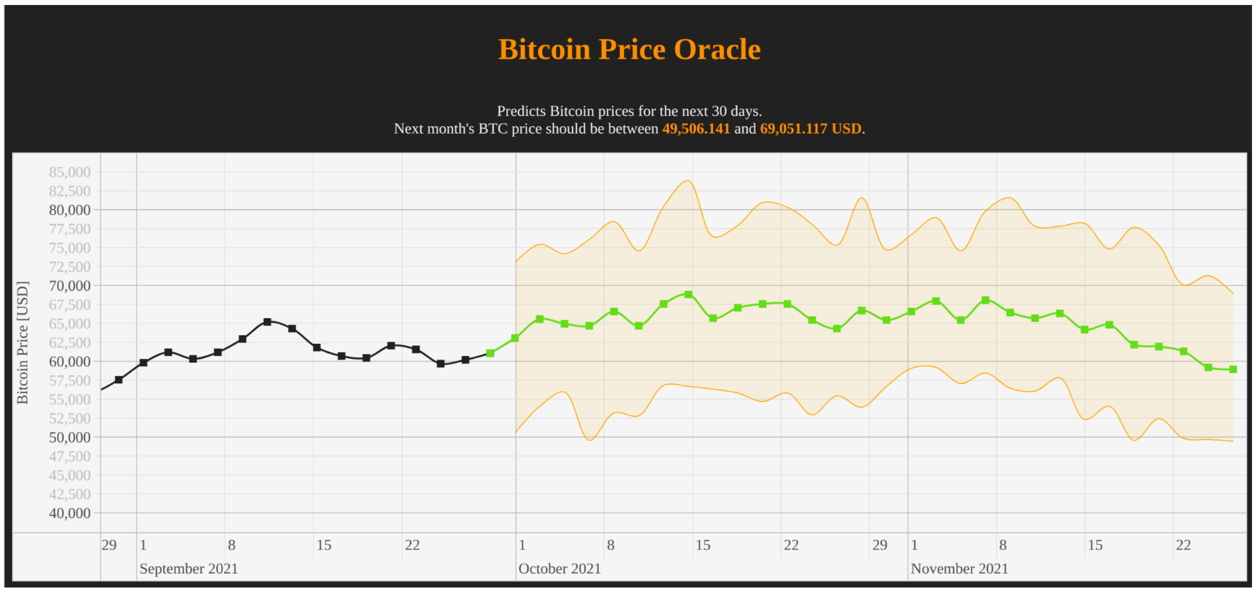


Bitcoin Price Forecasting Project Report

30.10.2021

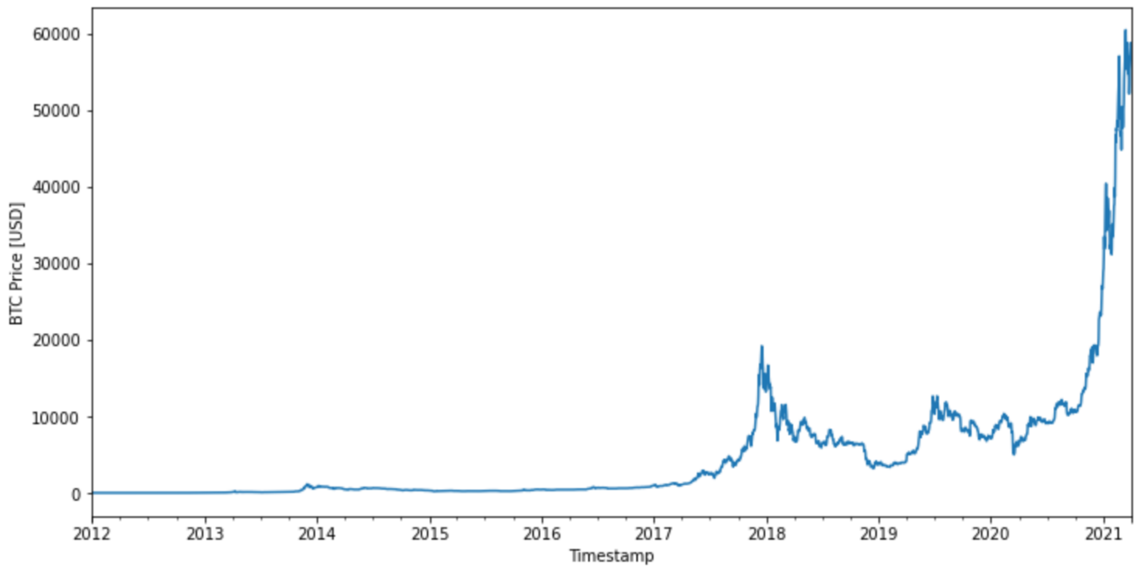
Florian Mitterbauer

1. Data Exploration: Loading, Exploration, Feature Selection

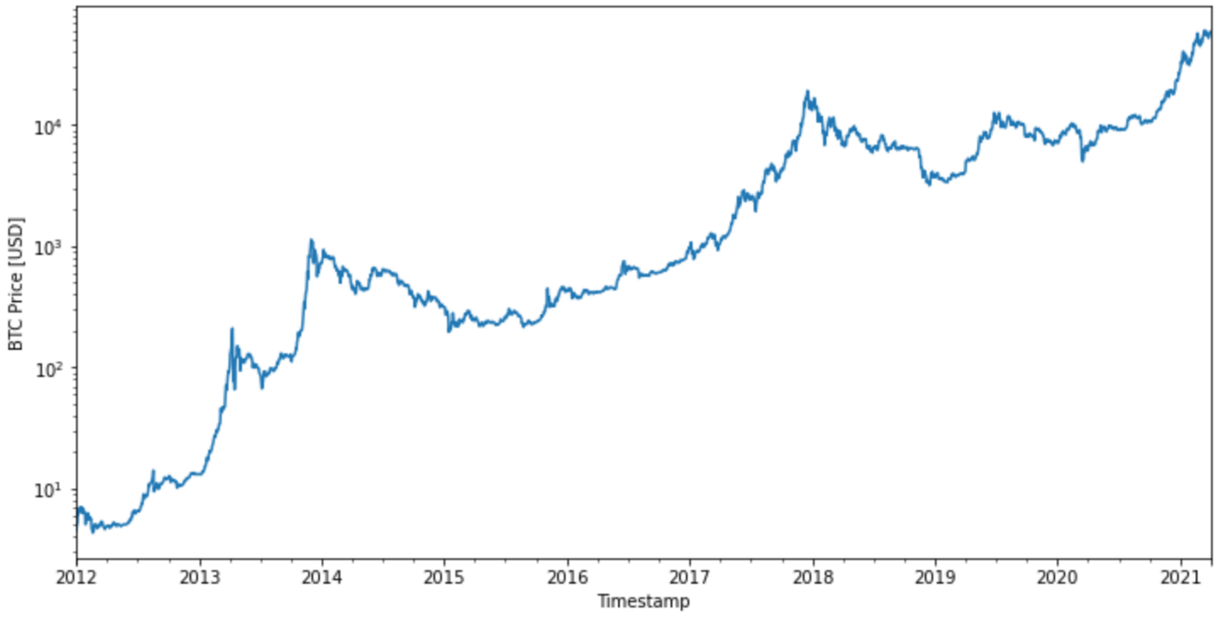
The Bitcoin price data source was in the form of a CSV with a rough size of 300 MB which was sourced from Kaggle. It originated from a very old crypto exchange named Bitstamp and contained price data from 2012 until 2021. The data was in a for financial assets typical format of Open, High, Low, Close and Volume (OHLCV). The prices were all in USD but the Volume column as well in BTC and additionally there was a Weighted Price column present.

For the, in my opinion, most important column, the Weighted Price, I extracted some statistical values and drew a graph over the whole time series. Once in linear scale and a second time with a log10 scale for the y axis, since the Bitcoin price was increasing so much during the whole time that a logarithmic makes sense.

Weighted Price Statistics [USD]	
Average	6,008.93
Median	3,596.80
Minimum	3.80
Maximum	61,716.21



Weighted Bitcoin price in USD on linear scale from 2012 to 2021



Weighted Bitcoin price in USD on logarithmic scale from 2012 to 2021

One of the requirements of a time series for AWS deepAR (probably in general as well) was that it must not contain any empty/NaN/null values. I checked and found that the dataset contained a couple of them. Since the data also needed to stay continuous, I decided to fill the empty values with ffill which forward propagates null values.

The cleaned data now had to be scaled down to a more digestible size. The source data has a resolution of one minute, meaning one price value per minute, which adds up to too many rows, considering the total time frame is over 9 years.

So I decided to resample the time series from one minute up to one day by averaging the minute prices to a daily price. That brought it down from 48,573,77 to 3,379 rows.

At first I also thought about using one hour instead of one day as a resolution. But I figured out that I couldn't achieve my expected prediction time frames with it. I wanted to predict in a time frame of 30 days which would be only 30 data points with a daily resolution, that would've been totally fine with the best practice from deepAR to have less than 400 data points for a good/accurate prediction. But with a resolution of one hour a 30 day time frame would have resulted in 720 data points which is way above the recommended maximum. So I chose to go with the resolution of 1 day to be able to nicely predict for a month ahead.

After scaling the dataset I had to decide which features/columns to keep and which to remove. Initially I thought to only go with the Weighted Price column only to realize later after researching a bit more about time series analysis, that it's better to have more time serieses. So I chose Weighted Price, High and Low also because I thought this way I could capture more the upper and the lower bound of the price movements, since especially with avering out many (minutely) into very few (daily) prices cuts out many very short lived high and low values from the dataset. I was hoping that by including High and Low columns, even though they were also averaged, I could get at least some of the precious information about very high and very low prices back for the deep learning algorithm to train on.

2. Time Series Creation

DeepAR has very specific requirements on how it expects the input time series data. Also considering how far in the future one wants to be able to predict.

The expected data format is a JSON that should (in our case) contain the start timestamp and an array of prices for each time series. It then automatically deducts the timestamps of each individual value when you specify the frequency, later on when defining the hyper parameters. That's basically the reason as to why the data can't have time gaps cause otherwise it would screw up the training.

So in my case this results in 3 time series all starting at the end of March 2012 with 3,379 prices each for High, Low and Weighted Price.

When creating a DeepAR trained model two specific values need to be predefined:

1. Prediction length
2. Context length

The prediction length defines how many data points the model will be able to predict in the future. In my case this should be 30 days (basically roughly a month). The prediction length is fixed for a given trained model, which means that the inference result will always have the length of the prediction length, e.g. it will always return 30 predicted prices.

The context length is the second important parameter to choose. It defines how long the algorithm should look in the past to infer future values from past data. The context length should be at least as long as the prediction length. Factoring in seasonality, there's no need to stretch the context length to a year or so, because even though the context length might be only two months, the algorithm will also sample data outside of the context period.

The chosen context length was 60 days (double the prediction length basically). This also influenced the creation of the test and the training dataset.

Usually when preparing data for machine learning you randomly sample and split it in different portions for training and test/validation. But in the case of deepAR, the training and test dataset is not randomized and can contain the same values. The split in this case happens by a certain time. I chose to go with the best practice to use the whole time series for the test dataset and for training also to use the whole time series minus the prediction length (exclusive the last 30 days of price data).

After preparing the test and training JSON I uploaded them to S3, ready for the model to train on.

3. Training, Hyperparameter Tuning

For training with DeepAR I created an estimator with among others following parameters:

- Image_uri: uri of forecasting-deepar
- Instance_type: ml.c4.xlarge
- Instance_count: 1

After that it was time to prepare the hyperparameters for the estimator. I kept these parameters fixed:

- time_freq: D
- context_length: 60
- prediction_length: 30

So the time frequency (mentioned earlier) was set to daily (D) and the context and prediction length to the previously discussed values.

The other parameters were tuned to the best fit I found. At first I kept the number of cells (of each hidden layer) and the number of (hidden) layers fixed at 40 cells and 2 layers and tried different epochs 100 - 600, mini batch sizes 32 - 128, drop out rates 0.1 - 0.15 and early stopping parameters 10 - 300.

During the first few trainings I saw that the epoch losses were not steadily decreasing but jumping a bit up and down and only going down slowly. So I decided to keep the dropout rate at an increased value of 0.15 (a value of 0.2 was considered as max value for DeepAR). For the mini batch size, the minimum value of 32 yielded the best results, so I kept it like that. Since the epoch losses were not really much increasing after 100 epochs and the early stopping was terminating the training usually between 100 and 200 epochs, so I kept the epoch count to 200 and settled for a early stopping of 40, which seems high but I wanted to make sure to really capture the best epoch loss possible. Next I tried to increase the learning rate, in the hope that it might help with the ups and downs of the loss values between epochs, but increasing it actually had worse results so I kept it at its default of 0.001.

When comparing the training results of different hyper parameter configurations I always focused on the lowest possible value of root-mean-square error (RMSE) and the quantile losses, displayed at the end of each training.

Another parameter that could be changed was the likelihood (noise model) which to be used for estimating uncertainties. Of the available ones only two applied to the Bitcoin price prediction use case:

- gaussian: usually used for real world data
- student-T: also for real world data but good with bursty data (default)

At first I tried with the gaussian likelihood cause I considered the source data not very bursty/choppy. It yielded good results but I still wanted to try the student-T since it was the default likelihood method for DeepAR. I got better results with it, so my source data probably is either bursty or the student-T is just better for it for another reason.

After researching a bit more about DeepAR I found that you could increase the number of hidden layers beyond 4, so I tried to tune the layers and the cells. The best results were achieved with 6 hidden layers and 100 cells. It didn’t increase the training time that much but the results were by magnitudes better. I got three times lower RMSE and a quantile loss at around 0.1 for most quantiles which was also three times lower than with the trainings before.

At the end I decided on those hyper parameters:

num_cells / Cells per hidden layer	100
layers / Hidden layers	6
likelihood	student-T
epochs	200
mini_batch_size	32
learning_rate	0.01
dropout_rate	0.15
early_stopping_patience	40

They resulted in a model with following losses:

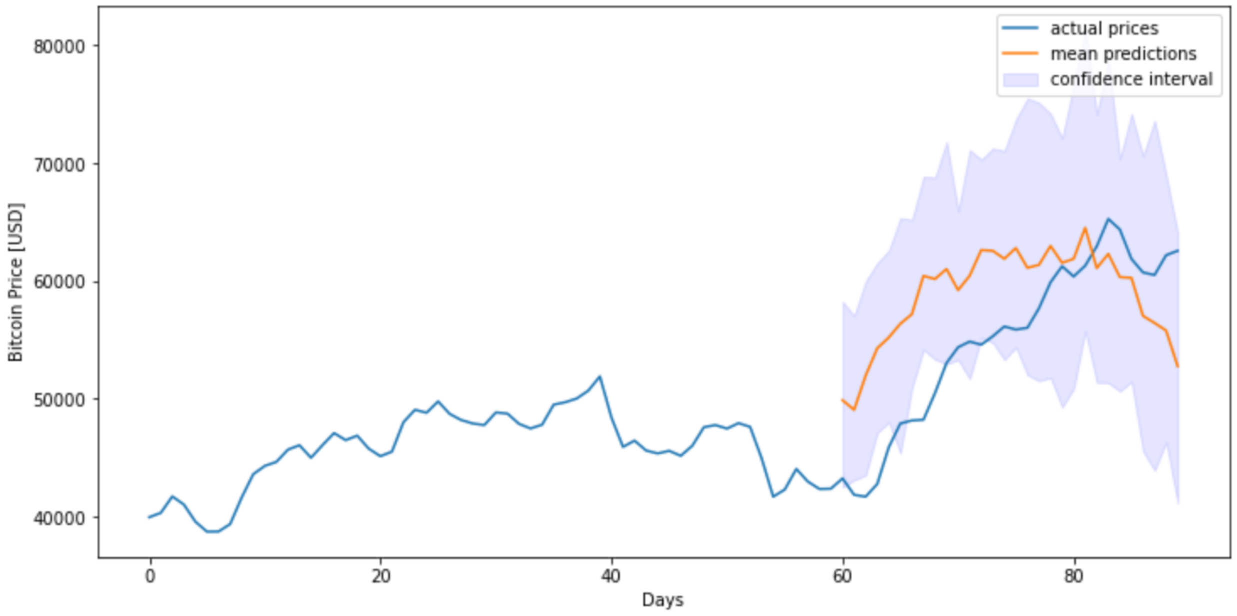
RMSE	4559.28641501311
mean_absolute_QuantileLoss	252814.03333333335
mean_wQuantileLoss	0.0511922787673776
wQuantileLoss[0.1]	0.026615664524137515
wQuantileLoss[0.2]	0.035788473100831684
wQuantileLoss[0.3]	0.04361698768198024
wQuantileLoss[0.4]	0.05539865067286393
wQuantileLoss[0.5]	0.06399183037893764
wQuantileLoss[0.6]	0.06572183859800423
wQuantileLoss[0.7]	0.06510322582670063
wQuantileLoss[0.8]	0.06001858800678422
wQuantileLoss[0.9]	0.04447542210039976

4. Evaluation, Benchmark Comparison

For the evaluation of the model I created a predictor and deployed it as an endpoint with configured JSON serializer and deserializer so that I can feed it with JSON data and it will return a nice JSON.

The source dataset only provided Bitcoin prices until March 2021, so I decided to test the predictor with latest prices. For this I created a method that fetches the latest daily Bitcoin prices up to a defined number of days in the past (https://min-api.cryptocompare.com).

I fetched the Bitcoin prices of the last 90 days. The first 60 were used as context data input for the predictor endpoint and the last 30 days to compare the predictions with actual price values. In the request I also defined that it should also return the upper and lower decile quantiles (q10 and q90) in addition to the mean prediction value.



The resulting graph shows the closeness between the actual prices (blue) and the predictions (yellow). The mean predictions were mostly not so close to the actual prices but were at least following the same trend most of the time. But the actual price was at least most of the time within the quantile area, which was good.

The results look already promising, but it's still interesting to combine them with other time series forecasting algorithms/methods.

For comparison I sourced benchmark results from the paper Statistical and Machine Learning forecasting methods: Concerns and ways forward' from Spyros Makridakis, Evangelos Spiliotis, Vassilios Assimakopoulos (https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0194889).

They Compare statistical methods with machine learning algorithms for time series forecasting. They use the Symmetric mean absolute percentage error (sMAPE) to compare the performance of the algorithms.

I used this formula to calculate it for my results :

$$\text{SMAPE} = \frac{100\%}{n} \sum_{t=1}^{n} \frac{|F_t - A_t|}{|A_t| + |F_t|}$$

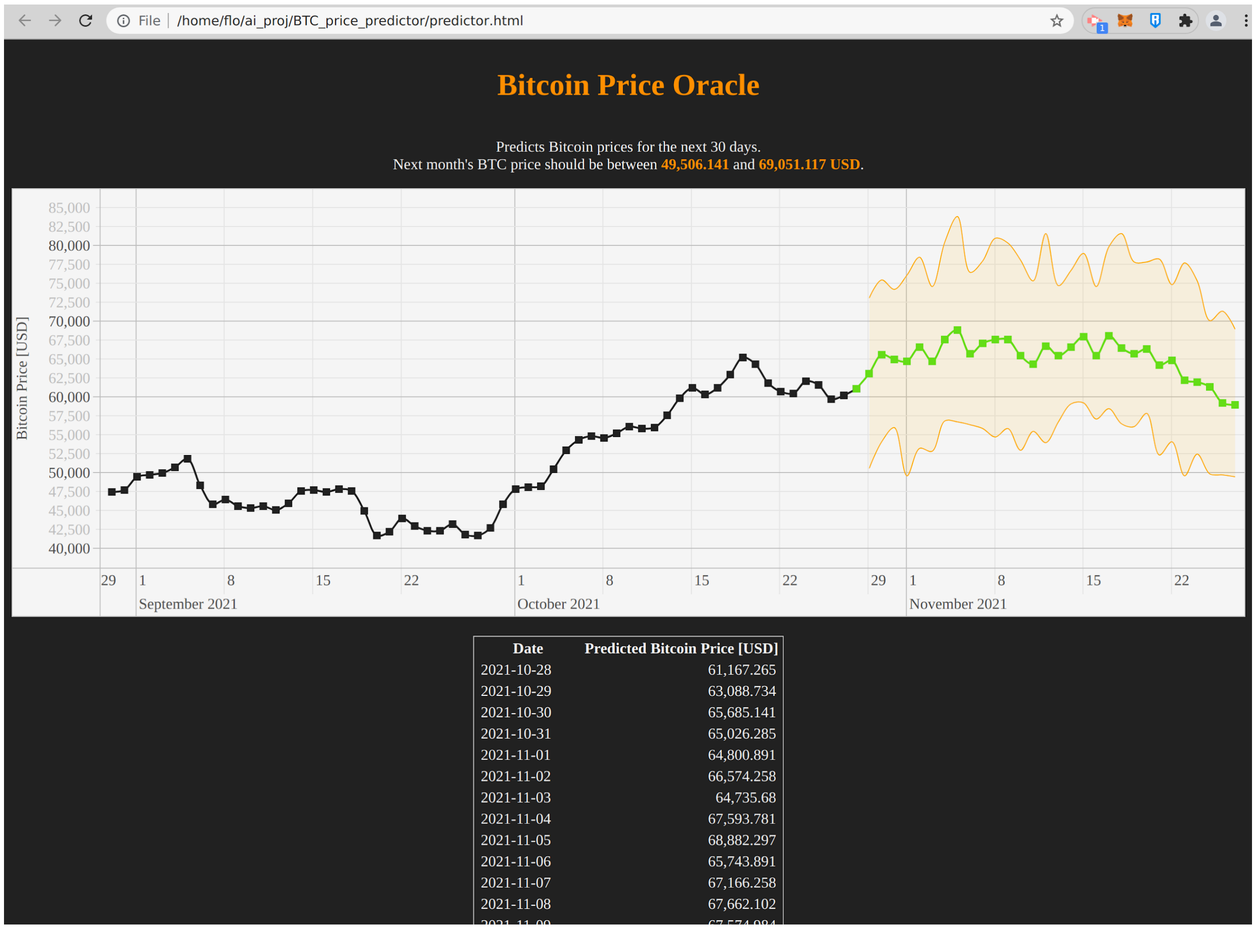
F is the forecast values, A the actual values and n the number of values of A and F (they must have the same length).

Rank	Method	sMAPE(%)
x	**My DeepAR**	**5.55**
1	Multi-Layer Perceptron (MLP)	8.34
2	Bayesian Neural Network (BNN)	8.58
3	Gaussian Processes (GP)	9.62
4	Generalized Regression Neural Networks (GRNN)	10.33
5	K-Nearest Neighbor regression (KNN)	10.34
6	Support Vector Regression (SVR)	10.40
7	CART regression trees (CART)	11.72
8	Radial Basis Functions (RBF)	15.79

From the results it looks like I scored better than all of them, but I must admit that I calculated the score with the context length that yielded the best predictions. I would probably need to create multiple sMAPE with different time series for predictions and average the results to get a more realistic one. But having the best result just looks nice.

5. Web App, Deployment

For the web application I created a static web page displaying the past two months of Bitcoin prices and the predictions for the next month including the confidence interval between q10 and q90 of the predictions.



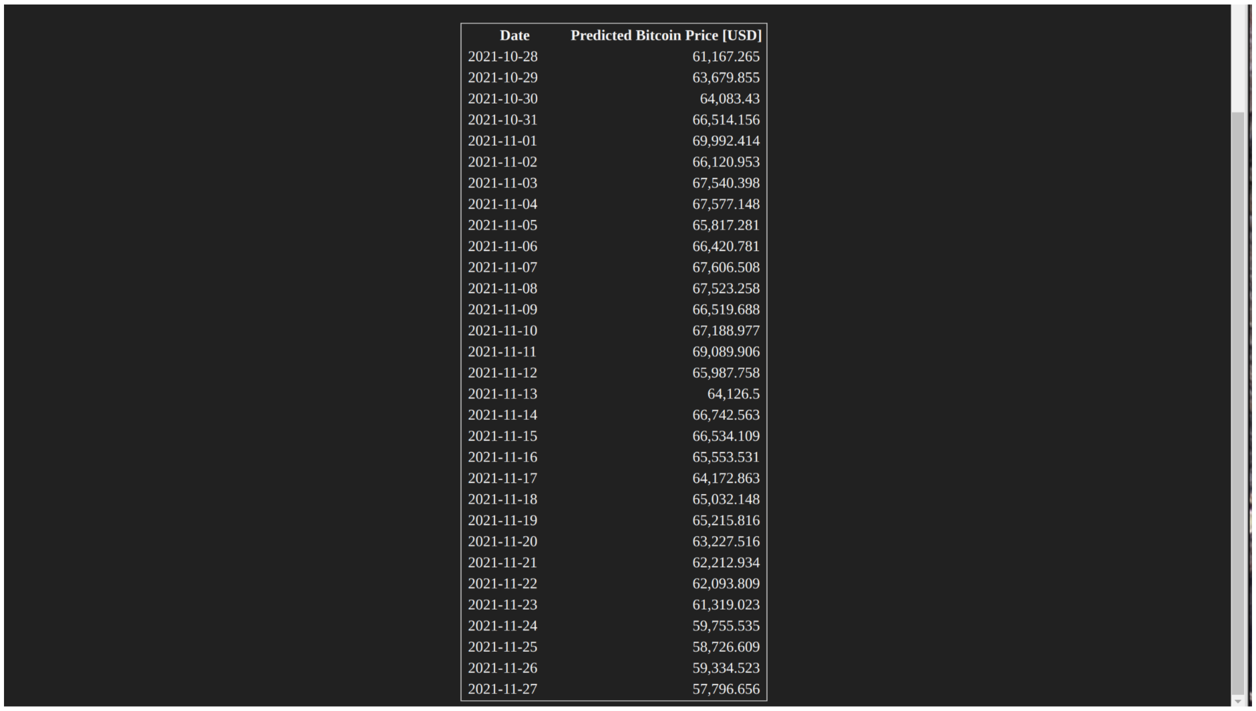
Bitcoin Price Oracle

Predicts Bitcoin prices for the next 30 days.
Next month's BTC price should be between **49,506.141** and **69,051.117 USD**.

Date	Predicted Bitcoin Price [USD]
2021-10-28	61,167.265
2021-10-29	63,088.734
2021-10-30	65,685.141
2021-10-31	65,026.285
2021-11-01	64,800.891
2021-11-02	66,574.258
2021-11-03	64,735.68
2021-11-04	67,593.781
2021-11-05	68,882.297
2021-11-06	65,743.891
2021-11-07	67,166.258
2021-11-08	67,662.102

For the backend I created an AWS Lambda function which is fetching the last 60 daily bitcoin prices, formats it nicely to call the deployed model endpoint and returns the past data with the predictions (including mean, quantile 10 and quantile90 prices) in a nicely formatted way for the page to display it. The Lambda is exposed through an created GET endpoint of an API Gateway, which the web page is using to fetch the data.

The bottom of the page contains a table of all predicted prices (mean prediction).



Date	Predicted Bitcoin Price [USD]
2021-10-28	61,167.265
2021-10-29	63,679.855
2021-10-30	64,083.43
2021-10-31	66,514.156
2021-11-01	69,992.414
2021-11-02	66,120.953
2021-11-03	67,540.398
2021-11-04	67,577.148
2021-11-05	65,817.281
2021-11-06	66,420.781
2021-11-07	67,606.508
2021-11-08	67,523.258
2021-11-09	66,519.688
2021-11-10	67,188.977
2021-11-11	69,089.906
2021-11-12	65,987.758
2021-11-13	64,126.5
2021-11-14	66,742.563
2021-11-15	66,534.109
2021-11-16	65,553.531
2021-11-17	64,172.863
2021-11-18	65,032.148
2021-11-19	65,215.816
2021-11-20	63,227.516
2021-11-21	62,212.934
2021-11-22	62,093.809
2021-11-23	61,319.023
2021-11-24	59,755.535
2021-11-25	58,726.609
2021-11-26	59,334.523
2021-11-27	57,796.656