series analysis

A simple linear regression finds the slope, beta, and intercept, alpha, of a line that's the best fit between a dependent variable, y, and an independent variable, x. The x's and y's can be two time series.

A linear regression is also known as Ordinary Least Squares, or OLS, because it minimizes the sum of the squared distances between the data points and the regression line.

```
stats.linregress(df_OCUL['OCUL_Close'], df_SLDB['SLDB_Close'])
```

```
.inregressResult(slope=0.9660909550339072, intercept=1.8023922669163879, rvalue=0.6482834070098535, pvalue=4.616292551928441e-6
l, stderr=0.050794326885850345, intercept_stderr=0.27021120315302516)
```

I need to add a column of ones as a dependent, right hand side variable. The reason I have to do this is because the regression function assumes that if there is no constant column, then I want to run the

regression without an intercept. By adding a column of ones, statsmodels will compute the regression coefficient of that column as well, which can be interpreted as the intercept of the line.

The statsmodels method "add constant" is a simple way to add a constant.

The first argument of the statsmodel regression is the series that represents the dependent variable, y, and the next argument contains the independent variable or variables. In this case, the dependent variable is the OCUL returns and the independent variables are the constant and SLDB returns. The method "fit" runs the regression and results are saved in a class instance called results.

```
df_3 = sm.add_constant(df_3)  # this automatically adds a column of 1's named const
df_3.head()
```

Date	const	OCUL_Close	OCUL_ret	SLDB_Close	SLDB_ret	URGN_Close	URGN_ret
2022-07-25	1.0	4.26	NaN	10.440	NaN	8.14	NaN
2022-07-26	1.0	4.31	0.011737	10.215	-0.021552	8.06	-0.009828
2022-07-27	1.0	4.45	0.032483	10.050	-0.016153	8.32	0.032258
2022-07-28	1.0	4.30	-0.033708	10.740	0.068657	8.16	-0.019231
2022-07-29	1.0	4.48	0.041860	10.125	-0.057263	7.84	-0.039216

Notice that the first row of the return series is NaN. Each return is computed from two prices, so there is one less return than price. To delete the first row of NaN's, use the pandas method "dropna".

OLS Regression Results

Dep. Variable:	OCUL_ret	**R-squared:**	0.042
Model:	OLS	**Adj. R-squared:**	0.040
Method:	Least Squares	**F-statistic:**	21.59
Date:	Fri, 09 Aug 2024	**Prob (F-statistic):**	4.34e-06
Time:	12:27:40	**Log-Likelihood:**	741.19
No. Observations:	500	**AIC:**	-1478.
Df Residuals:	498	**BIC:**	-1470.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0025	0.002	1.018	0.309	-0.002	0.007
SLDB_ret	0.1883	0.041	4.646	0.000	0.109	0.268

Omnibus:	148.460	**Durbin-Watson:**	1.960
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1027.104
Skew:	1.103	**Prob(JB):**	9.27e-224
Kurtosis:	9.666	**Cond. No.**	16.5

The intercept is 0.0025 (also referred to as beta), and the slope is 0.1883

Autocorrelation

Autocorrelation is the correlation of a single time series with a lagged copy of itself. It's also called "serial correlation". Often, when we refer to a series's autocorrelation, we mean the "lag-one" autocorrelation. So when using daily data, for example, the autocorrelation would be the correlation of the series with the same series lagged by one day.

What does it mean when a series has a positive or negative autocorrelation? With financial time series, when returns have a negative autocorrelation, we say it is "mean reverting".

Alternatively, if a series has positive autocorrelation, we say it is "trend-following".

Since stocks have historically had negative autocorrelation over horizons of about a week, one popular strategy is to buy stocks that have dropped over the last week and sell stocks that have gone up.

The "rule" argument indicates the desired frequency. 'W' stands for weekly, M' stands for monthly. You can use any number of functions after resample. Here we used last for the last date of the period. But you could use first date of the period, or even an average over the period.

```
df_3_weekly = df_3.resample(rule='W').last()
autocorrelation_OCUL = df_3_weekly['OCUL_ret'].autocorr()
autocorrelation_SLDB = df_3_weekly['SLDB_ret'].autocorr()
autocorrelation_URGN = df_3_weekly['URGN_ret'].autocorr()
```

```
The autocorrelation_OCUL for weekly frequency is:  0.07456910092830552
The autocorrelation_SLDBL for weekly frequency is:  0.04916164029942228
The autocorrelation_URGN for weekly frequency is:  -0.07335996146896823
```

Note the weekly time series is positively autocorrelated for OCUL and SLDB, and negative for URGN.

```
df_3_monthly = df_3.resample(rule='ME').last()
autocorrelation_OCUL = df_3_monthly['OCUL_ret'].autocorr()
autocorrelation_SLDB = df_3_monthly['SLDB_ret'].autocorr()
autocorrelation_URGN = df_3_monthly['URGN_ret'].autocorr()
```

```
The autocorrelation_OCUL for monthly frequency is:  -0.04058287412161152
The autocorrelation_SLDBL for monthly frequency is:  0.21287438846810985
The autocorrelation_URGN for monthly frequency is:  -0.11325764651424772
```

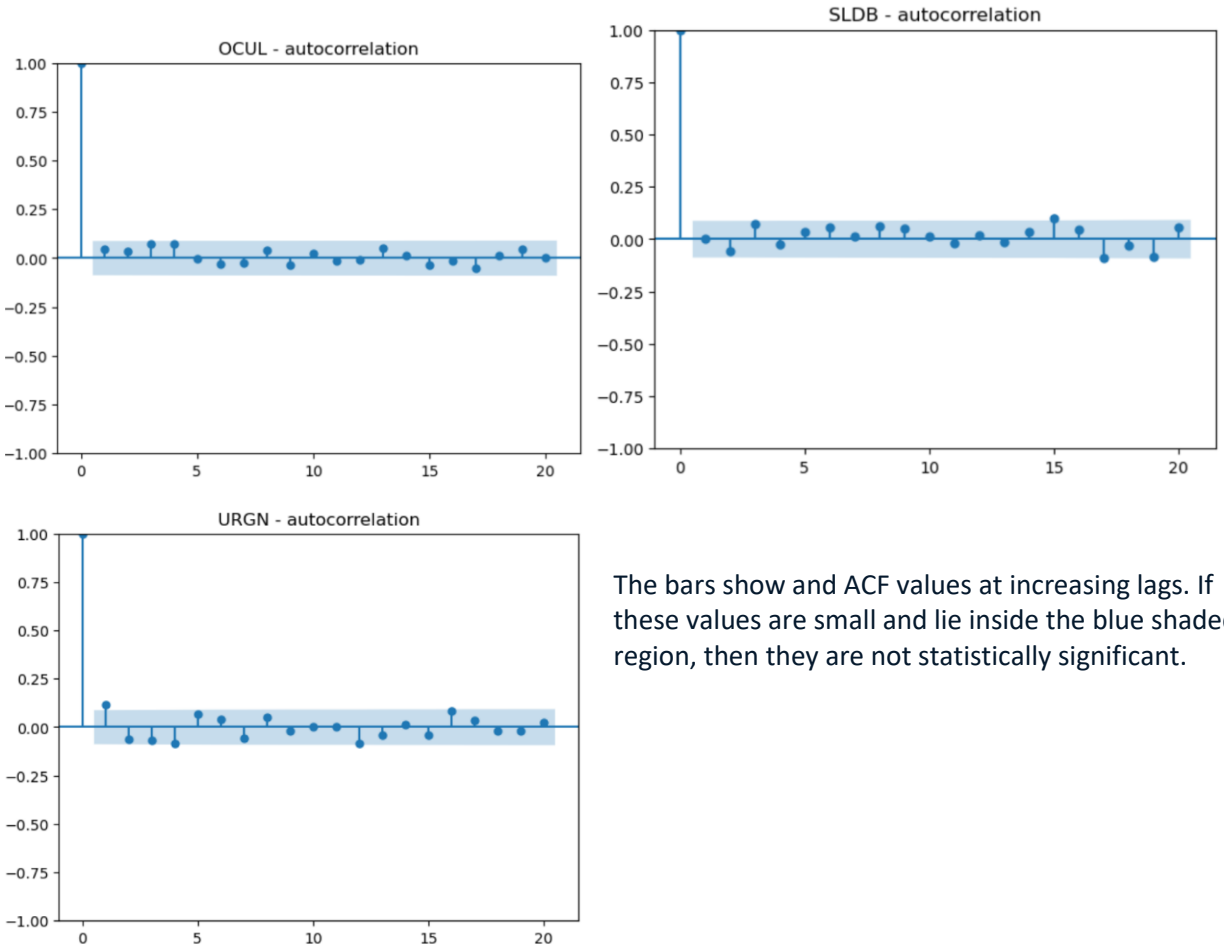
Note the monthly time series is positively autocorrelated for SLDB, and negative for OCUL and URGN.

ACF - Autocorrelation Function

shows not only the lag-one autocorrelation as shown before, but the entire autocorrelation function for different lags. Any significant non-zero autocorrelations implies that the series can be forecast from the past.

This autocorrelation function implies that you can forecast the next value of the series from the last two values, since the lag-one and lag-two autocorrelations differ from zero. The ACF can also be useful for selecting a parsimonious model for fitting the data.

Here is the ACF plot that contains confidence intervals for each lag, which is the blue region in the figure.



The bars show and ACF values at increasing lags. If these values are small and lie inside the blue shaded region, then they are not statistically significant.

Statistical Test for Random Walk

To test whether a series like stock prices follows a random walk, you can regress current prices on lagged prices.

- If the slope coefficient, beta, is not significantly different from one, then we cannot reject the null hypothesis that the series is a random walk.
- However, if the slope coefficient is significantly less than one, then we can reject the null hypothesis that the series is a random walk.

An identical way to do that test is to regress the difference in prices on the lagged price, and instead of testing whether the slope coefficient is 1, now we test whether it is zero.

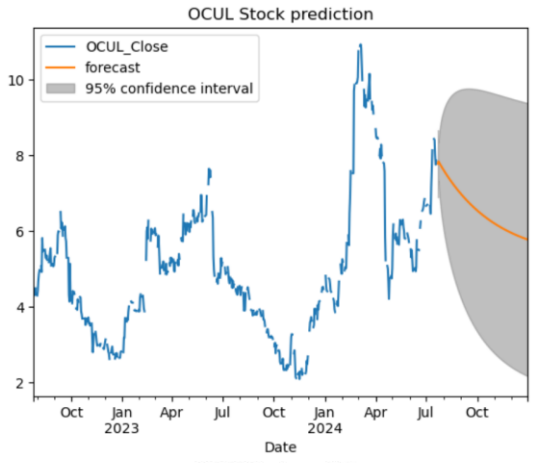
This is called the **"Dickey-Fuller"** test. If you add more lagged prices on the right hand side, then it's called the **Augmented Dickey-Fuller** test, **ADF**.

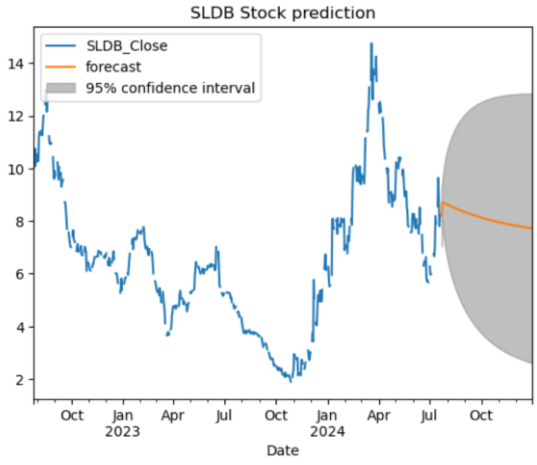
With the ADF test, the "null hypothesis" (the hypothesis that we either reject or fail to reject) is that the series follows a random walk. Therefore, a low p-value (say less than 5%) means we can reject the null hypothesis that the series is a random walk.

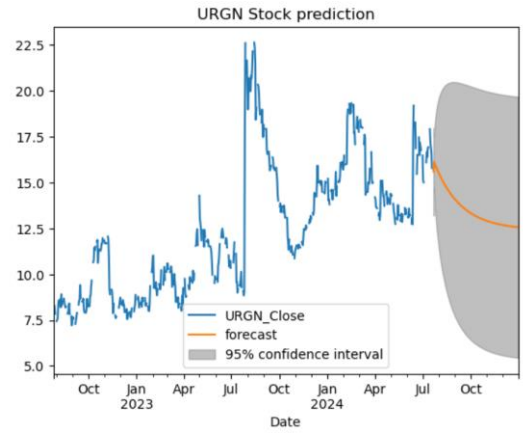
- p-value for OCUL stock prices is 0.22 > 0.05, therefore we reject the null hypothesis that the series is a random walk
- p-value for SLDB stock prices is 0.38 > 0.05, therefore we reject the null hypothesis that the series is a random walk
- p-value for URGN stock prices is 0.15 > 0.05, therefore we reject the null hypothesis that the series is a random walk

Forecasting With an AR Model

To plot forecasts, both in-sample and out-of-sample, I use statsmodels function called *plot_predict*. The first argument in the *plot_predict* function is the result from the fitted model. I also give *plot_predict* the starting and ending data points for forecasting. If the index of the data is a DatetimeIndex object as it is here, I can pick dates for the start and end dateIn this plot, the confidence interval gets wider the farther out the forecast is.





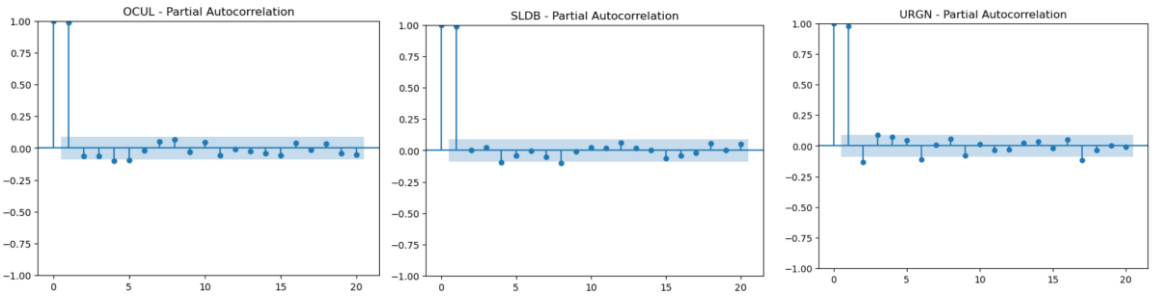


Choosing the right model

The order of an AR(p) model will usually be unknown. Here are two techniques that can help determine the order of the AR model:

-The Partial Autocorrelation Function PACF
-and the Information Criteria

The **Partial Autocorrelation Function PACF** measures the incremental benefit of adding another lag. The more parameters in a model, the better the model will fit the data. But this can lead to overfitting of the data.



-for an AR(1) model, only the lag-1 PACF is significantly different from zero.

-the number of significant lags for the PACF indicate the order of the AR model, in this case is 1.

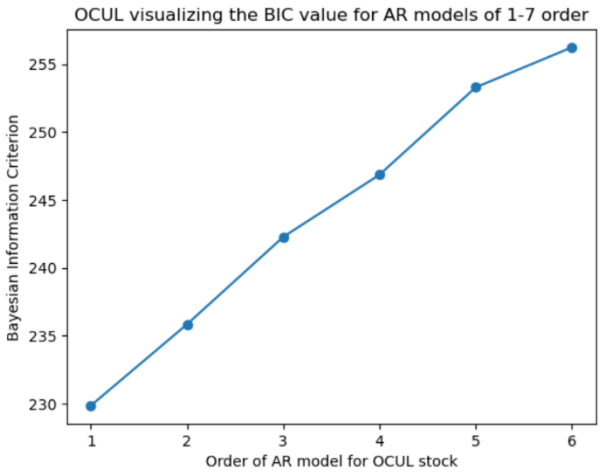
The **information criteria** adjusts the goodness-of-fit of a model by imposing a penalty based on the number of parameters used. Two common adjusted goodness-of-fit measures are called the Akaike Information Criterion **AIC** and the Bayesian Information Criterion **BIC**.

Follow the same procedure from before to fit the data to a model AR(1). I can get the full output using summary or just the AR parameters using the params attribute.

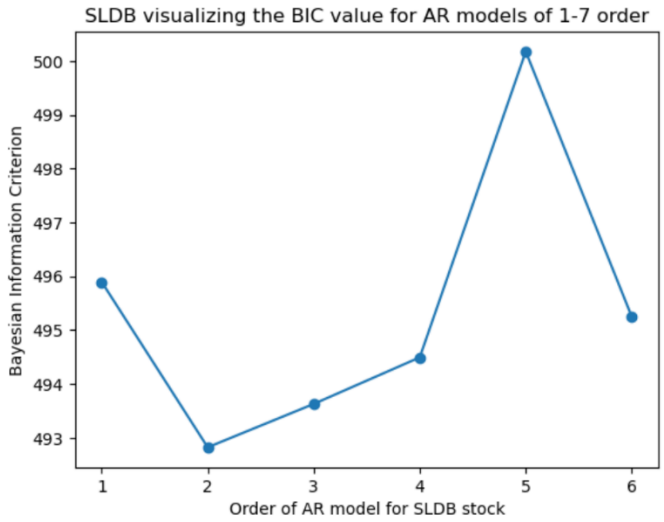
I can also get the **AIC** or **BI**C using those attributes.

In practice, the way to use the Bayesian information criterion is to fit several models, each with a different number of parameters, and choose the one with the lowest information criterion.

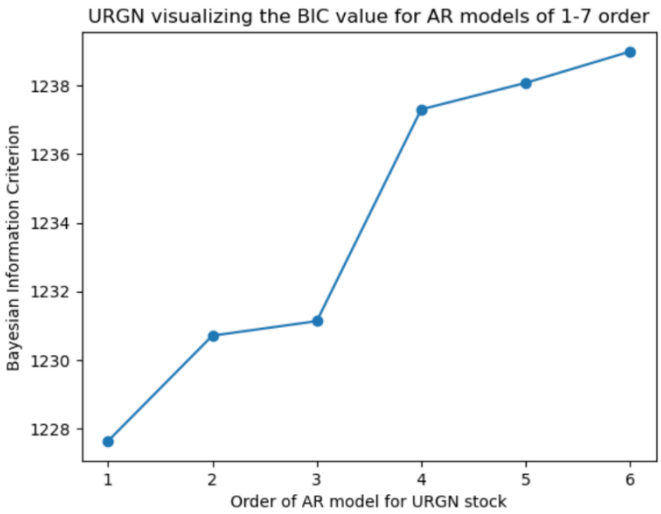
```
The AIC - Akaike information criterion for OCUL is: 216.06393762277145
The BIC - Bayesian information criterion for OCUL is: 229.8348407673615

The AIC - Akaike information criterion for SLDB is: 482.12299622768285
The BIC - Bayesian information criterion for SLDB is: 495.8938993722729

The AIC - Akaike information criterion for URGN is: 1213.8599041488242
The BIC - Bayesian information criterion for URGN is: 1227.6308072934141
```



-the BIC minimum for OCUL is reached at p=1, corresponding to an AR(1) model



-the BIC minimum for SLDB is reached at p=2, corresponding to an AR(2) model



-the BIC minimum for URGN is reached at p=1, corresponding to an AR(1) model

Moving Average Model - MA¶

In an **MA** model, today's value equals a mean plus noise, plus a fraction theta of yesterday's noise. Since there is only one lagged error on the right hand side, this is called an MA model of order 1, or simply an MA(1) model. If the MA parameter, theta, is zero, then the process is white noise. MA models are stationary for all values of theta.

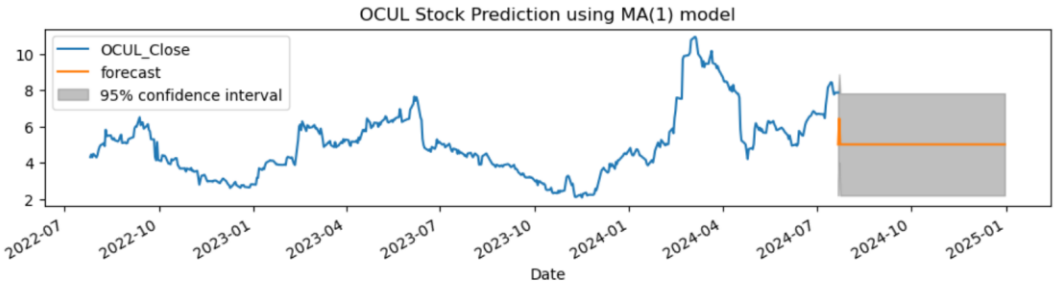
Estimating an MA Model

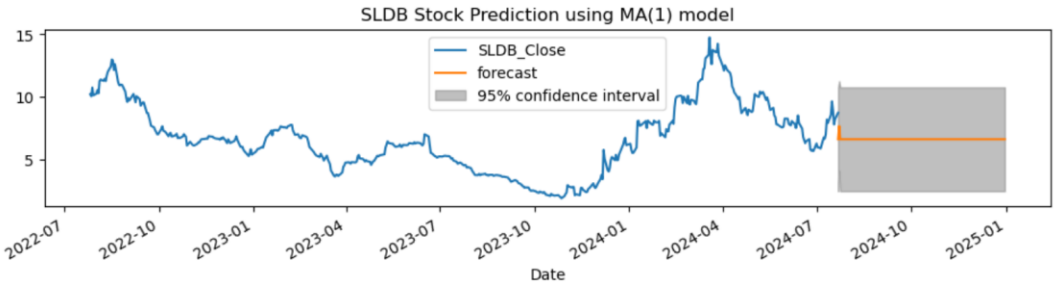
The same module that I used to estimate the parameters of an AR model can be used to estimate the parameters of an MA model.

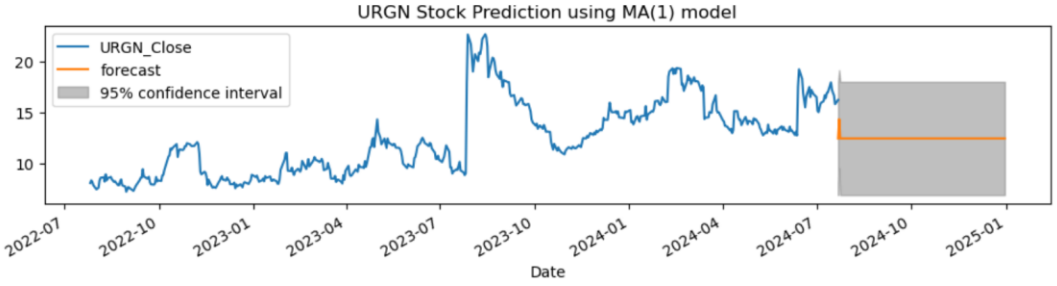
Create an instance of that class called mod, with the arguments being the data that I'm trying to fit, and the order of the model. However, now the order is (0,0,1), for an MA(1), not (1,0,0) as it was for an AR(1). And as before with an AR model, once I instantiate the class, I then use the method fit to estimate the model, and store the results in result.

Forecasting an MA model

The MA model expresses the current value of the series as a linear combination of past error terms (or shocks). This is different from autoregressive (AR) models, which use past values of the series itself.







ARMA models

An ARMA model is a combination of an AR and MA model. The time series is regressed on the previous values and the previous shock terms. This is an ARMA(1,1) model. More generally we use ARMA(p,q) to define an ARMA model.

The **p** tells us the order of the *autoregressive* part of the model and the **q** tells us the order of the *moving-average* part.

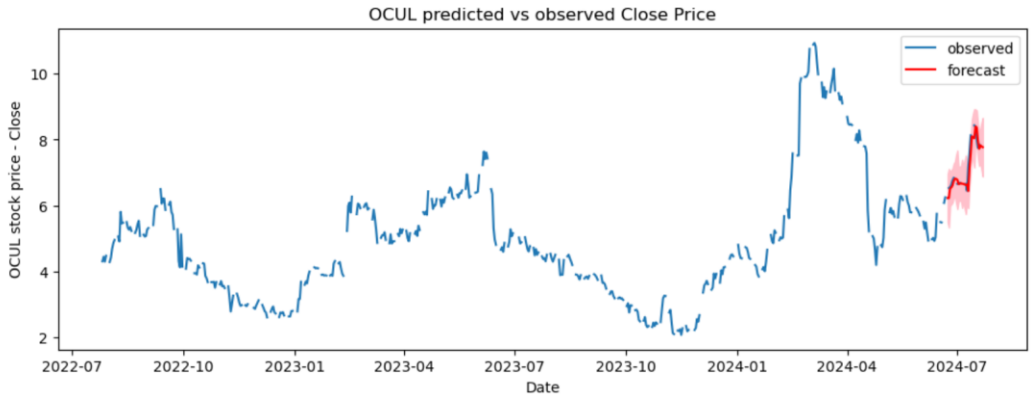
Predict the next value One-step-ahead predictions:

In the time period we have data for, we can make lots of these predictions in-sample; using the previous series value to estimate the next ones. This is called a one-step-ahead prediction. This allows us to evaluate how good our model is at predicting just one value ahead.

We can use a fitted ARIMA model to make these predictions. Starting from a ARIMA fitted results object, we can use its ***get_prediction*** method to generate in sample predictions.

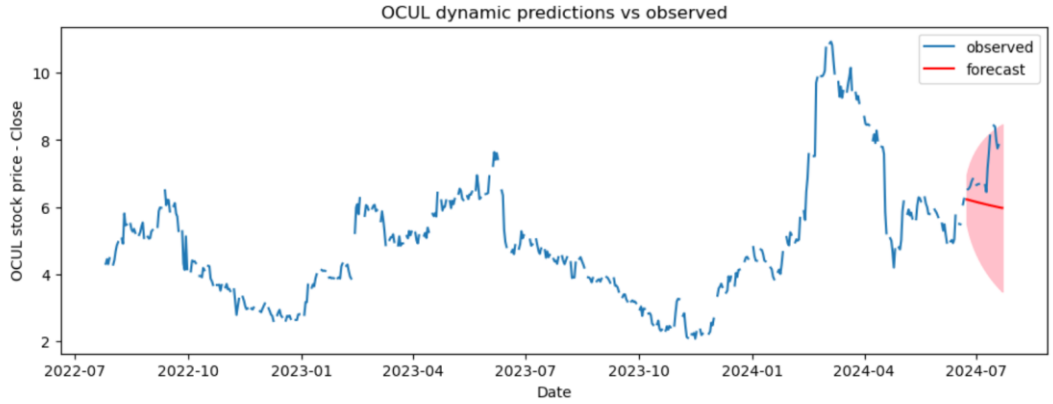
We set the start parameter as a negative integer stating how many steps back to begin the forecast. Setting start to -25 means we make predictions for the last 25 entries of the training data

Plotting predictions



Making dynamic predictions

The only difference is that in the ***get-predictions*** method, we set the parameter dynamic equals true. Everything else is exactly as before.



Model Diagnostic

To diagnose our model we focus on the residuals to the training data. The residuals are the difference between the our model's one-step-ahead predictions and the real values of the time series. We want to know, on average, how large the residuals are and so how far our predictions are from the true values. To answer this we can calculate the mean absolute error of the residuals.

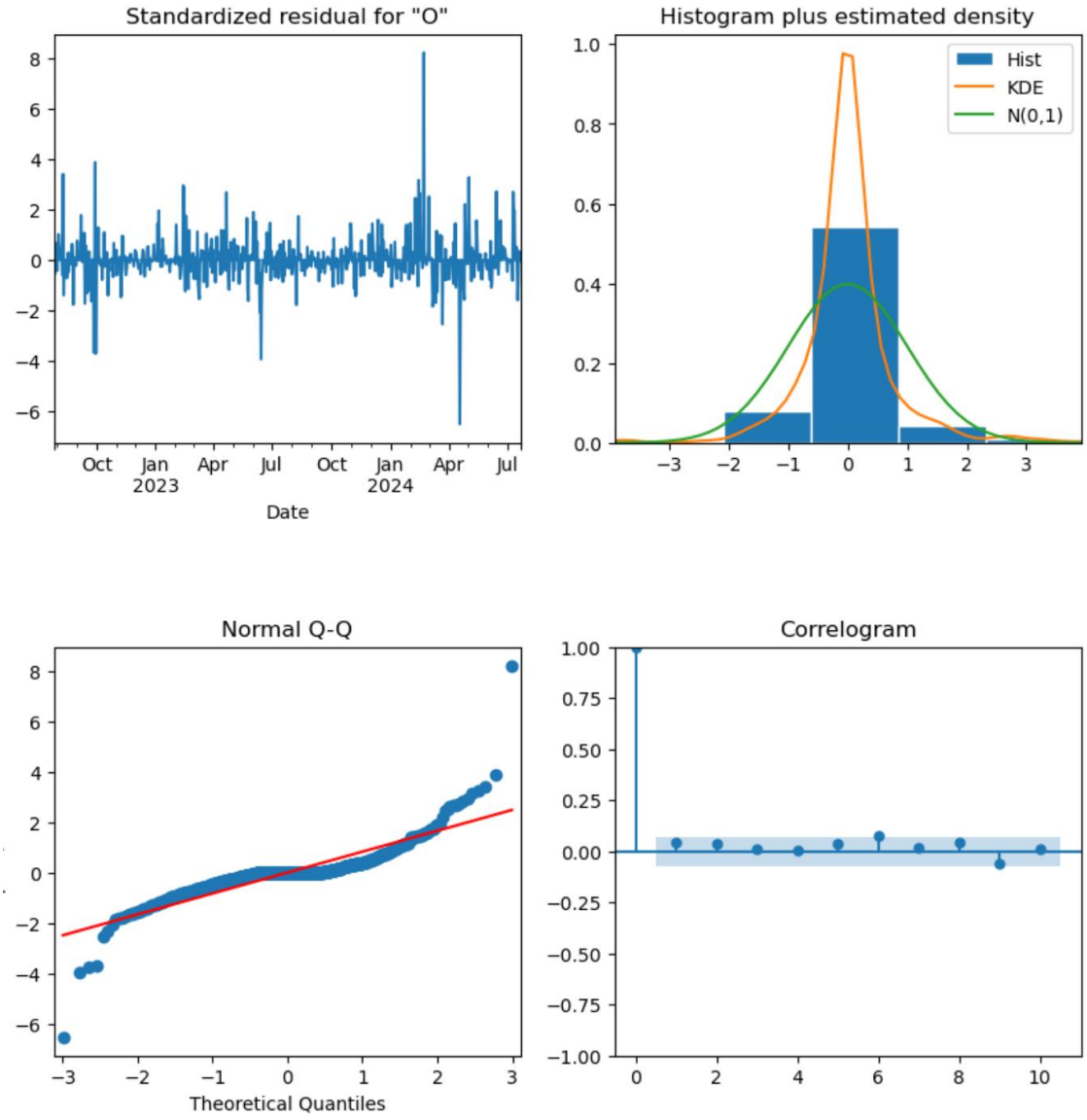
```
1 mae = np.mean(np.abs(residuals_OCUL))
2 mae
```

```
0.18973246069471034
```

The mean absolute error is 0.19 for OCUL.

Plot Diagnostics

For an ideal model the residuals should be uncorrelated white Gaussian noise centered on zero.



The **histogram** shows us the measured distribution; the orange line shows a smoothed version of this histogram; and the green line, shows a normal distribution. If our model is good these two lines should be almost the same.

The **normal Q-Q** plot is another way to show how the distribution of the model residuals compares to a normal distribution. If our residuals are normally distributed then all the points should lie along the red line, except perhaps some values at either end.

The last plot is the **correlogram**, which is just an ACF plot of the residuals rather than the data. 95% of the correlations for lag greater than zero should not be significant. If there is significant correlation in the residuals, it means that there is information in the data that our model hasn't captured.

Prob(Q) is 0.24, is the p-value associated with the null hypothesis that the residuals have no correlation structure

Prob(JB) is 0, is the p-value associated with the null hypothesis that the residuals are Gaussian normally distributed.

If either p-value is less than 0.05 we reject that hypothesis.

Prob(Q) indicates Residuals are not correlated and are normally distributed

Metrics

We now take our models for stock price predictions and leverage it to gain insights into OCUL, SLSB and URGN price predictions for next day, next week, next month. Note that this relies on the implicit assumption that all other market factors are priced in, in the actual stock market price. Essentially this assumes prices are set by a free market.

We can now use our model to gain insight into what stock price daily, weekly, monthly evolution could/should be, and how that might change under various scenarios.

Fit ARIMA Model On OCUL Daily Data

Scenario 1: Make daily predictions for last 20 values

```
The mean absolute error for actual daily OCUL values are:  7.511538432153847
The mean absolute error for predicted daily OCUL values are:  7.389809359016743
```

OCUL daily stock price autoregressive predictions are $0.18 away from the true values, on average

Scenario 2: Make dynamic daily predictions for last 30 values

```
The mean absolute error for actual next OCUL values are:  7.511538432153847
The mean absolute error for predicted future OCUL values are:  6.096883769294792
```

OCUL daily stock price dynamic predictions are $1.42 away from the true values, on average

Fit ARIMA Model On OCUL Weekly Data

```
The mean absolute error for actual weekly OCUL values are:  6.911499976999998
The mean absolute error for predicted weekly OCUL values are:  6.776868423526291
```

How far are OCUL weekly stock price predictions from the true values, on average: $0.50

```
1 mae = np.mean(np.abs(residuals_OCUL))
2 mae
```

```
0.5017219449432536
```

Fit ARIMA Model On OCUL Monthly Data

```
The mean absolute error for monthly actual OCUL values are:  5.310000026250001
The mean absolute error for monthly predicted last OCUL values are:  5.11845338014836
```

How far are OCUL monthly stock price predictions from the true values, on average: $1.28

```
1 mae = np.mean(np.abs(residuals_monthly_OCUL))
2 mae
```

```
1.2856351975782871
```

More Models:

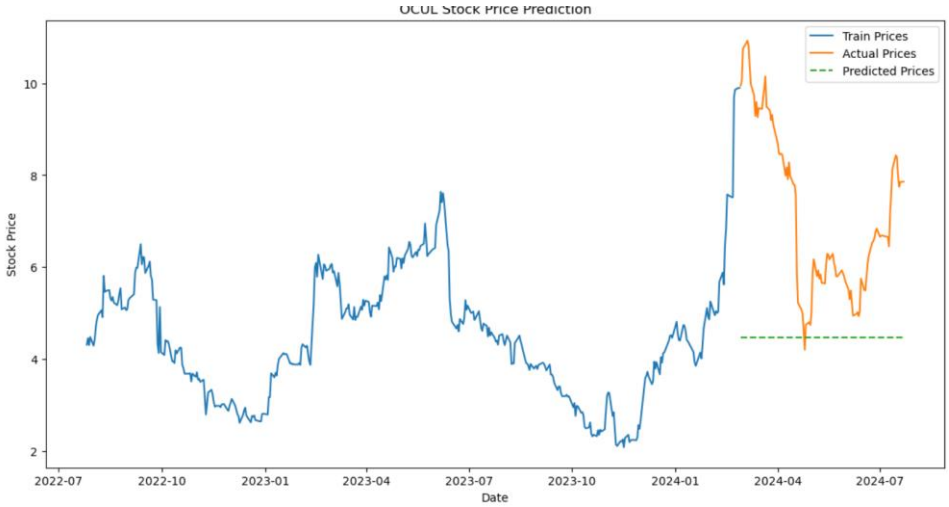
- **Dummy Regressor – strategy = 'mean',**
- **Random Forest and**
- **Decision Tree**

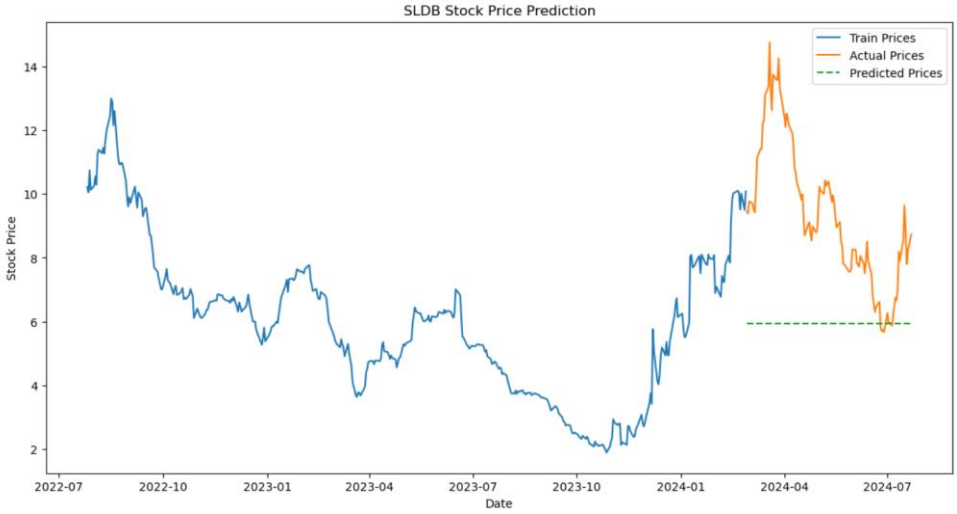
Dummy regressor:

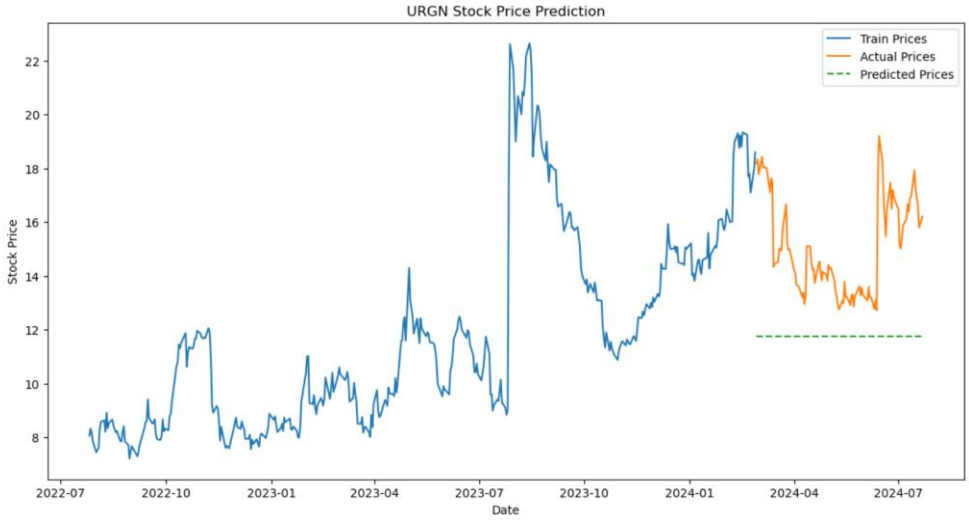
The DummyRegressor from scikit-learn is a regression model that uses simple rules to make predictions. This model is particularly useful for providing a baseline against which to measure the performance of actual regression models.

```
Evaluation results for OCUL:
MAE = 2.710534214391601, RMSE = 3.2431841595016793
Evaluation results for SLDB:
MAE = 3.4618874827669996, RMSE = 4.078958714150425
Evaluation results for URGN:
MAE = 3.285825030912499, RMSE = 3.736536646951657
```

Plot predictions using Mean strategy dummy regressor, with an 0.80 train/test split





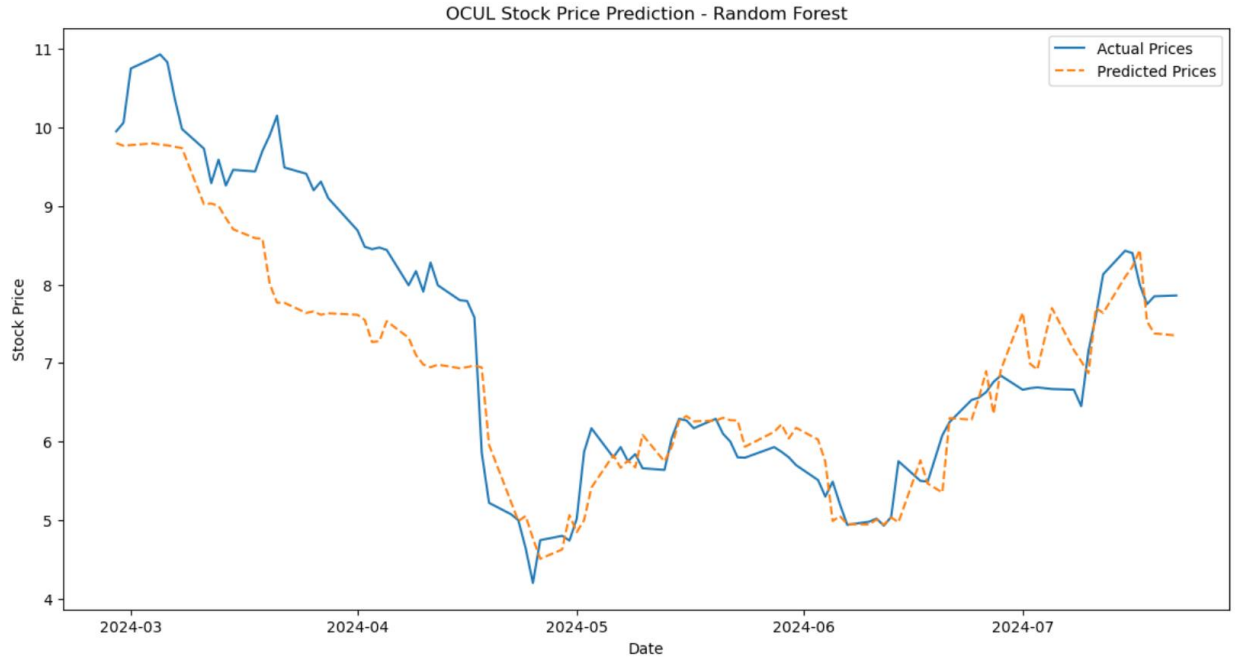


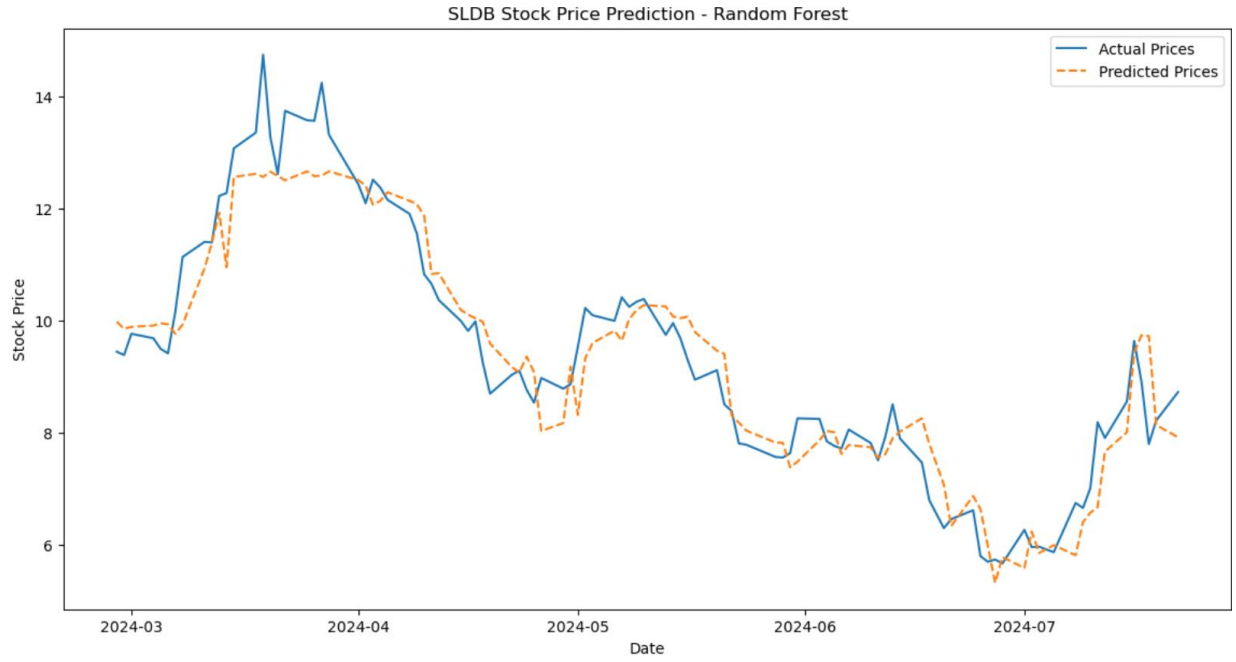
Machine Learning Models: Decision trees and Random Forest

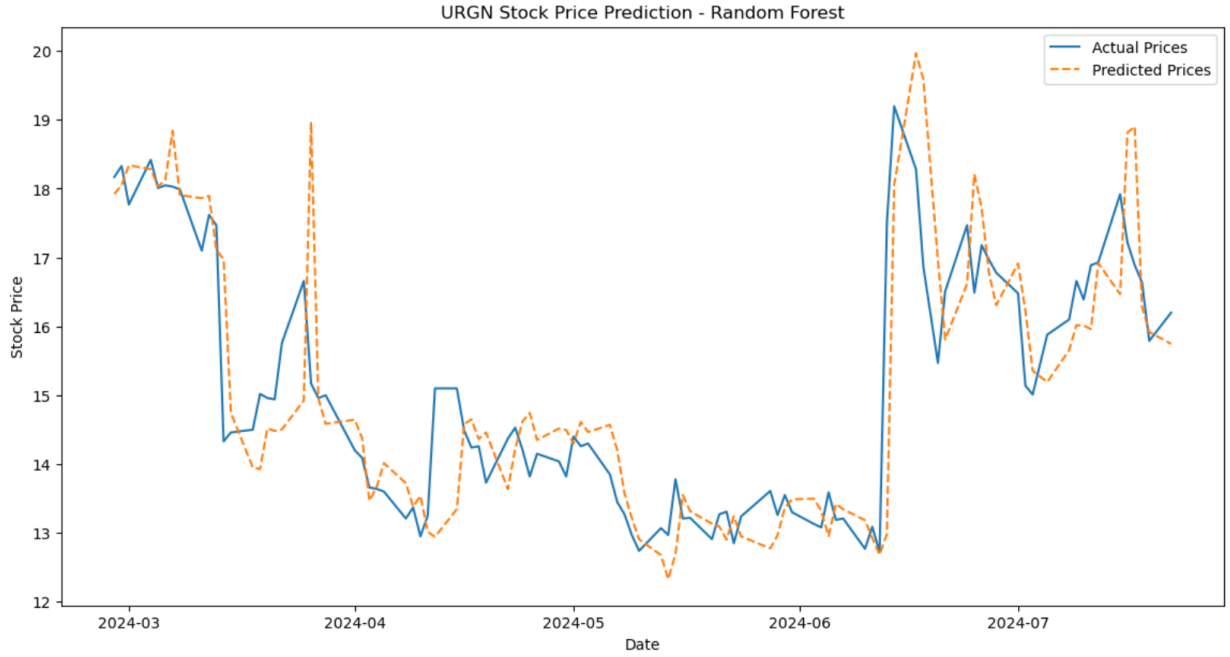
```
Evaluation results for OCUL:
Decision Tree: MAE = 0.8647000407499997, RMSE = 1.1875352664402687
Random Forest: MAE = 0.5567745286040973, RMSE = 0.7439513192702546
Evaluation results for SLDB:
Decision Tree: MAE = 0.6560499993200001, RMSE = 0.8287953677408658
Random Forest: MAE = 0.5063575059780991, RMSE = 0.6573426125752424
Evaluation results for URGN:
Decision Tree: MAE = 0.8389999687, RMSE = 1.2386540663576158
Random Forest: MAE = 0.6557249682400004, RMSE = 0.9956472959620062
```

The random Forest model is best machine learning model based on MAE metric

Plot predictions Random Forest model







Models for OCUL /Metric	ARIMA daily	ARIMA weekly	ARIMA monthly	Dummy R	Random Forest	Decision Tree
MAE	0.18	0.50	1.28	2.71	0.55	0.86
The best model corresponds to lowest MAE, which is ARIMA, on a daily frequency, MAE = 0.18						