SPRINGBOARD - DSC

CAPSTONE PROJECT 3

PREDICTING BITCOIN PRICE

By IGOR VEREVKIN

August 2021

1. INTRODUCTION:

Bitcoin(BTC) is the world’s first and largest cryptocurrency. On August 1, 2021 the price of Bitcoin was almost 40 thousand dollars and its total market capitalization is estimated to be just over 748 billion dollars.

The goal of this project is to study time series forecasting algorithms that can be used to predict BTC price the next day. Then we would like to extend the prediction for the next 30 days.

Intended stakeholders are all the people interested in Bitcoin market.

During this project we were able to build several time series forecasting models. The lowest Mean Absolute Percentage Error (MAPE) error with the test set we could achieve was 3% using ARIMA(2,1,1) model. All other model showed much worse results.

All the implementation details can be found in the notebooks we developed.

All the models presented in this report are built using the same train and test splits.

Link to the notebooks:
https://github.com/igorverevkin1992/Predicting-Bitcoin-Price

2. APPROACH

2.1. Data Acquisition And Wrangling

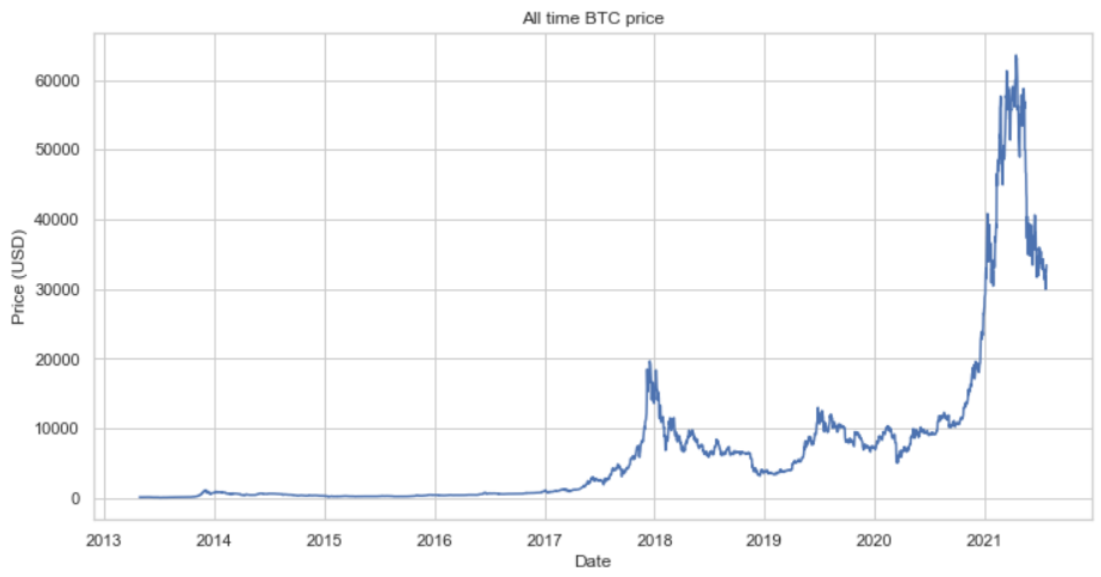
The dataset we used for the project was scraped from CoinGecko cryptocurrency exchange and includes date, price total market capitalization and total volume for BTC from April 2013 to July 2021.
The dataset was downloaded from:
https://www.coingecko.com/en

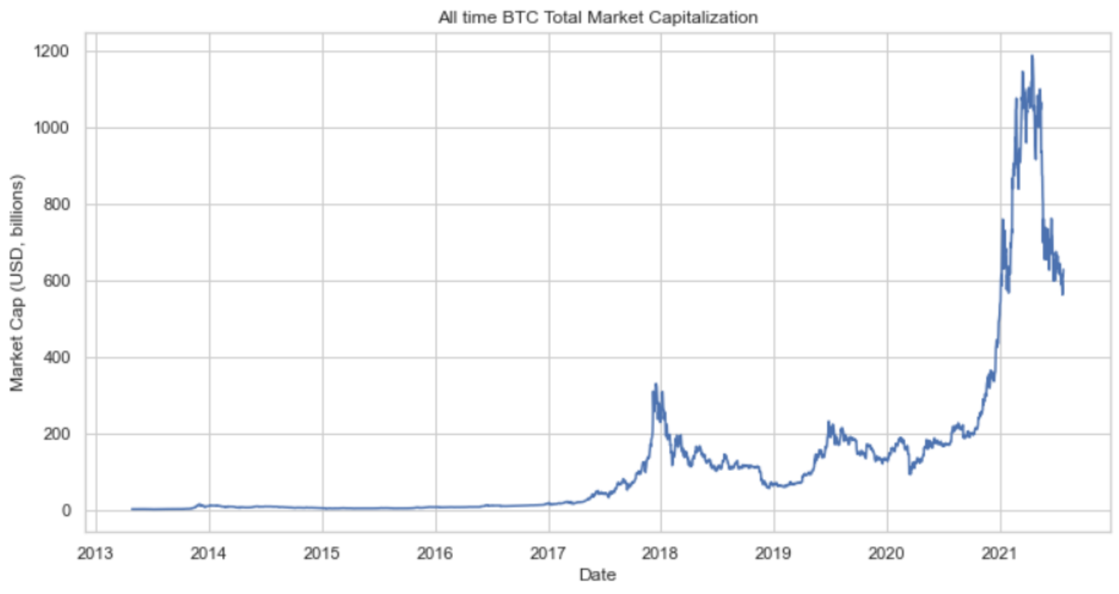
During Data Wrangling step of the Data Science Method, we checked if the data had any missing values or duplicates.

There were almost no missing values in the data as well as no duplicates.

2.2. Storytelling and Inferential Statistics

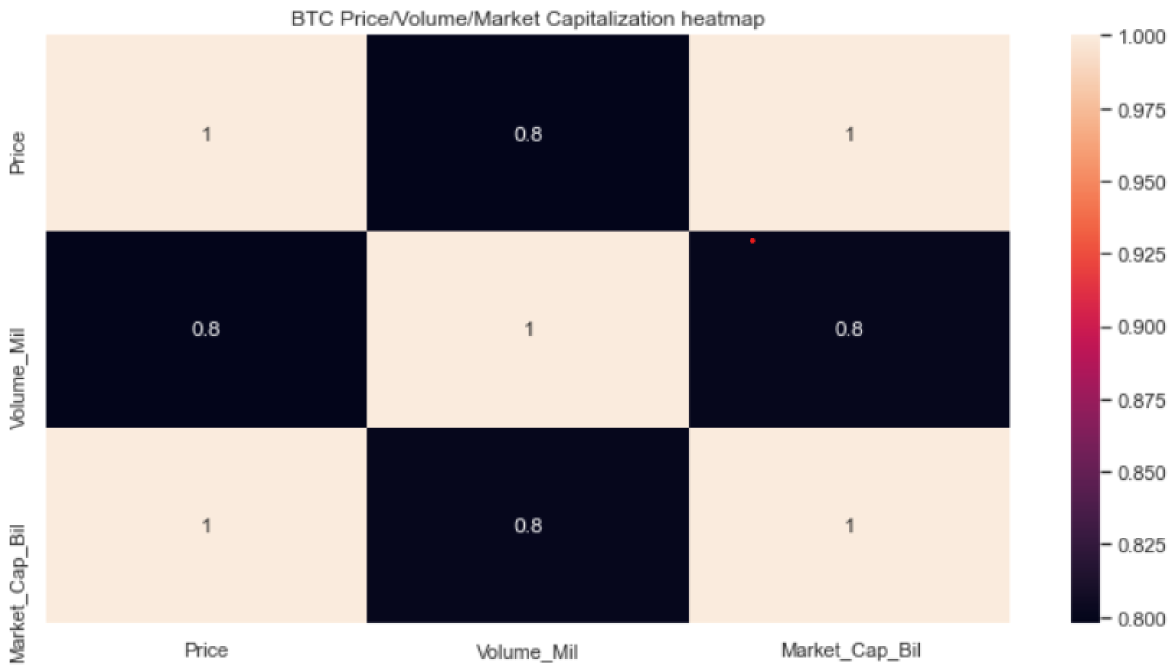
First, we needed to see a big picture of our data. Let’s take a look at BTC price and total market cap for April 2013 - July 2021





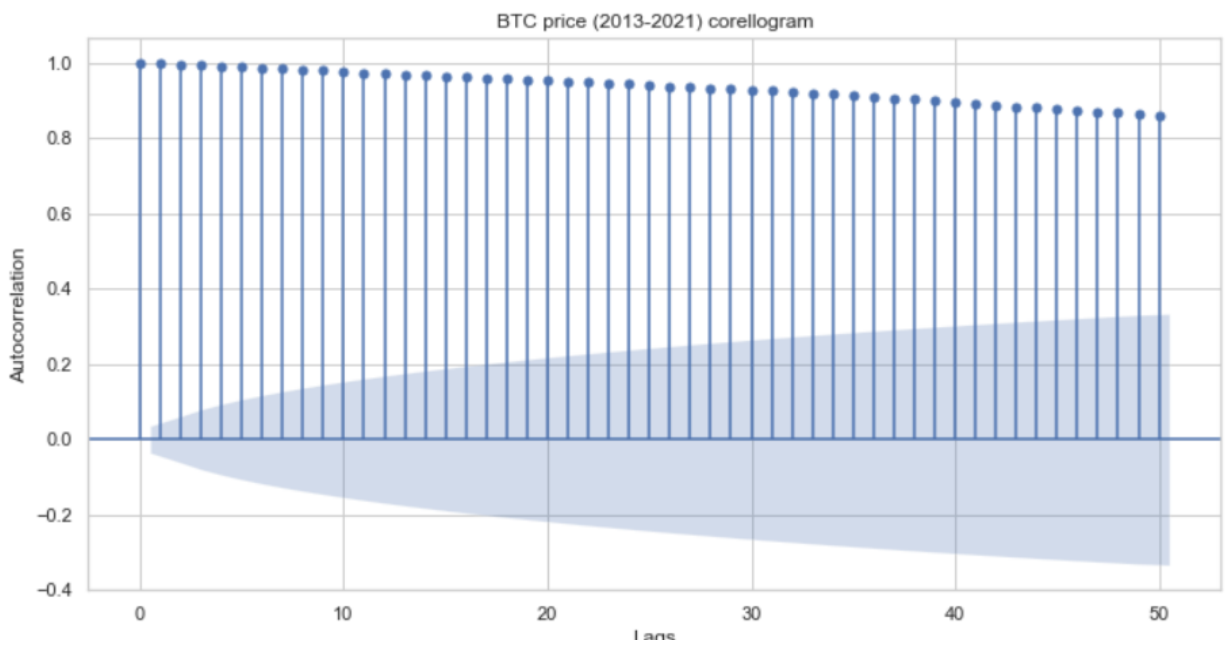
We can clearly see almost perfect positive correlation between Bitcoin’s price and market cap. It makes sense since the higher the price is, the higher the total cap is.

Let’s see if the correlation is indeed very strong. Here is a correlation heatmap for BTC.



We can see that the correlation between Price and Market cap is indeed 1, and both Price and Market Cap are also strongly positively correlated with Volume.

The next important thing was to find out if the BTC price was autocorrelated. For that we created a correlogram, that clearly showed strong autocorrelation.



And we checked the quantitative value of BTC price autocorrelation too.

```
In [37]: print(BTC_price.autocorr())
         0.9986630809622649
```

As we can see both from the correlogram and pandas autocorr() function, BTC price is indeed very strongly autocorrelated.

Next step was to perform **Hypothesis Testing**. We formulated two hypotheses:

- *Null Hypothesis*: Non-Stationarity exists in BTC Price
- *Alternative hypothesis*: Stationarity exists in BTC Price

In order to find out if our Null hypothesis was correct or not we performed Augmented Dickey-Fuller Test.

```
res_norm = adfuller(BTC_price)
print('p-value is:', res_norm[1])
```

```
p-value is: 0.7874864598299681
```

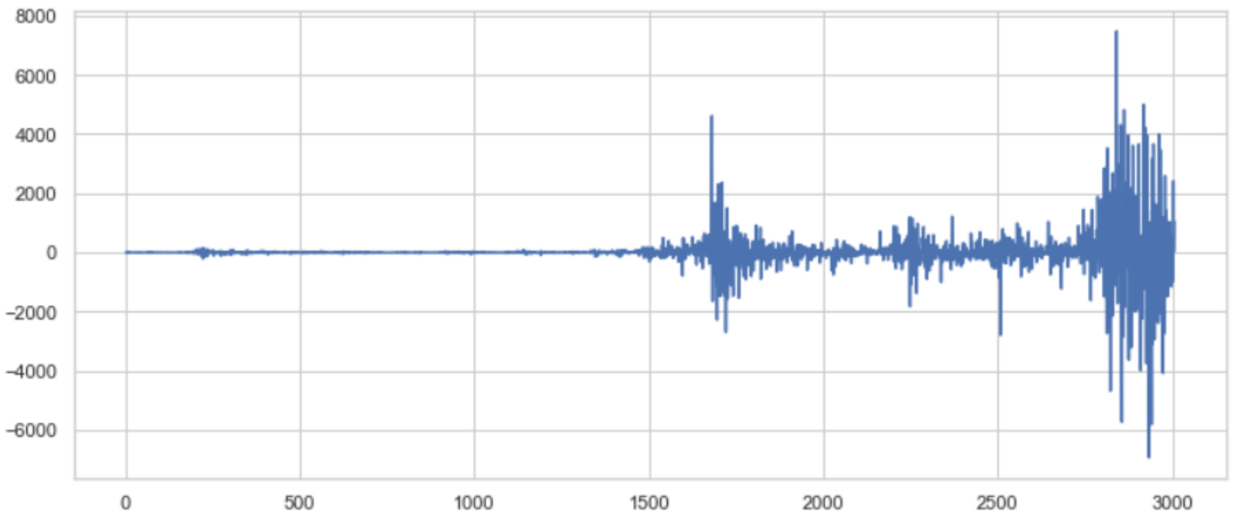
With such high p-value we **failed to reject** the null hypothesis. BTC price thus was not stationary, and we had to take that into account in our next step – modeling.

2.3. Baseline Modeling

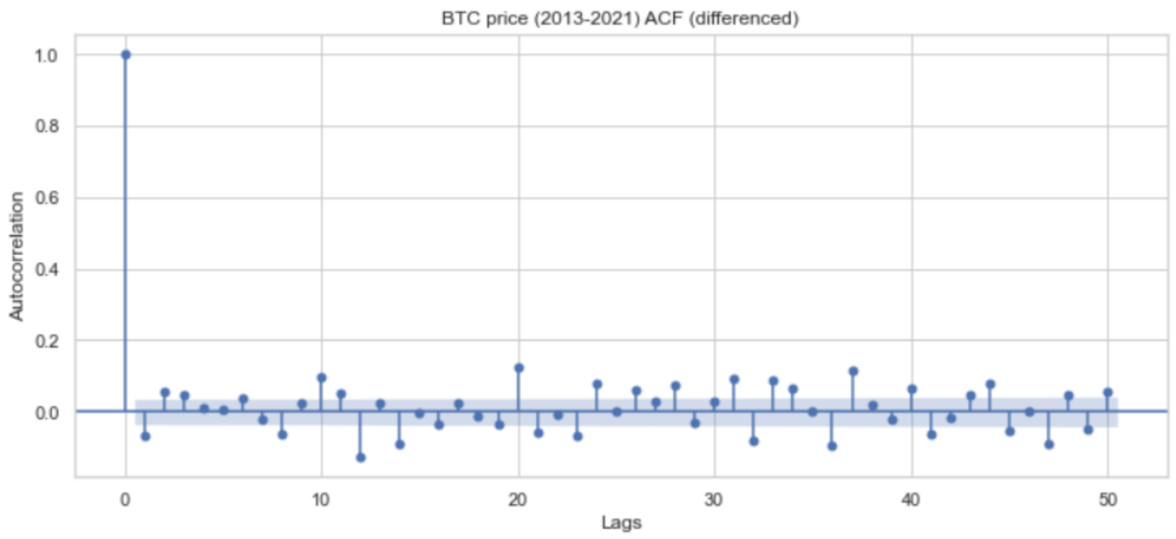
IMPORTANT NOTE:

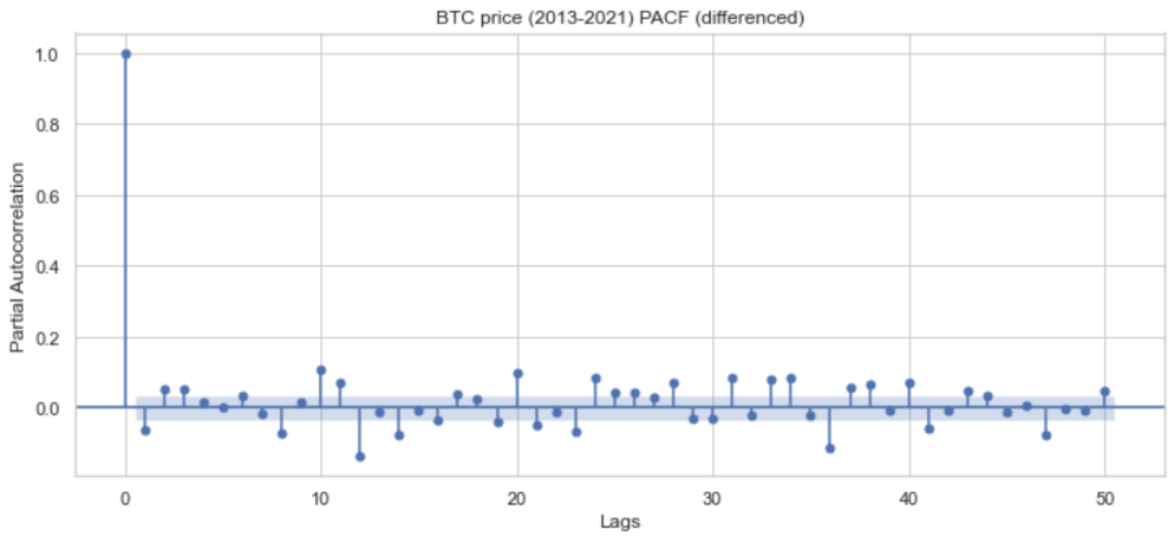
Since we're modeling time series, we will the most popular metrics used for forecasting error - Mean Absolute Percentage Error(MAPE) - as our main metric to assess the model's performance.

Since our problem is a time series analysis problem, we picked **ARIMA**[1] as our baseline model. We scaled the data with Scikit-learn MinMaxScaler and split it into train and test sets. Notice that train/test split preserved the time order. Then we differenced the time series (made it stationary).



After that, we manually plotted ACF and PACF so we could determine which order to use for our ARIMA model.



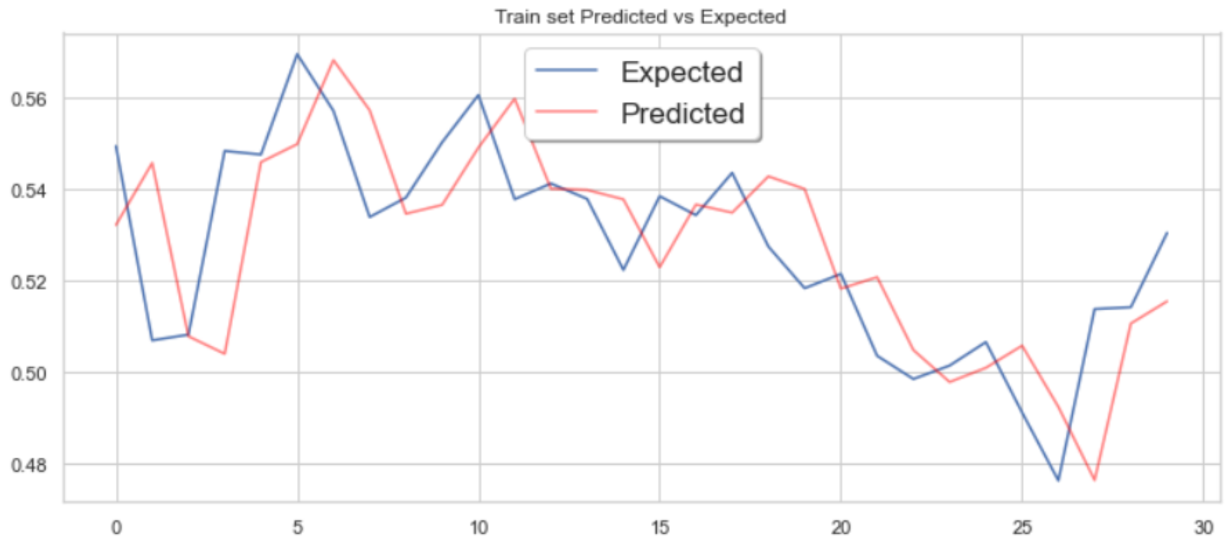


From the ACF we PACF and differencing we determined that the best order for our model would be (2,1,1). Then we trained the model on the test set and evaluated it using the test set.

```
r2 = r2_score(test_norm, predictions_test)
print(f'Test set r-squared for ARIMA(2,1,1) is: {round(r2, 2)}')
mape = mean_absolute_percentage_error(test_norm, predictions_test)
print(f'Test set MAPE for ARMIA(2,1,1) is: {round(mape,2) * 100}%')
```

```
Test set r-squared for ARIMA(2,1,1) is: 0.34
Test set MAPE for ARMIA(2,1,1) is: 3.0%
```

MAPE of 3% is a great result. Most retail companies allow up to 15-20% of MAPE.



From this Predicted vs Expected plot we can see that our baseline ARIMA(2,1,1) model was able to accurately catch the data structure.

2.4. Extended Modeling

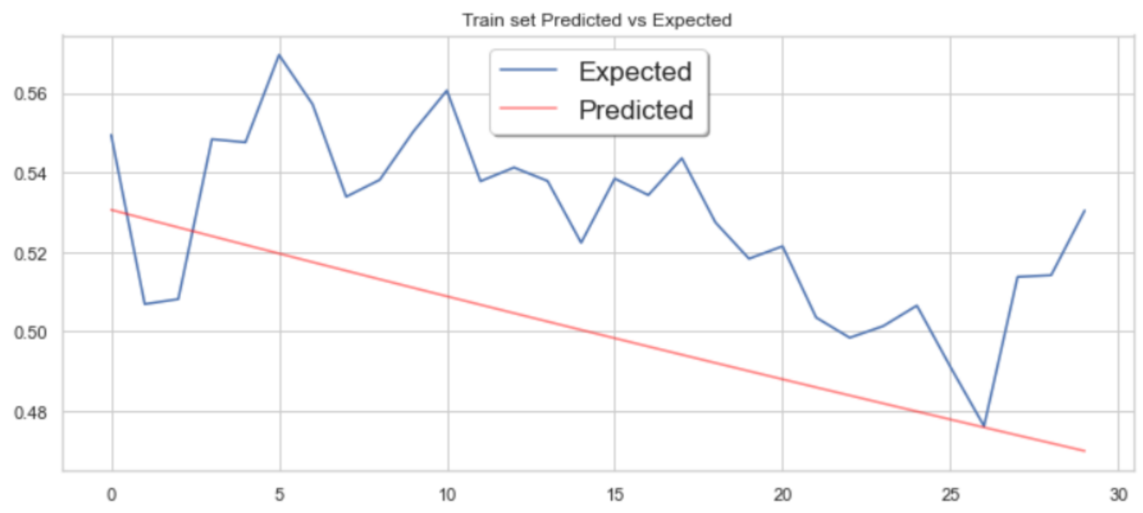
Even though our Logistic Regression model showed good results, we would like to see how other time-series forecasting algorithms would perform. We built three models (Exponential Smoothing[1], Facebook Prophet[2] and PyCaret[3]). The process was the same for all of them – we built out-of-box model, evaluated it, tuned hyperparameters, evaluated tuned model and printed the results for each model.

Below you can see the results.

Exponential Smoothing:

```
r2 = r2_score(observations_test_mul, predictions_test_mul)
print(f'Test set r-squared for Double ES (multiplicative) is: {round(r2, 2)}')
mape = mean_absolute_percentage_error(observations_test_mul, predictions_test_mul)
print(f'Test set MAPE for Double ES (multiplicative) is: {round(mape, 2) * 100}%')
```

```
Test set r-squared for Double ES (multiplicative) is: -1.31
Test set MAPE for Double ES (multiplicative) is: 6.0%
```



The results of the out-of-box exponential smoothing model are bad. Even though MAPE is 6%, we can see that the model did not catch the data's structure.

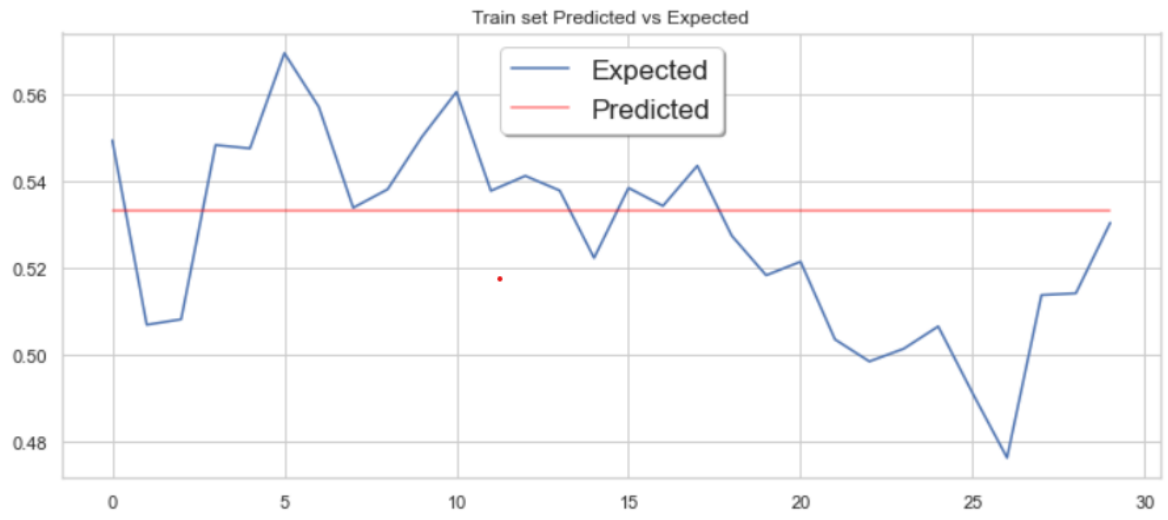
Now let's see the results of the tuned model. We used "brute" argument of the function which performs grid search of the optimal starting values.

[1] https://www.statsmodels.org/stable/examples/notebooks/generated/exponential_smoothing.html
[2] https://facebook.github.io/prophet/
[3] https://pycaret.org/

```
r2 = r2_score(observations_test_mul, predictions_test_mul)
print(f'Test set r-squared for Double ES (gridsearch) is: {round(r2, 2)}')
mape = mean_absolute_percentage_error(observations_test_mul, predictions_test_mul)
print(f'Test set MAPE for Double ES (gridsearch) is: {round(mape, 2) * 100}%')
```

```
Test set r-squared for Double ES (gridsearch) is: -0.07
Test set MAPE for Double ES (gridsearch) is: 4.0%
```

MAPE is still small, however we have r-squared less than 0, meaning the model again was not able to catch the structure. We can see it on the plot.



Overall, Exponential Smoothing showed poor results and should not be recommended for production use.

Facebook Prophet:

Prophet is a time-series forecasting algorithm developed by Facebook and is a part of Kats - a framework for time-series analysis, which includes several models. It is important to note that, according to its developers, the main idea behind Prophet is to make generic time series analysis easier - generic meaning time series that are most common in business, which have any of the following characteristics:

- hourly, daily, or weekly observations with at least a few months (preferably a year) of history

- strong multiple "human-scale" seasonality: day of week and time of year

- important holidays that occur at irregular intervals that are known in advance

- a reasonable number of missing observations or large outliers

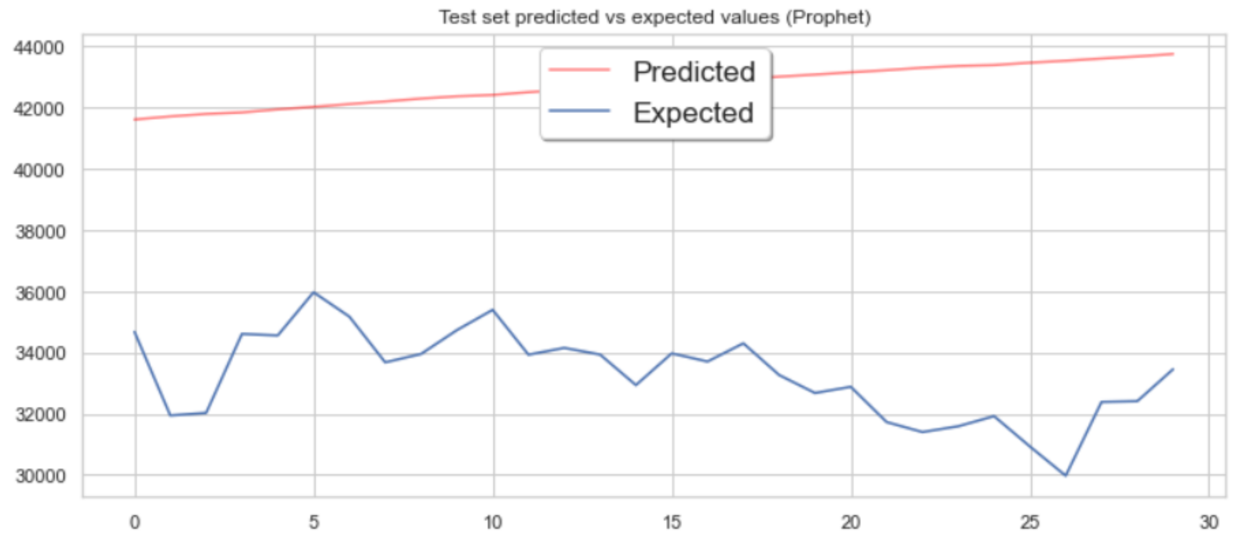
- historical trend changes, for instance due to product launches or logging changes

- trends that are non-linear growth curves, where a trend hits a natural limit or saturates

Since our BTC price does not particularly meet those standards, we did not anticipate that Prophet would show good results.

```
r2 = r2_score(test_pr['y'], btc_prophet_pred['yhat'])
print(f'Test set r-squared for Prophet is: {round(r2, 2)}')
mape = mean_absolute_percentage_error(test_pr['y'], btc_prophet_pred['yhat'])
print(f'Test set MAPE for Prophet is: {round(mape,1) * 100}%')
```

```
Test set r-squared for Prophet is: -45.99
Test set MAPE for Prophet is: 30.0%
```



Indeed, the results are quite poor. The prediction is way off and MAPE is very high.

Let's look at the tuned Prophet performance. Here are the parameters we used for grid search.

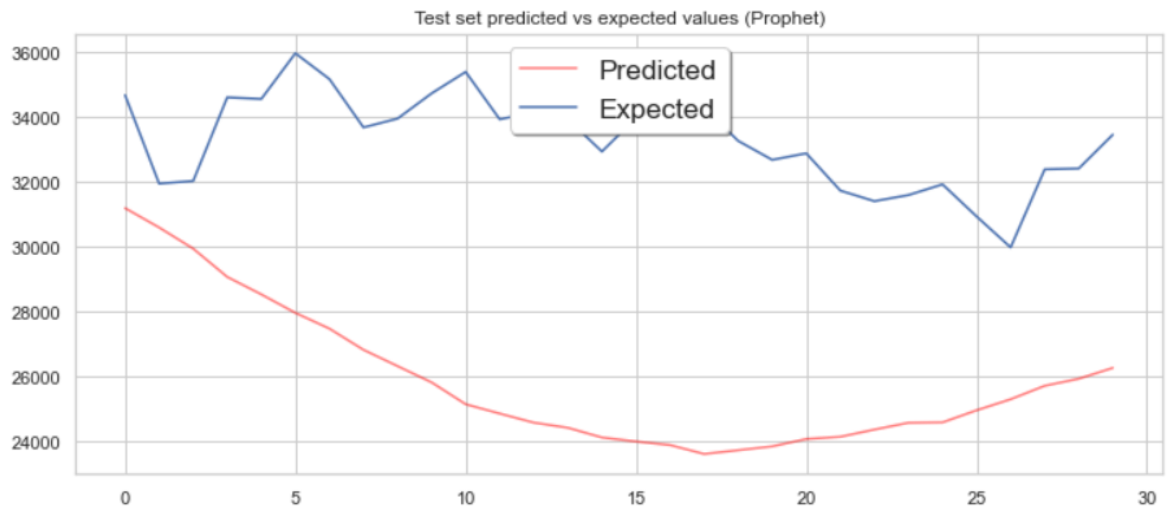
```
params_grid = {'seasonality_mode':('multiplicative','additive'),
               'changepoint_prior_scale':[0.1,0.2,0.3,0.4,0.5],
              'holidays_prior_scale':[0.1,0.2,0.3,0.4,0.5],
              'n_changepoints' : [3,5,10,100]}
grid = ParameterGrid(params_grid)
cnt = 0
for p in grid:
    cnt = cnt+1

print('Total Possible Models',cnt)
```

And the results.

```
r2 = r2_score(test_pr['y'], btc_prophet_pred['yhat'])
print(f'Test set r-squared for Prophet is: {round(r2, 2)}')
mape = mean_absolute_percentage_error(test_pr['y'], btc_prophet_pred['yhat'])
print(f'Test set MAPE for Prophet is: {round(mape,2) * 100}%')
```

```
Test set r-squared for Prophet is: -29.5
Test set MAPE for Prophet is: 22.0%
```



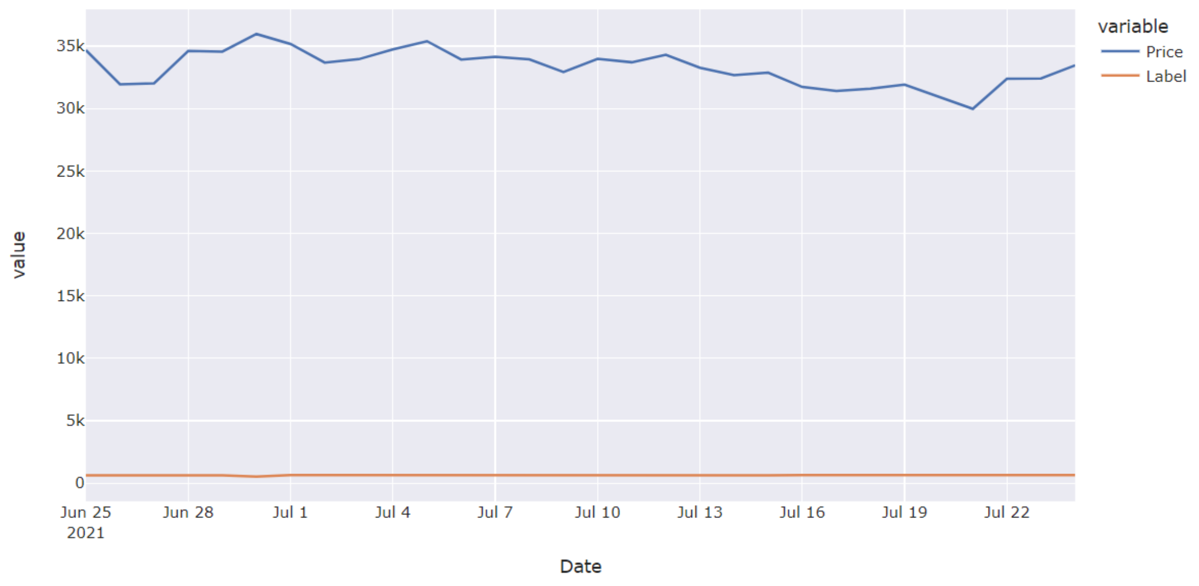
As we expected, Prophet was unable to catch the data's structure and showed poor results. It should not be recommended for production use.

Pycaret:

PyCaret is a Python open source machine learning library. It is a Python version of popular R package "Caret". The main idea of this package is to allow easy modeling, evaluation and tuning of classification and regression algorithms. Time series analysis, being a case of regression, can also be performed with PyCaret.

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Passive Aggressive Regressor	15933.7515	501479443.4791	22393.7367	-1.0281	3.0910	0.9345

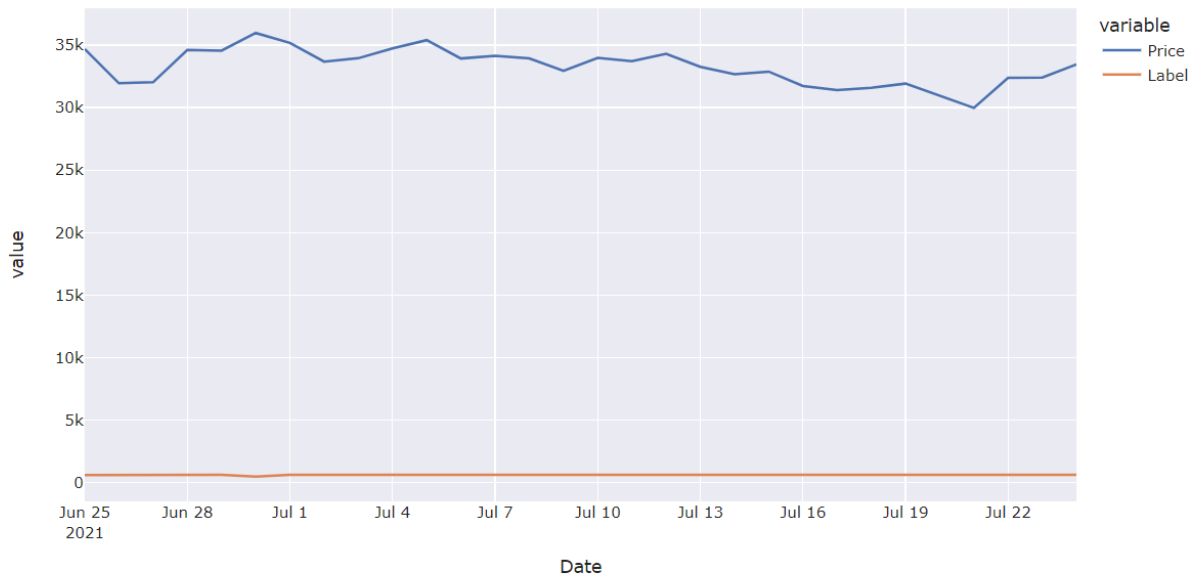
NOTE: MAPE is shown here in decimal. In order to get the percentage, we need to multiply it by 100. Thus, MAPE here is 93.5%.



Indeed, we see that prediction is absolutely off. Out-of-box PyCaret model didn't work pretty much at all. Now let's take a look at the tuned model. Here you can see the set of optimal hyperparameters.

```
PassiveAggressiveRegressor(C=1.0, average=False, early_stopping=False,
                           epsilon=0.1, fit_intercept=True,
                           loss='epsilon_insensitive', max_iter=1000,
                           n_iter_no_change=5, random_state=42, shuffle=True,
                           tol=0.001, validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

And the results of the tuned model.



No improvement whatsoever. MAPE is the same – 93%

3. FINDINGS

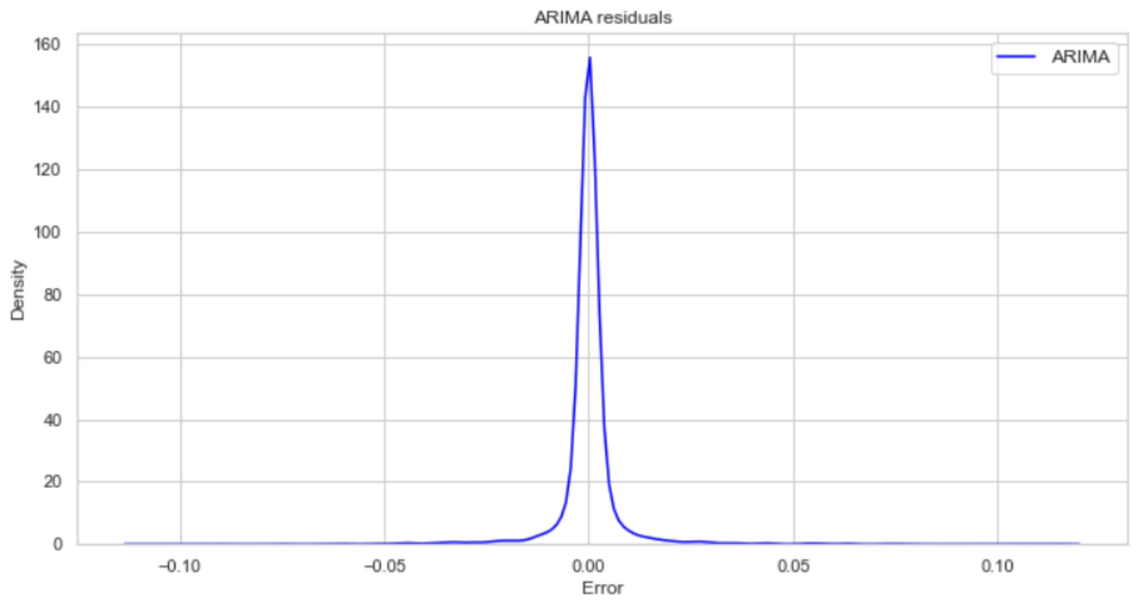
After we finish modeling, we can perform comparative analysis of each model's performance with regard to the others. This will let us identify the best model, which we can then implement for solving our original problem – predicting Bitcoin price.

Here you can see a table with all the models' performance metrics.

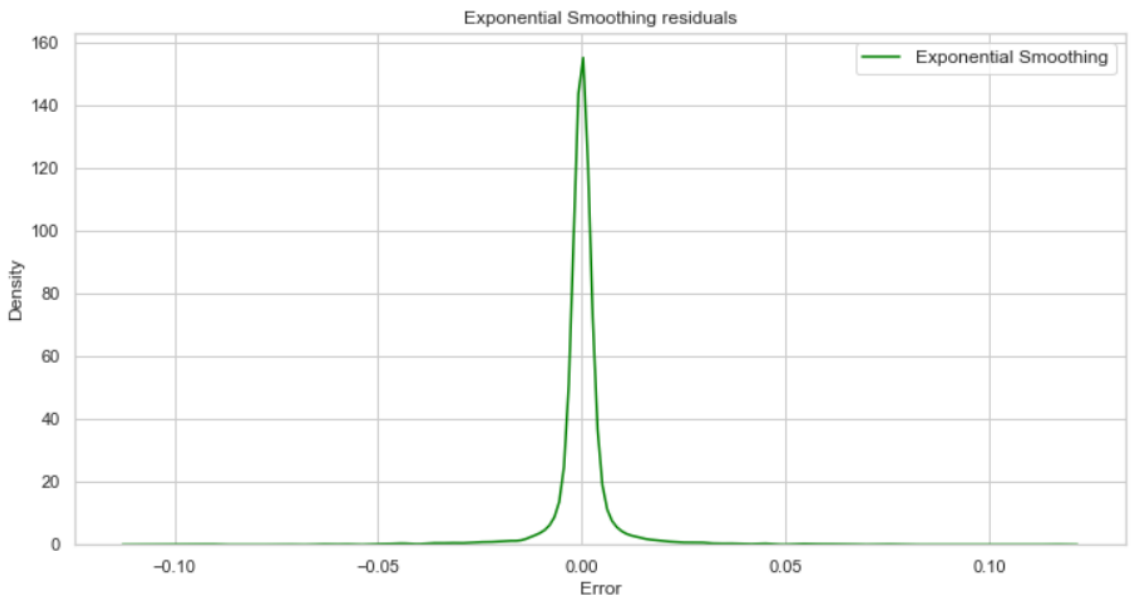
MODEL	OBSERVATIONS	r-squared	MAPE
ARIMA(2,1,1)	30	0.37	3%
Exponential Smoothing (additive)	30	-0.23	4%
Exponential Smoothing (multiplicative)	30	-1.31	6%
Exponential Smoothing (grid search)	30	-0.07	4%
Facebook Prophet (no grid search)	30	-45.99	30%
Facebook Prophet (grid search)	30	-29.5	22%
PyCaret (no grid search)	30	-44	93%
PyCaret (grid search)	30	-43	93%

Now let's take a look at each model's residuals distribution.

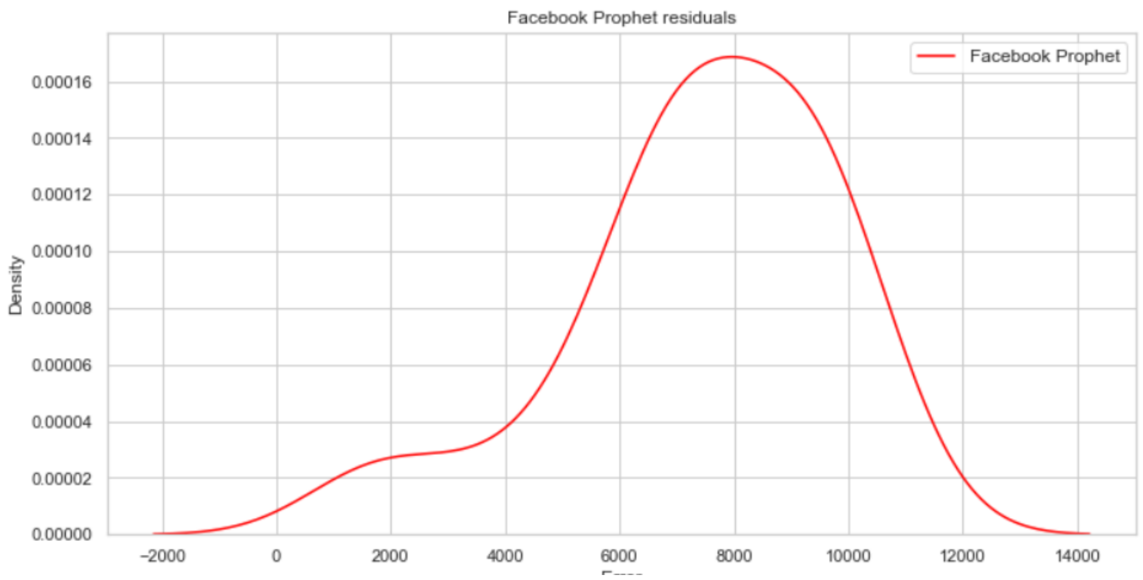
ARIMA (2,1,1)



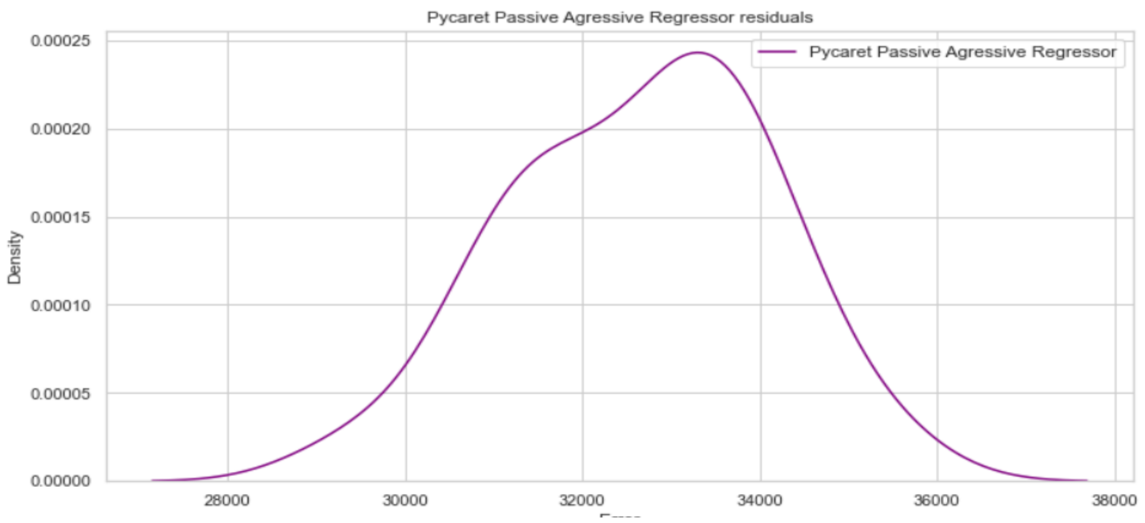
EXPONENTIAL SMOOTHING (MULTIPLICATIVE)



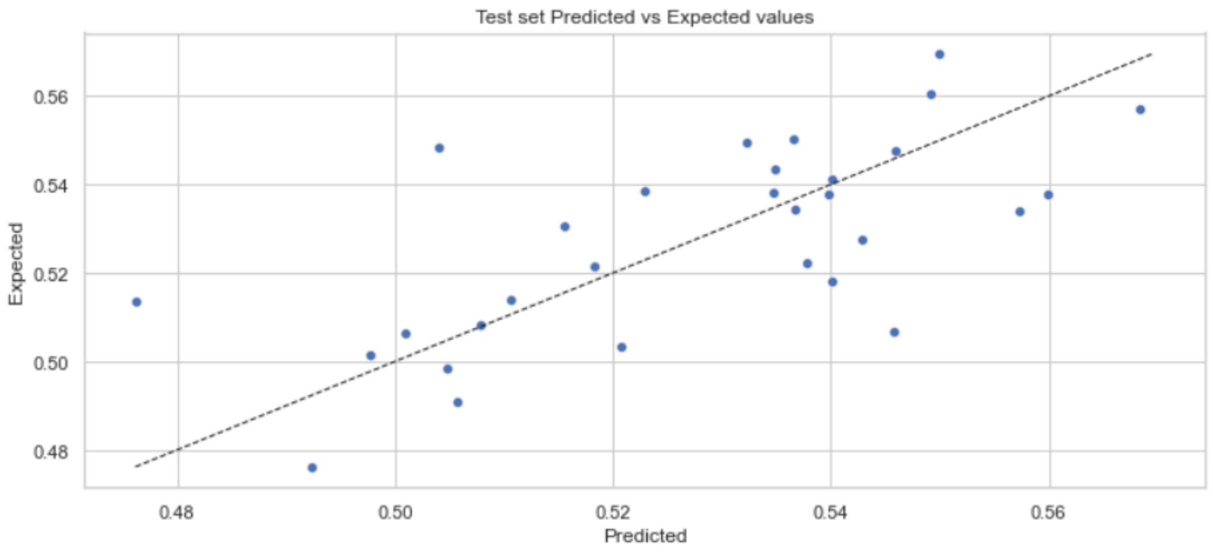
FACEBOOK PROPHET



PYCARET PASSIVE AGGRESSIVE REGRESSOR



Only ARIMA and Exponential Smoothing models showed MAPE which is acceptable. You can also see that their residuals distributions are really narrow and close to 0. However, we know that only ARIMA (2,1,1) was able to actually capture the data's structure. ARIMA, however, did it really well. Here is the residuals plot for ARIMA (2,1,1).



We can see, even there are some outliers, the model performed well and showed good results.

4. Conclusion and Future Work

Now we should answer the question – which model should we recommend to implement?

From the analysis above it's quite clear that among all four models we built in this project only one – namely ARIMA (2,1,1) – showed low error, and was able to accurately capture the time series structure and overall performed well.

For future work, in order to better predict Bitcoin price it would be useful to try neural networks as well as the sentiment analysis of the Bitcoin-related twitter accounts.

5. Recommendations for the Clients

After performing all the steps described above, we can recommend ARIMA (2,1,1) for productional implementation.