Predicting price volatility of Bitcoin by analyzing the frequency content of returns

ML in Finance - Capstone Project

Gersh Yagudaev

10/24/2022

Contents

Figures

Summary

In this project, a set of ElasticNet models were used to predict price volatility of Bitcoin by analyzing the frequency content of returns. The ultimate aim of this is to generate a trading strategy that can outperform a simple buy-and-hold approach based on the model's predictions. The data used for this is BitFinex's 400+ cryptocurrency trading pairs at 1 minute intervals.

The model aims to predict the frequency composition of the next 4 days of Bitcoin's returns, and uses this predicted frequency composition to estimate the variance of the returns.

The resulting model achieved good Out Of Sample performance, with an R2=0.2955. A simple trading strategy tested based on this model consistently outperforms Bitcoin's spot returns provided that certain underlying assumptions about Straddle prices are met. It is important to note that due to the simplicity of the trading strategy and the severity of its assumptions, it should not be used to make any conclusions about real-world trading utility of the model.

Background Information

The Short Time Fourier Transform (further – STFT) is used extensively in this project. It is a "Fourier-related transform", that allows the user to determine the sinusoidal frequency content of local sections of a signal as it changes over time.

The purpose of this section is to familiarize the reader with this transform. We discuss the Fourier Transform, why it is not directly applicable to our data, the superiority of the STFT, and the Spectrogram representation of the STFT which we will use throughout the report.

Fourier Transform (FT)

Definition

The Fourier Transform is a fundamental building block of the STFT. It is a mathematical transformation that converts a function of time into a function of frequency. An example could be the conversion of the waveform of a musical chord into a list of the pitches present in the chord, and their intensity.

When talking about a function of time, we refer to that function as existing in the *Time Domain*, and when talking about the same function in frequency (i.e after the Fourier Transform) we refer to it as being in the *Frequency Domain*. "the transformed function encodes how much of every frequency (i.e how much of a sinusoid of a particular frequency) is present in the signal." – Dr. Shonlieb (Cambridge University)

Mathematically, the transform is defined as:

$$X(\omega) = \int_{-\infty}^{\infty} x(t) \cdot e^{-j\omega t} dt$$

And the inverse transform:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\omega) e^{j\omega t} d\omega$$

Where:

$$X(\omega) = \text{function in Frequency Domain}$$

$$x(t) = \text{function in Time Domain}$$

$$j = \sqrt{-1}$$

$$\omega = \text{frequency}$$

$$t = \text{time}$$

Intuition

We can graphically illustrate the intuition behind this:

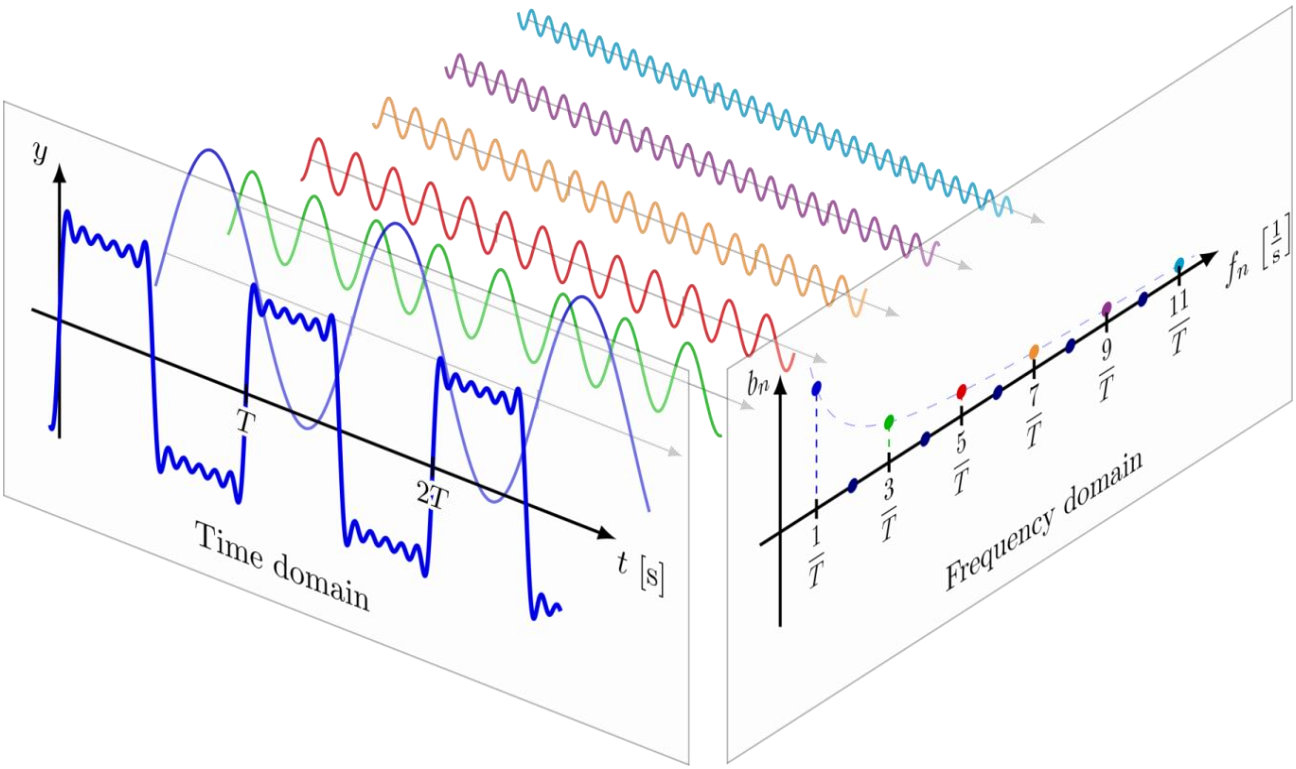


Figure 1 Fourier Transform

The above figure graphically demonstrates the operation of the FT. We start with a Time-Domain signal on the left-hand pane. Then, we expand it into a series of sinusoids, each with a different frequency and amplitude. The expansion is such, that if we were to add up all the sinusoids, their sum would equal the original function.

Now, a sinusoid is fundamentally expressed as $A \cdot \sin(\omega \cdot t)$, where A=amplitude, w=frequency, t=time. Since the sinusoid is a *repeating* function of time, we can characterize it as simply an amplitude and a frequency (A and w). This characterization can be observed on the right-hand pane of the figure, where we observe several plotted points. Notice that the height of each point corresponds to the amplitude of the relevant sinusoid, and its position on the Frequency axis is dictated by the frequency of the sinusoid.

An example of the benefit of the FT is the ability to go from a complicated, difficult to analyze square-like wave in Time Domain, to a nice sequence of points in the Frequency Domain. Looking at this sequence of points, we can immediately tell that the dominant frequency in our Time-Domain function is $\frac{1}{T}$, since that point in the frequency domain has the highest value. We can also see that frequency components from $\frac{5}{T}$ and onwards have minimal impact, due to their low

amplitude. As such, we could approximate the time domain function by only keeping the first two components of the Fourier Transform (first 2 sinusoids), and inverting the transformation.

Different time-domain signals map to different functions in the Frequency domain:

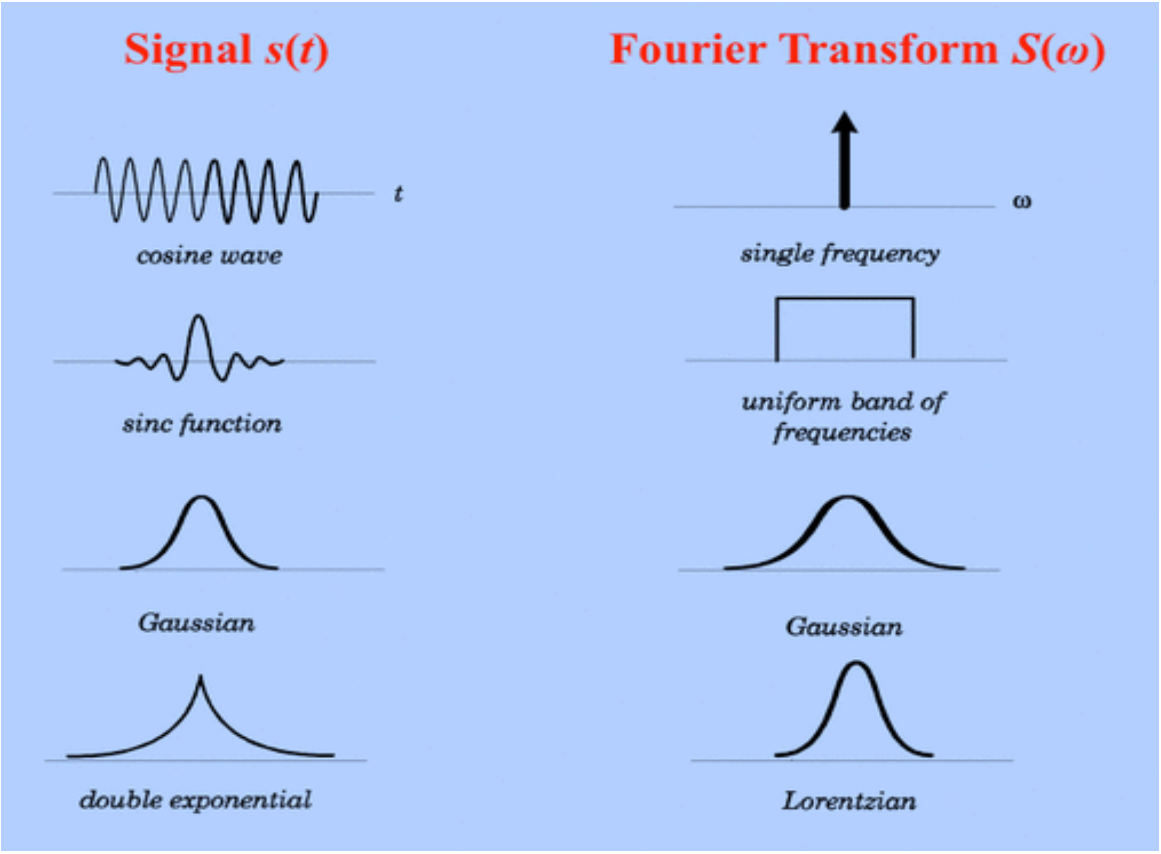


Figure 2 Fourier Transform Examples

In the figure above, notice that the cosine wave is represented by a single vertical line in frequency. This is because the Fourier Transform decomposes time-domain functions into a sum of sines and cosines, and expresses these sinusoids as points on the amplitude-frequency plane. Thus, a single sinusoid would be a single point.

Continuous VS Discrete

An important distinction to make is that the discussion above refers to the *continuous* time Fourier Transform. In other words – the inputs to the transform have to be analytic functions that are continuous in time. Therefore – actual, real world data cannot be directly used, as it is presented as samples in time, and is therefore discrete.

However, the intuition for discrete transforms is exactly the same, with slight changes in the underlying mathematics which are not relevant to this report.

Stationarity Problem

Despite the powerful nature of this mathematical tool, it is not directly applicable to price or returns data, as the transform requires stationarity. A stationary signal is defined such that its statistical properties, i.e mean, variance, etc remain static over its lifetime. This requirement exists, because all the FT is doing is matching a sequence of sines and cosines to approximate a given function in their sum. If the function is not stationary, we have a problem, because our basis functions are stationary. At best, it would take an “infinite” number of sinusoids to create an approximation.

If we used the Fourier Transform, then our entire model would be predicated on the assumption that there are no changes to variance, and therefore nothing to predict.

The figure below illustrates, what happens when stationarity is not respected:

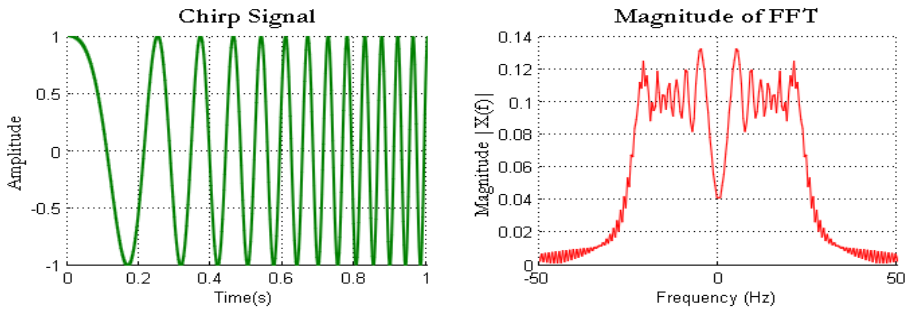


Figure 3 FT of Chirp Signal

Our time domain signal is a Chirp, which is a sinusoid whose frequency increases with time. Thus, its variance is changing over time. Taking a look at the FT, we observe a chaotic signal that is arguably harder to analyze than the time domain equivalent. This problem brings us to the STFT…

Short-Time Fourier Transform (STFT)

Definition

The STFT is a sequence of Fourier Transforms of a windowed signal. Therefore, instead of providing a singular "readout" of frequencies present in the signal throughout its entire duration, it instead generates information about the frequency content over time. In other words, instead of completely sacrificing time-information for full frequency-information, we sacrifice *some* time information for *some* frequency information.

The STFT is defined as:

$$X_{STFT}[m,\omega] = \sum_{n=-\infty}^{L} x[n]W[n-m]e^{-j\omega n}$$

Where n represents discrete time, and ω represents discrete frequency (since we are operating on samples of data rather than input & output functions)

This definition states, that $X[m,\omega]$ is the STFT of a time-domain signal x[n], which is obtained by cutting x[n] into several overlapping chunks (the overlap is necessary to reduce artifacts at the boundaries). This slicing is achieved by the multiplication $x[n]W[n-m]$, where W is some window function (which could be something as simple as a single square pulse). At each iteration of the sum, the window will "scan" along x[n], picking out a new slice.

These slices then individually undergo a standard Fourier transform performed by the $e^{-i\omega n}$ term.

The inverse STFT in discrete time has a more complicated definition, so we will omit it from discussion. However, it does exist, and the operation is readily accessible in Python.

Intuition

As mentioned earlier, the STFT is a sequence of standard Fourier Transforms, which are performed on small overlapping slices of some time-domain signal. This is useful, as it gets around the stationarity problem. A signal that isn't stationary across its lifetime is likely locally stationary if sliced into small enough pieces.

In other words – if stationarity means fixed mean & variance, then by slicing our data (price or returns) into small enough pieces, we end up with a sequence of transforms which, for the most part, do not include non-stationary events, therefore minimizing the impact of the problem.

We can observe the STFT flow diagrammatically:

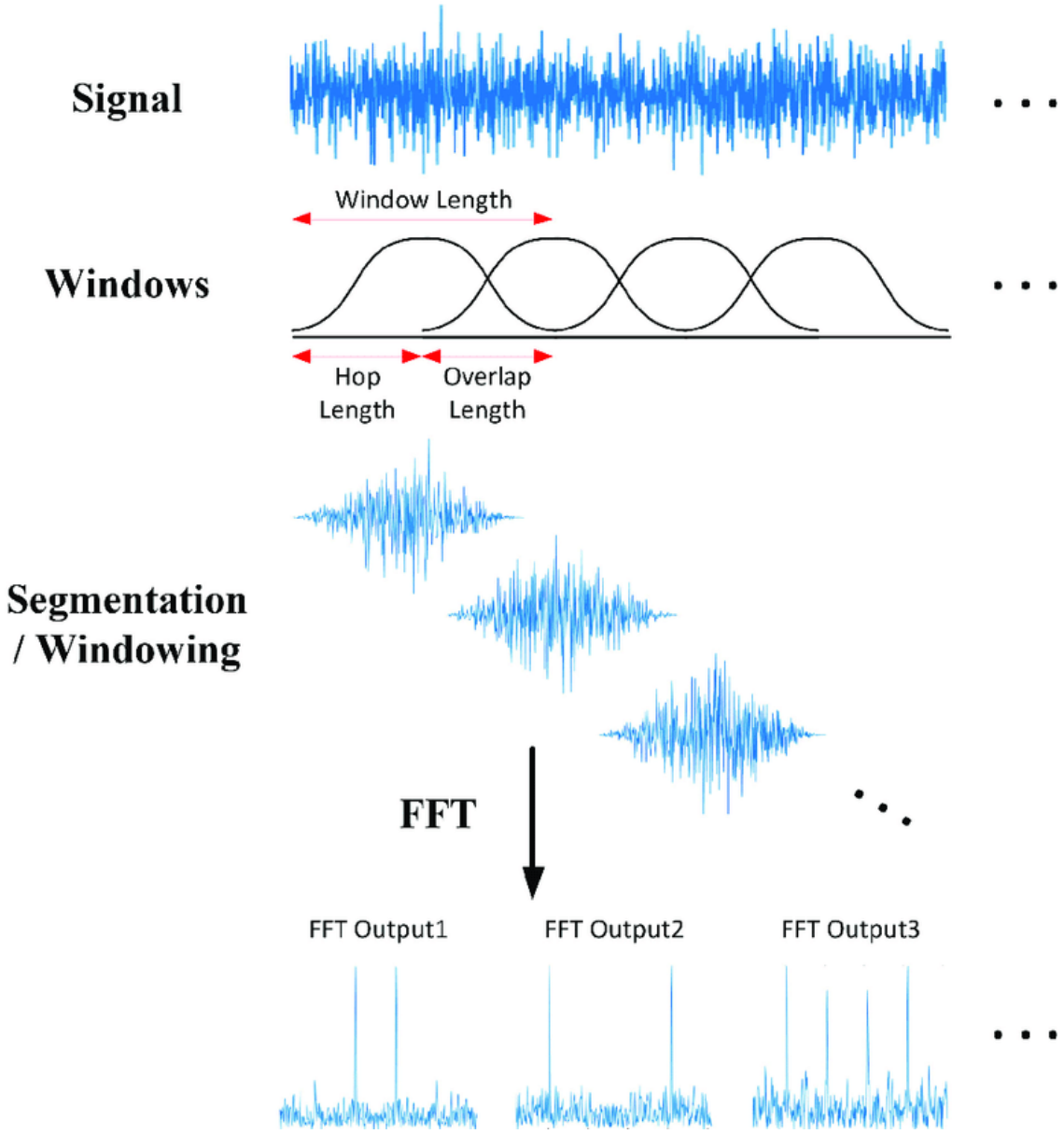


Figure 4 STFT

We start with a time-domain signal $x[n]$. Now, we need to generate a slice, which we do by multiplying it with a window function. This results in a small "wavelet" of signal, which rapidly attenuates at the window edges. Then, we compute the FT of this frame (FFT referenced in the image is the "Fast Fourier Transform" – a specific FT algorithm). As a result, we can now characterize the frequency content of *that specific frame of time*.

We then shift the window forward in time, and repeat the process.

In the end – we obtain a sequence of spectra:

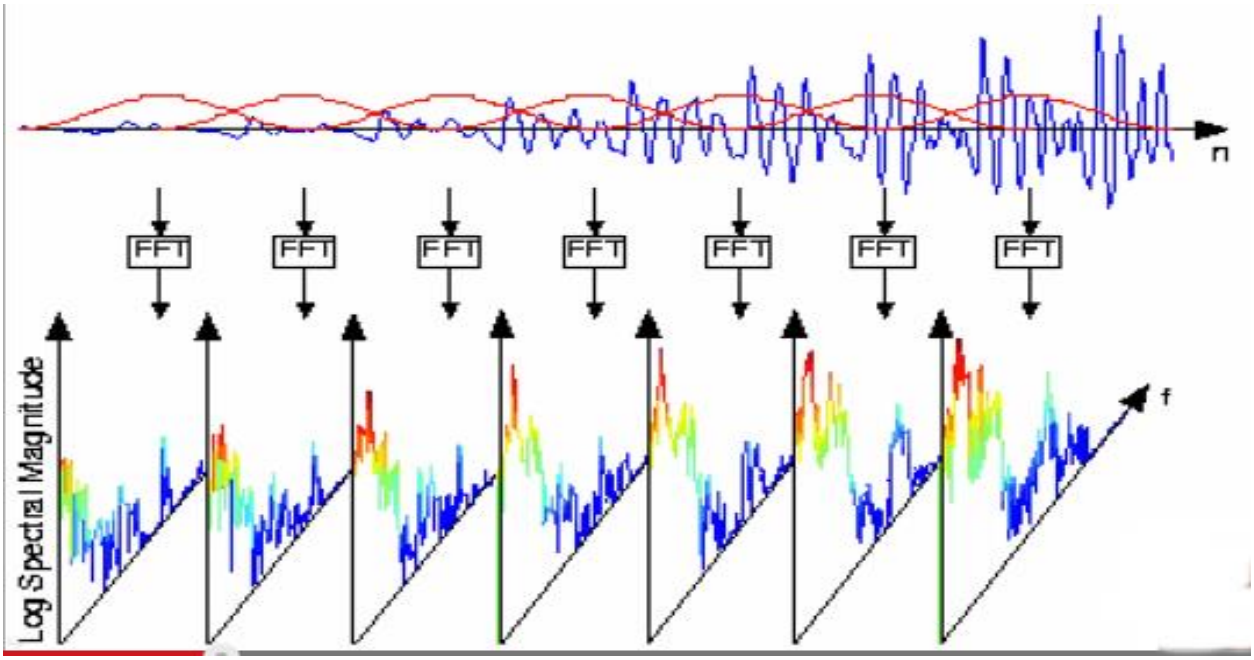


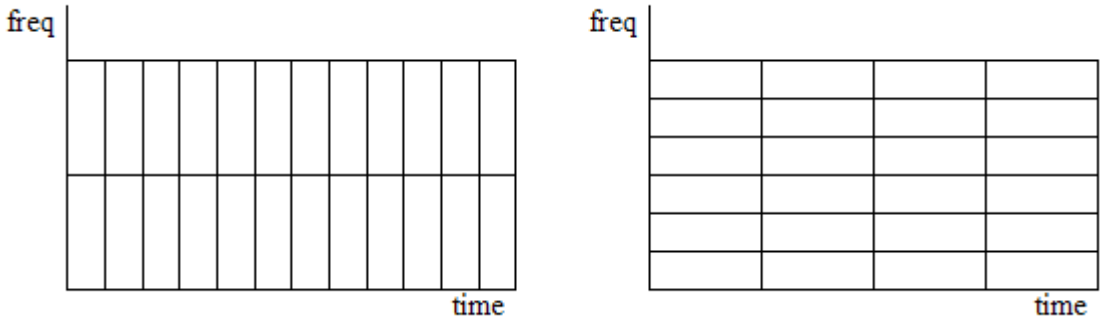
Figure 5 STFT Process Diagram

For each of these spectra, we have a good understanding of their frequency content, but it is not perfect due to artifacts introduced by the window edges. We also know where each spectrum is located in time. However – we lost time resolution, because we don't know what happened during the period encompassed by the window.

Parameters

The STFT has several parameters that significantly affect the results:

1. Window Type: Depending on window shape, we have a tradeoff between frequency dynamic range and resolution (how many frequencies can we detect, vs how precise are we in measuring their power)
2. Window Length: Must be a power of 2. Introduces a tradeoff between frequency resolution and time resolution. A smaller window gives us better time granularity, but lower quality Fourier Transforms, and vice versa, as shown below:



a.

Figure 6 STFT Resolution Issue

Left->small window, Right->large window

3. Window Overlap: Tradeoff between filtering out transients at window edges vs "smearing" data by introducing too much redundancy

The Spectrogram

The spectrogram is a special representation of the STFT, defined as:

$$\text{SPEC} = |X_{STFT}[n, \omega]|^2$$

I.e, the spectrogram is simply the squared magnitude of the STFT. It represents the power density spectrum. In other words, tells us about the power of various frequencies over time.

For example, the spectrogram of a chirp signal is:

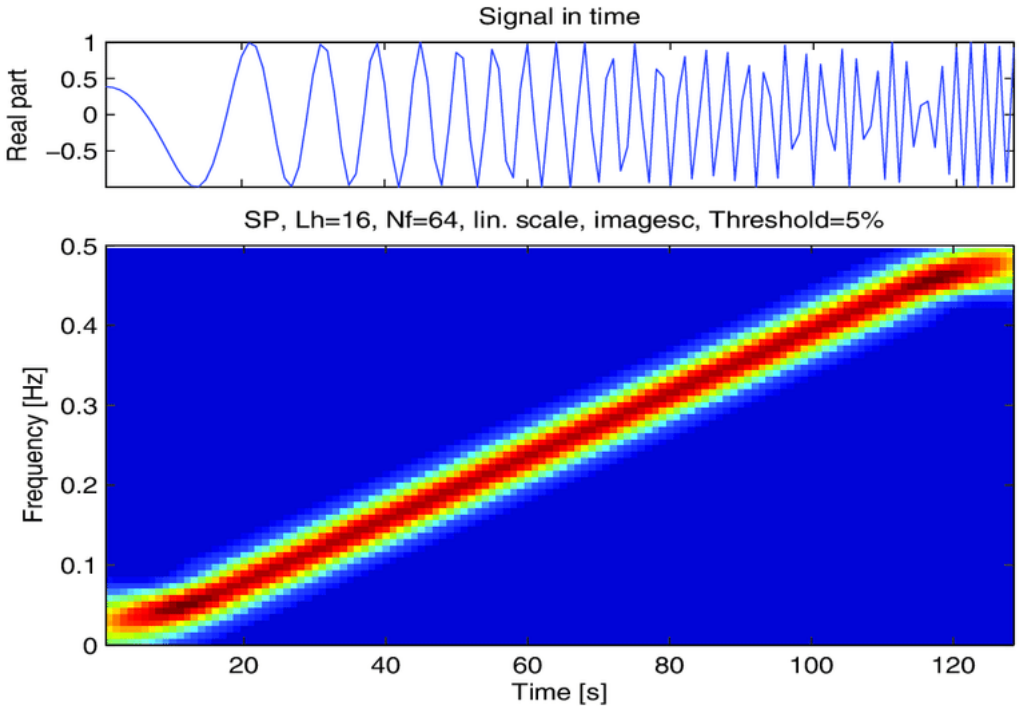


Figure 7 Spectrogram of a Chirp

The frequency of the sinusoid rises with time, so we observe an ascending line. Notice that the line is not perfect, but instead we see an “aura” around the dark “true” line in the center. This is a result of sacrificing some frequency resolution to maintain time information, as well as windowing artifacts.

Another way to look at a spectrogram is as a 2D table:

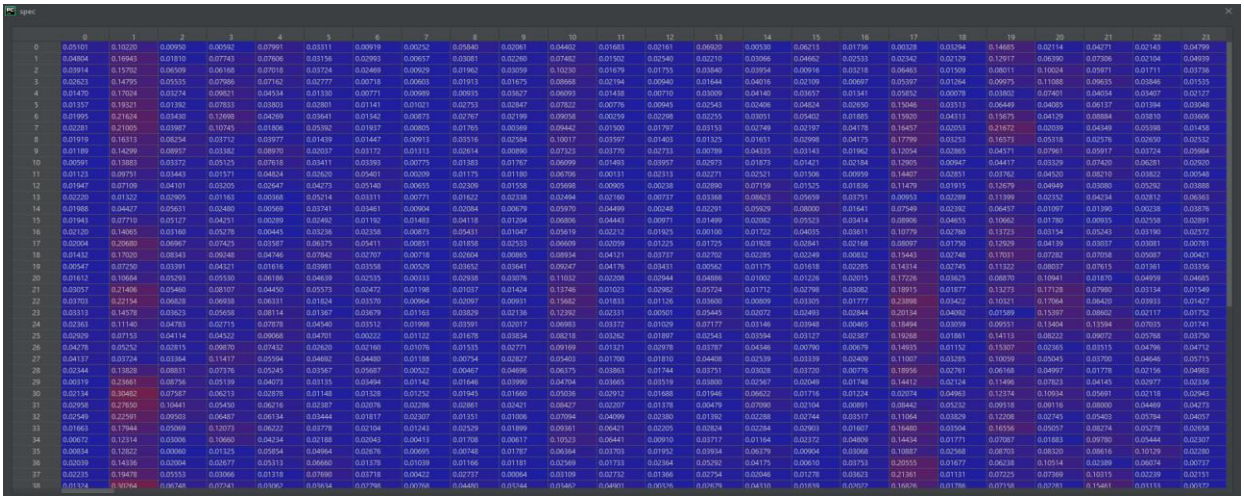


Figure 8 Spectrogram Representation in Python

The row index represents a specific frequency, column index represents an instance in time, and the cell value represents power. Thus – we have information on the power of various frequencies over time.

Problem Definition & Approach

The Question

This work attempts to **predict price volatility in crypto currencies by analyzing the frequency content of returns**. In other words, the goal is to try and guess the future variance of Bitcoin based on its historical spectral information.

The value of investigating this is twofold. First, the benefit of spectral analysis has been demonstrated numerous times for traditional assets. This has specifically been done in the context of variance, such as in the figure below:

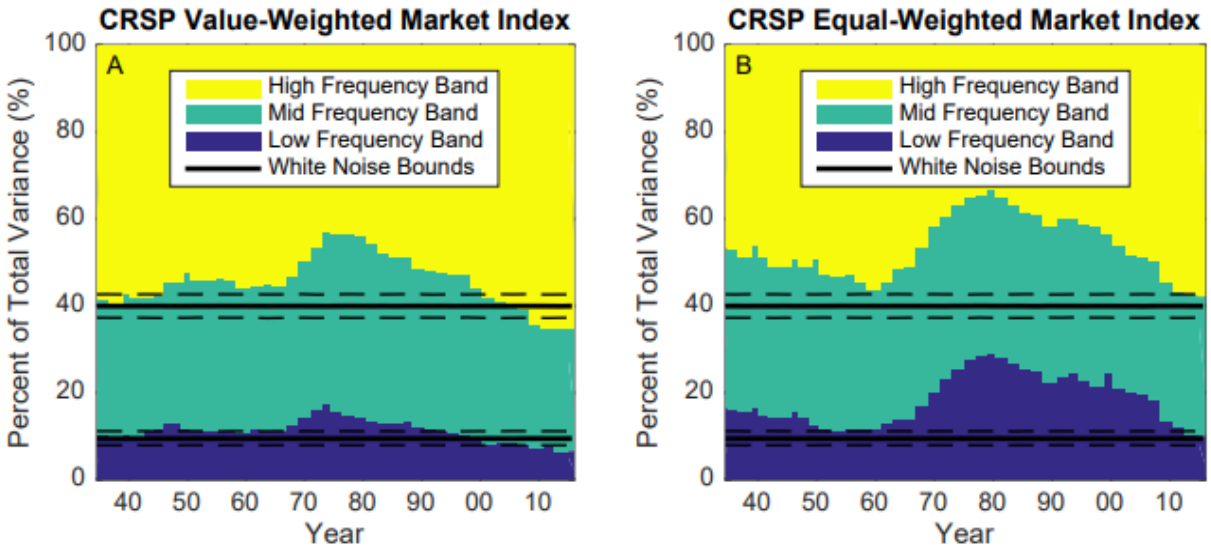


Figure 5: Spectral decomposition of the 10-year rolling sample variance of the daily returns of CRSP value-weighted market index (panel A), and the CRSP equal-weighted market index (panel B) from 1926 to 2015. Frequency components are grouped into 3 categories: high frequencies (more than 1 cycle per week), mid frequencies (between 1 cycle per week and 1 cycle per month), and low frequencies (less than 1 cycle per month).

Figure 9 Spectral Decomposition of CRSP

We can observe that the majority of the variance of daily returns for the CRSP index is actually composed of high frequency components. We can see interesting patterns, such a higher dominance of low-frequency components in the Equal-weighted index versus the market-weighted index. This would, for example, imply that the equal-weighted index is more vulnerable to shocks (which according to this study's definition are low frequency events).

While this does not directly predict volatility, it proves that information is contained in the spectrum of price & returns data.

Second – predicting volatility is akin to predicting price, and enables us to directly trade based on the outputs of our model using Straddles.

Finally – entire textbooks support spectral analysis of financial data:

"It has been observed that the properties of financial securities are not constant, but change over time. Economic shocks produce distinct effects over different time horizons.

However, the traditional inputs into portfolio analytics— means, variance, covariance, alphas, and beta—are static, and do not distinguish between the short- and long-term components of these dynamics.

"Spectral Portfolio Theory" - Shomesh E. Chaudhuri and Andrew W. Lo

The Data

This work uses the BitFinex cryptocurrency dataset, which contains the exchange rates of 400+ cryptocurrency pair at 1-minute resolution. Only the BTCUSD pair is used.

Further, only data from 2017 and onward is used, due to the presence of "holes" in times prior. This section of data has to be omitted, because our frequency analysis tooling (the STFT & Spectrogram) cannot accommodate holes in data. Attempts to interpolate or otherwise fill in the missing data would result in incorrectly detected frequencies, and simply dropping NAs doesn't work as the model will interpret "stitched" as next to each other despite missing data, which again introduces frequency inaccuracies.

Approach

The general idea is to predict future volatility based on past volatility of returns. We use volatility of returns, and not volatility of price, since price is extremely non-stationary, which requires the STFT slices to be extremely small. Thus, we would have virtually no resolution in frequency making spectral analysis pointless.

To implement the idea, a set of ElasticNet models was created, which in aggregate output a prediction for the frequency spectrum (spectrogram frame) of four future days. An original spectrogram is computed on the input data, and contains 65 frequencies. Each of these frequencies is used as a factor, in addition to Root-Mean-Square-Energy (RMSE) and Spectral Roloff.

Each ElasticNet is responsible for predicting the value of a single frequency component in the future. We can generate a full prediction by constructing a new STFT frame using aggregated predictions of the 65 ElasticNets. We then compute the inverse STFT, which gives us a time-domain signal – a prediction of "price". This intermediary result is a very poor predictor of actual prices, since we don't have any phase information. However, it *is* a good predictor of volatility. Thus – we calculate the volatility of our predicted price signal, and this is our *actual* prediction.

Direct prediction of variance, without intermediary frequency prediction has also been attempted, but achieved extremely poor results. The author theorizes that this is due to the strongly nonlinear relationship between frequency and variance. ElasticNet is fundamentally a modified OLS. While OLS is capable of modeling nonlinear phenomena based on the choice of factors, in this case it would not make sense since not a single factor used in our model (except perhaps RMS energy) can have some sort of linear mapping to variance.

Due to the presence of extra steps between model output and actual prediction, built-in model evaluation tools are not applicable. To evaluate the quality of the model a true out-of-sample test is performed. Further, a trading strategy is defined and tested based on the output of the model.

Experimental Evaluation

Methodology

Algorithm of Actions

In this project, multiple ElasticNets are implemented, predicting 65 frequency components 4 days in the future based on a spectrogram and some spectral features. The algorithm flow is as follows:

1. Import & Preprocess the data
    a. Remove duplicate entries
    b. Resample to 1H timescale
    c. Calculate hourly percentage returns
    d. Calculate hourly variance
    e. Calculate variance over running 4-day windows
2. Compute a spectrogram of the data
3. Calculate spectral features
    a. RMS Energy
    b. 1%, 50%, 99% Spectral Roloff
4. Create a prediction target dataframe by shifting the spectrogram forward 1 unit in time (lead)
5. Create an input dataset by merging the original spectrogram with the spectral features
6. Define 65 ElasticNet models (due to having 65 frequencies), such that each model predicts one of the 65 frequency bins based on the input dataset
7. Perform cross validation to determine Hyperparameters
8. Train the models on the first 80% of the dataset, withhold the remaining 20%
9. Perform a running test on the remaining 20%
    a. Generate a prediction for the next Spectrogram frame
    b. Compute the inverse STFT of the prediction
    c. Calculate the variance of the inverse STFT
    d. Append the “true” frame to the known 80%
    e. Repeat until the OOS test is complete
    f. Upon completion of test compare predicted variance to ground truth & naïve prediction
10. Define & Test an OOS trading strategy

Model Walkthrough

First, we import and preprocess the data. This involves resampling to 1H timescale, cleaning up the dataframe, and computing our variance measures. We end up with the following:

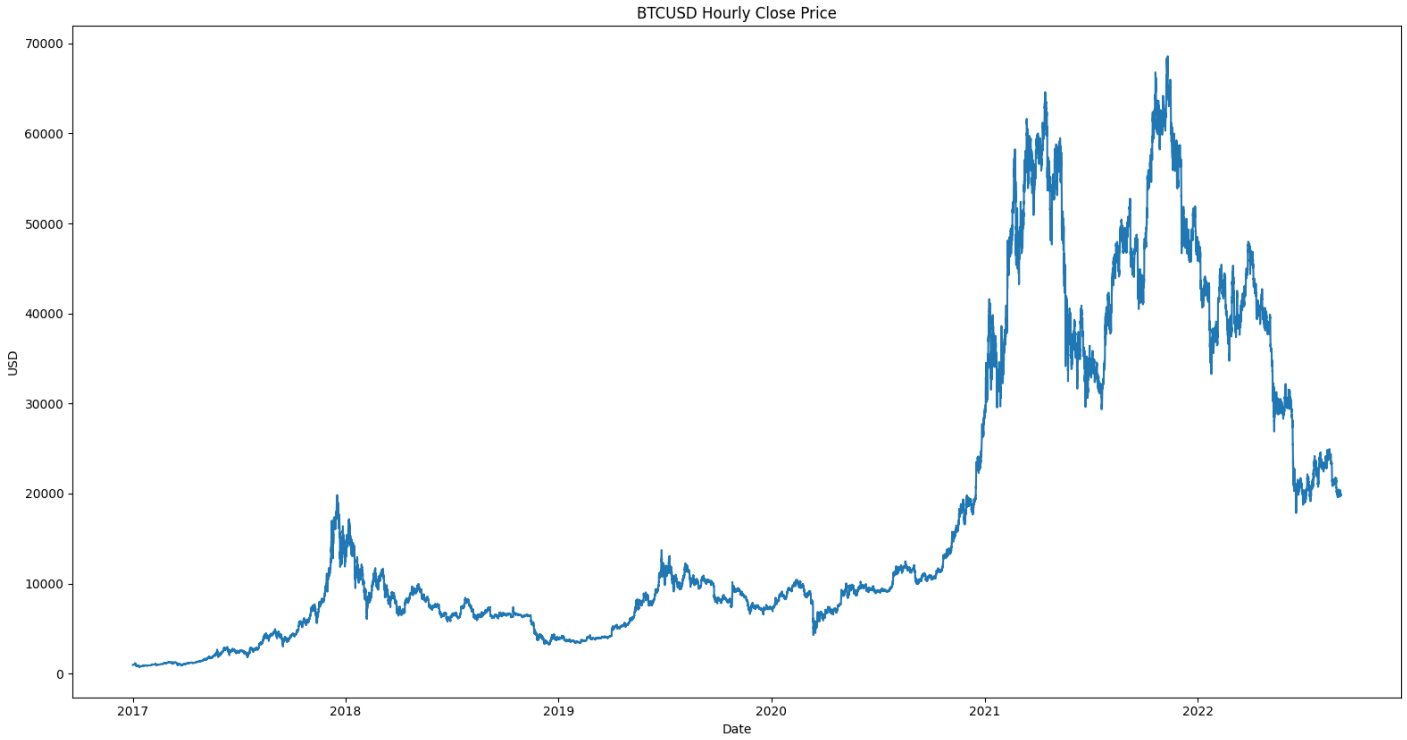


Figure 10 BTC Hourly Close

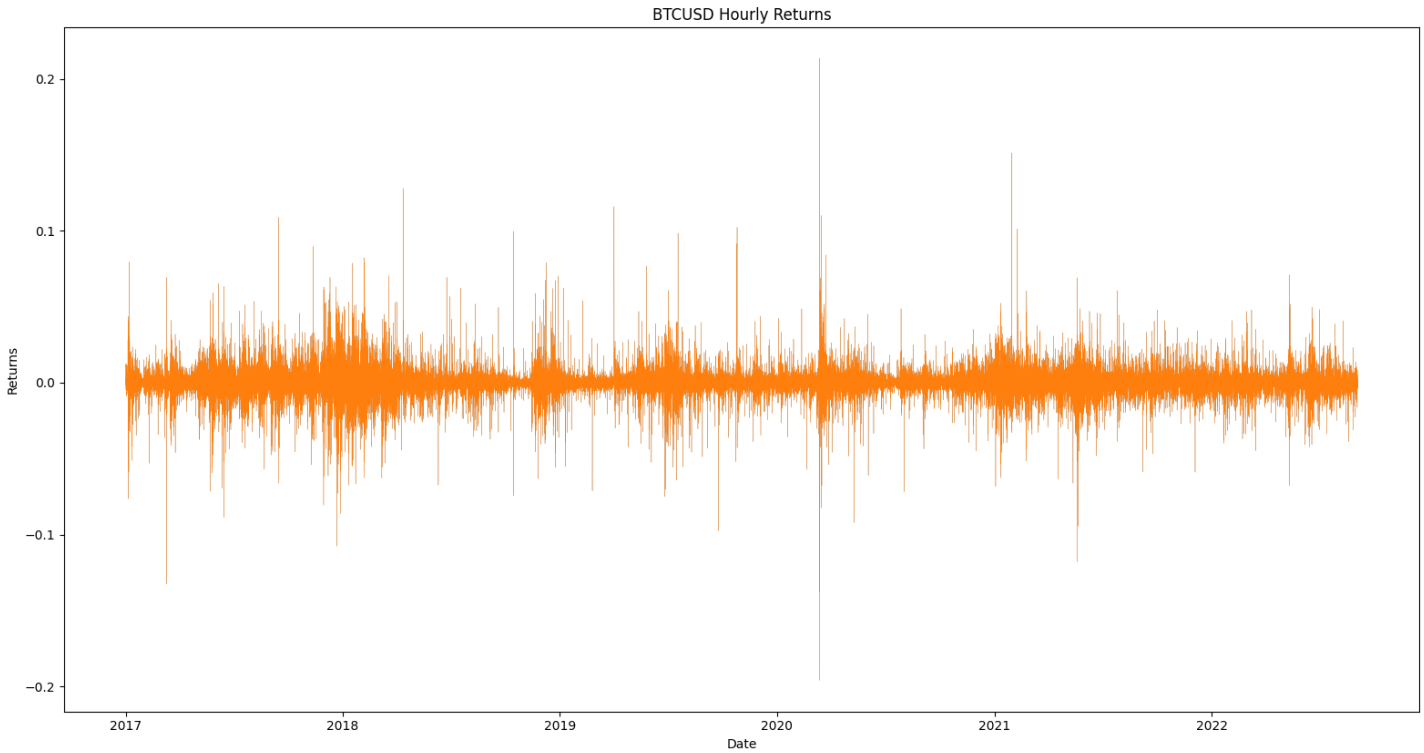


Figure 11 BTC Hourly Returns

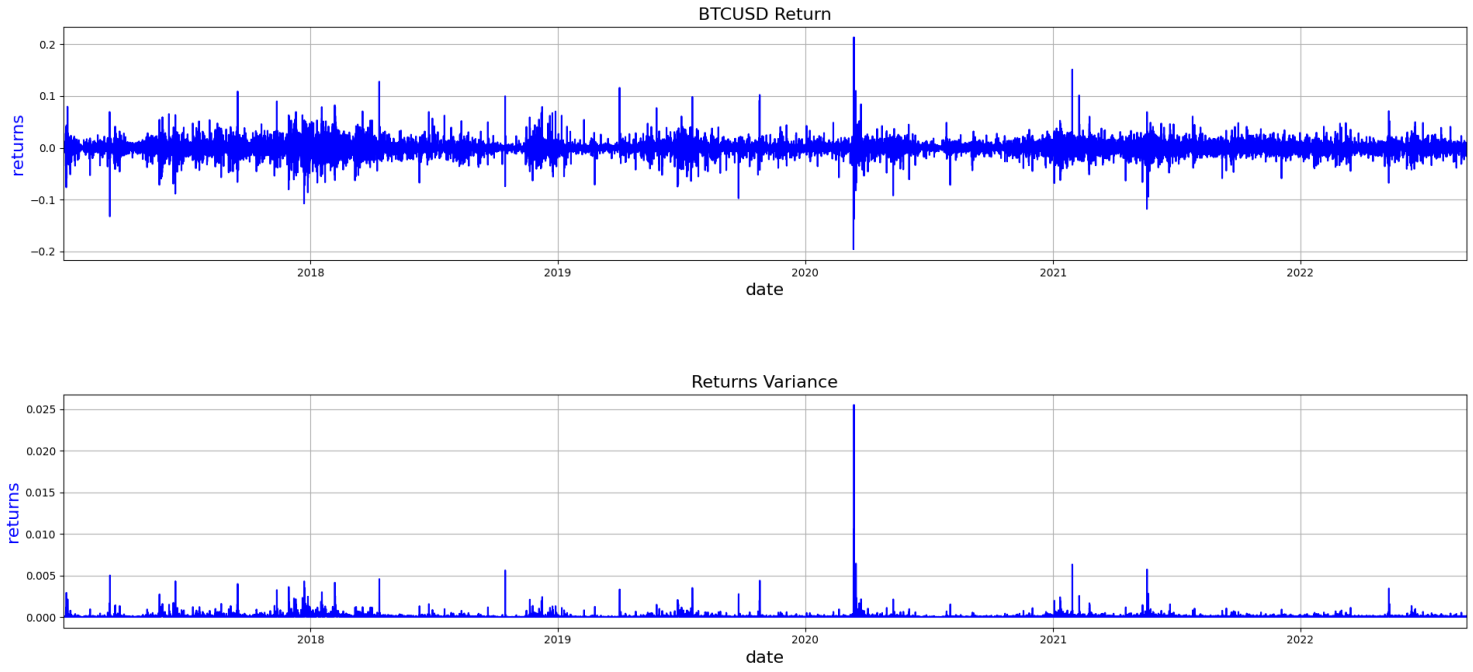


Figure 12 BTC Returns & Returns Variance

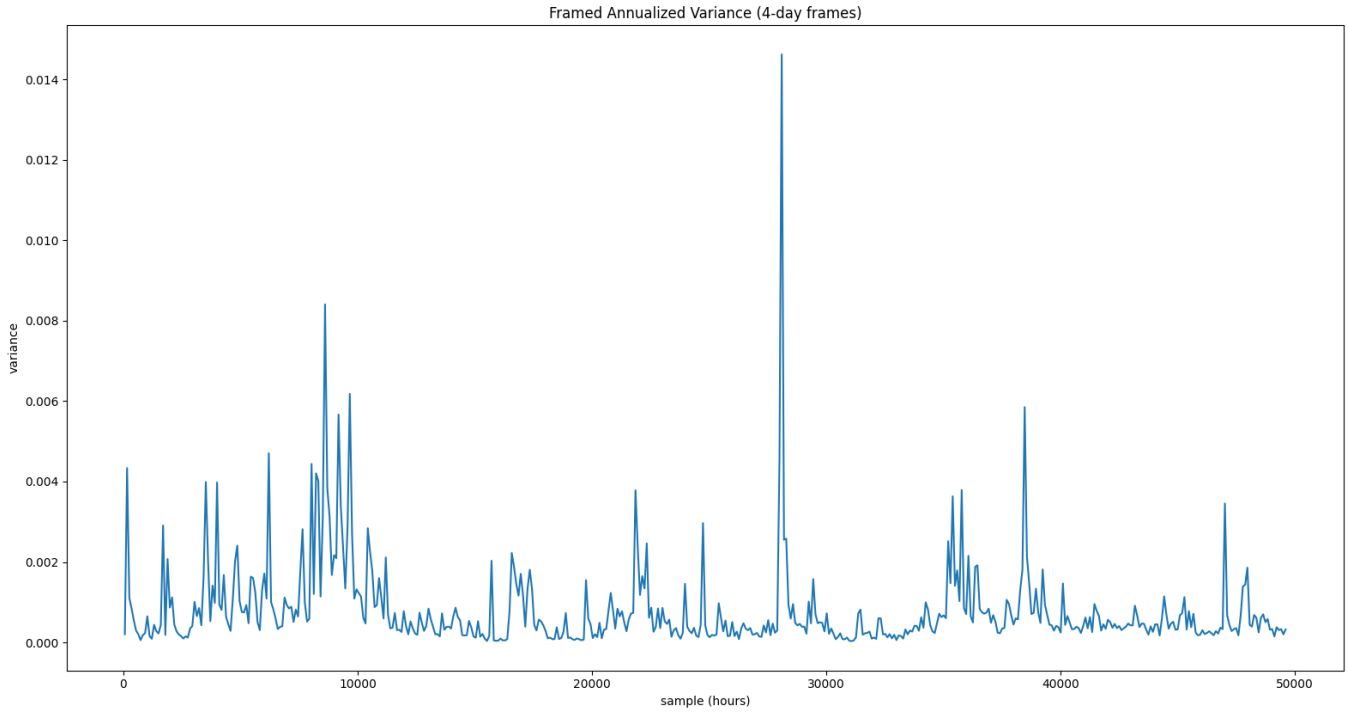


Figure 13 Framed Annualized Variance

Then, we need to compute a spectrogram, which involves selecting parameters such as window length, window type, and overlap size. After multiple iterations, the most success was found with a Hann window of length 128 and an overlap of 96. Note that this also means that each frame contains 96 hours, which is 4 days. Thus, each vertical line on the spectrogram actually contains 4-days of information. As a result, we will be unable to make predictions at a higher temporal resolution than 4 days.

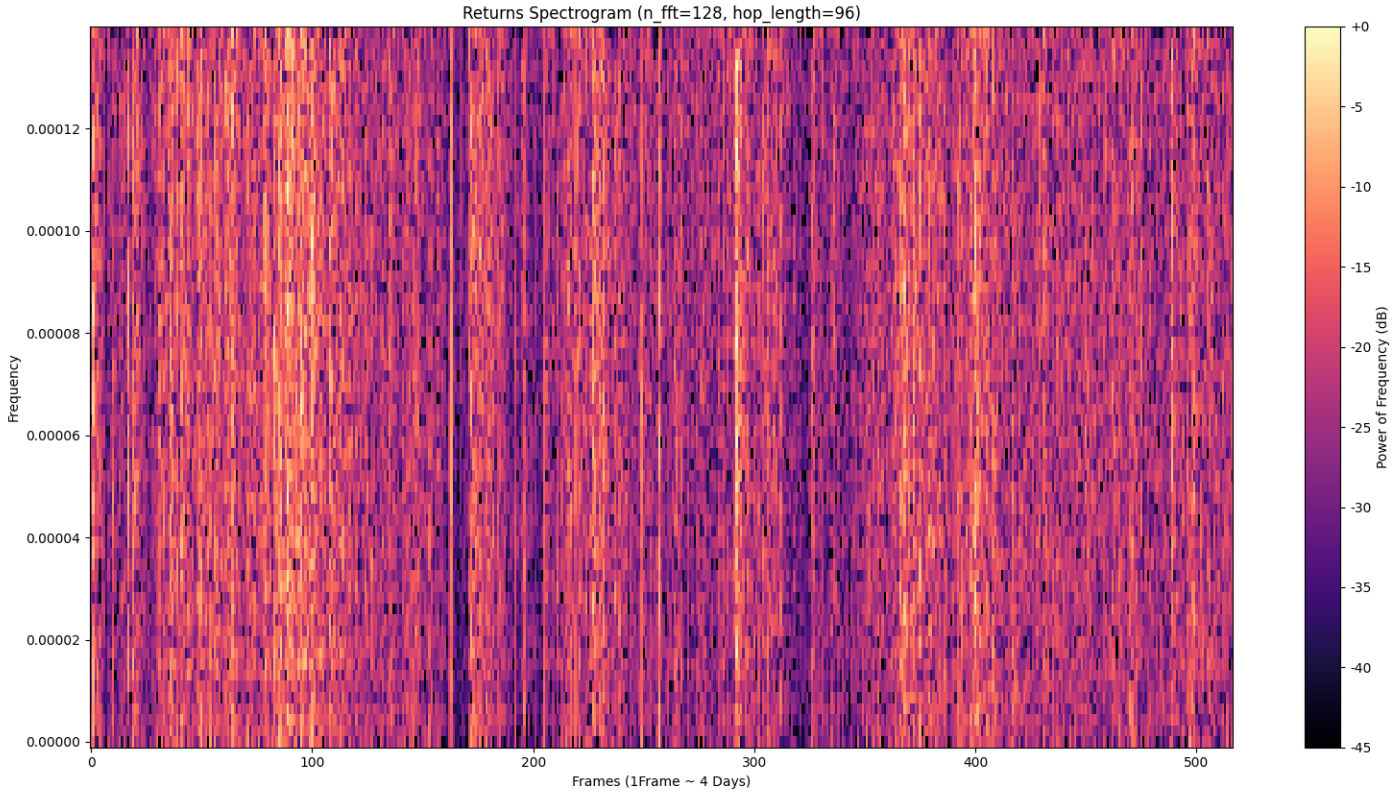


Figure 14 Spectrogram of Returns

Another way to represent the same information is by graphing the power of each frequency over time:

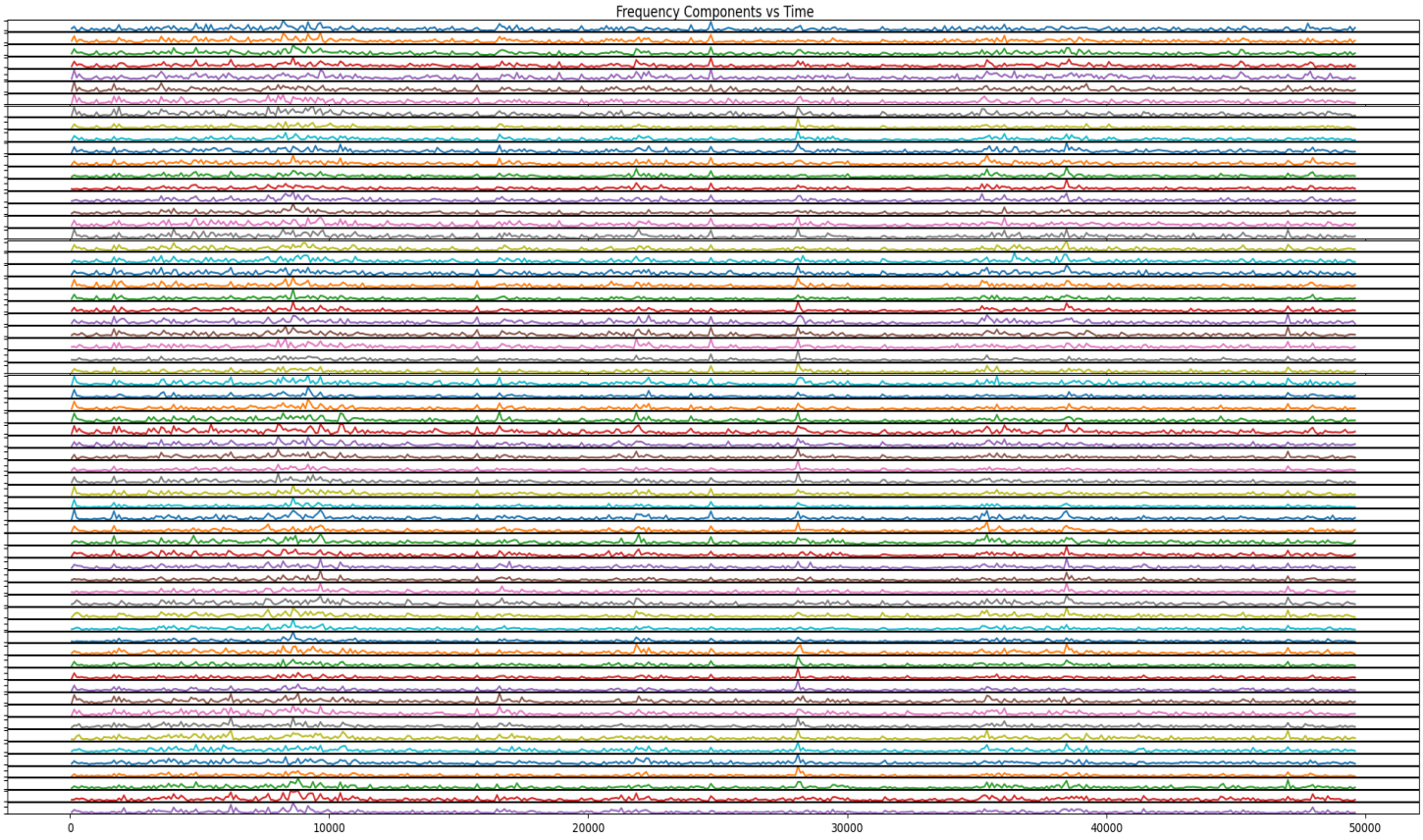


Figure 15 Decomposed Spectrogram of Returns

Once we have the spectrogram, we need to compute various features of interest, which we selected for the model. Specifically, we will be computing Root Mean Square Energy and Spectral Roloff.

RMS Energy is a measure of the spectral energy at a given time. In other words, it is equal to the root mean squared value of the sum of the powers of each frequency, i.e the energy of a single frame. It is defined as $RMS = \sqrt{\frac{1}{n}\sum_i \omega_i^2}$. This feature is included, because high volatility events would naturally imply an increase in energy across many frequencies, and consequently an increase in the RMS energy.

Spectral Roloff is defined as some frequency, such that *some percentage* of energy in a frame is present at or below the frequency. This way, a 50% Spectral Roloff Frequency would be a point such that half the energy is below it, and half above. We look at 3 roloff boundaries – 1%, 50%, 99%.

After a primary run of the model, it was discovered that spectral roloff has virtually no effect, and as such it was removed in favor of a lag1 RMSE in addition to non-lagged RMSE.

The above features are graphically represented in Figure 16 Model Features

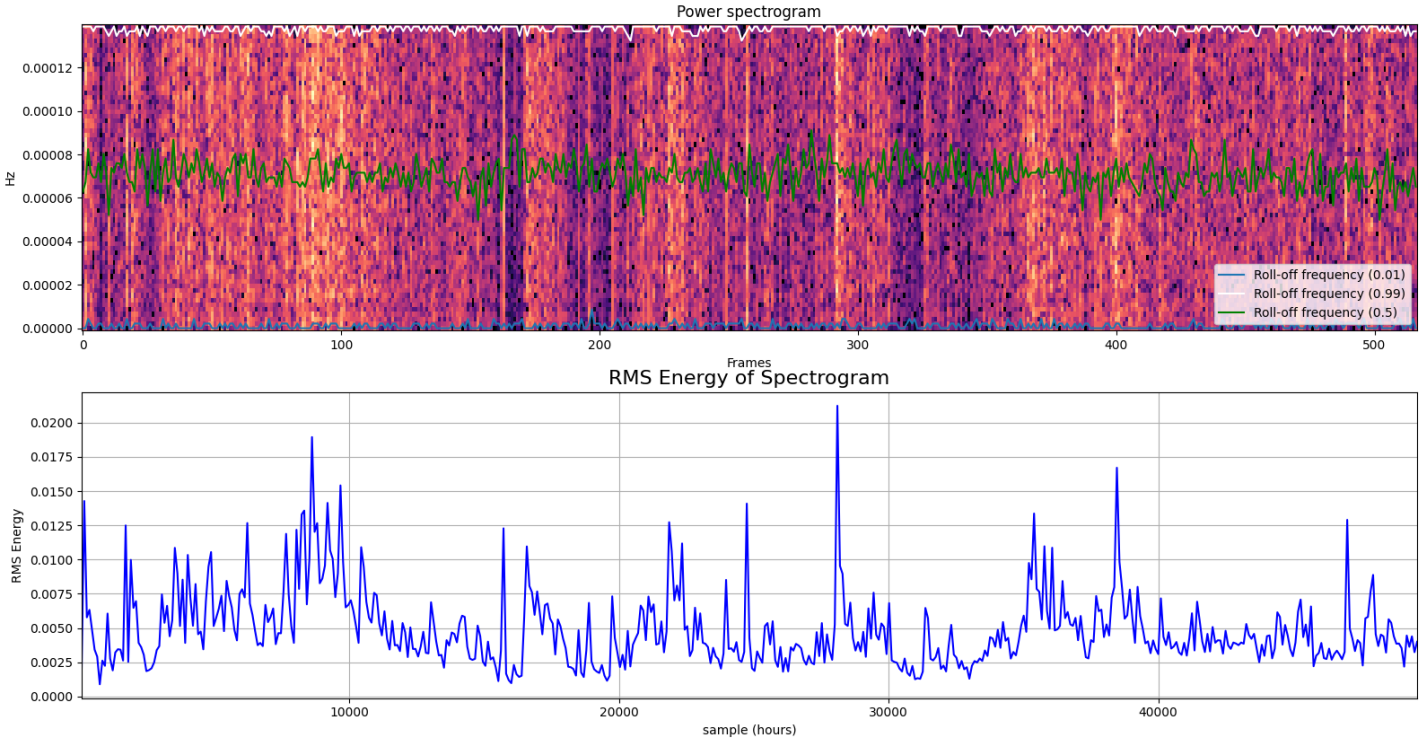


Figure 16 Model Features

At this point, we have all the input data for our model. To proceed, we take our original spectrogram, and shift it forward in time. This will be our target dataframe. We also append our calculated features to the original spectrogram. Now, we have the following:

- A "to predict" dataframe of dimensions (515, 65) – 65 frequencies and 515 frames
- A "factors" dataframe, of dimensions (515, 67) – 65 frequencies, RMSE, 1-lag RMSE, 515 frames

We then define 65 ElasticNet models, each of them trying to predict a single column from the “to predict” dataframe based on the entire “factors” dataframe. To determine the Hyperparameters to be used in the models, we perform cross-validation, and arrive at $\alpha = 0.001$, L1_ratio=0.005. Per-model hyperparameter optimization was attempted, but the results were difficult to interpret, and the author suspects an implementation error. Thus, the same Hyperparameters were used for all models, with the following result:

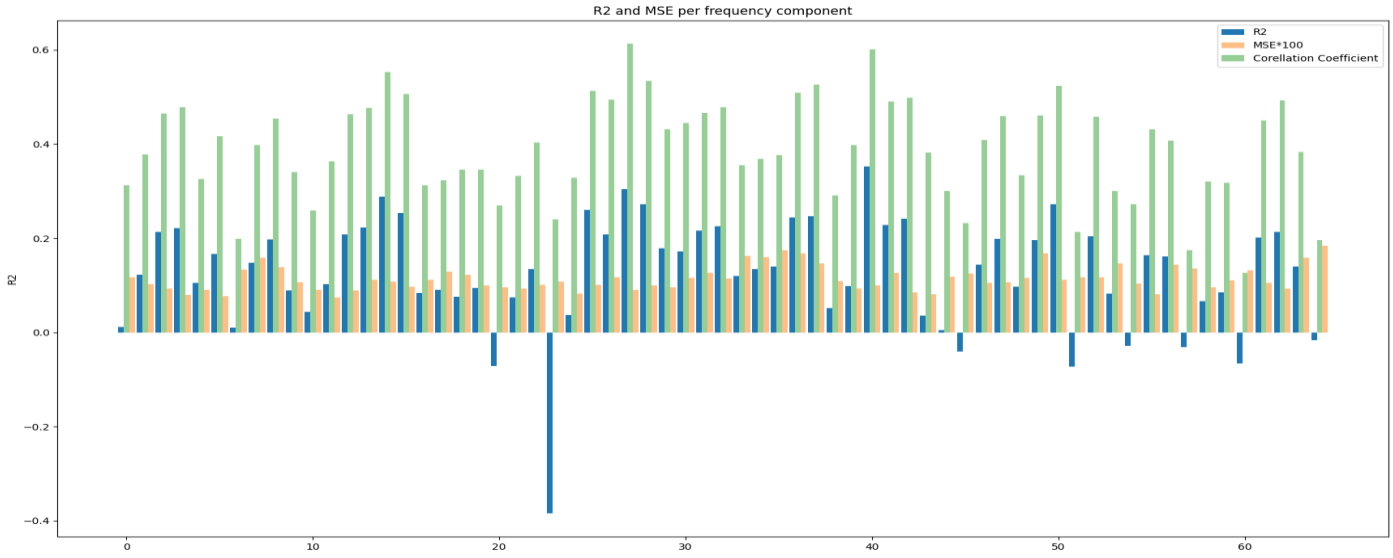


Figure 17 Cross Validation Results

At this point, we are ready to perform an out-of-sample test of the model. To do this, the last 100 frames of data are sliced away from the input spectrogram. The models are trained on this data, and then perform a rolling prediction of STFT frames, with the following result:

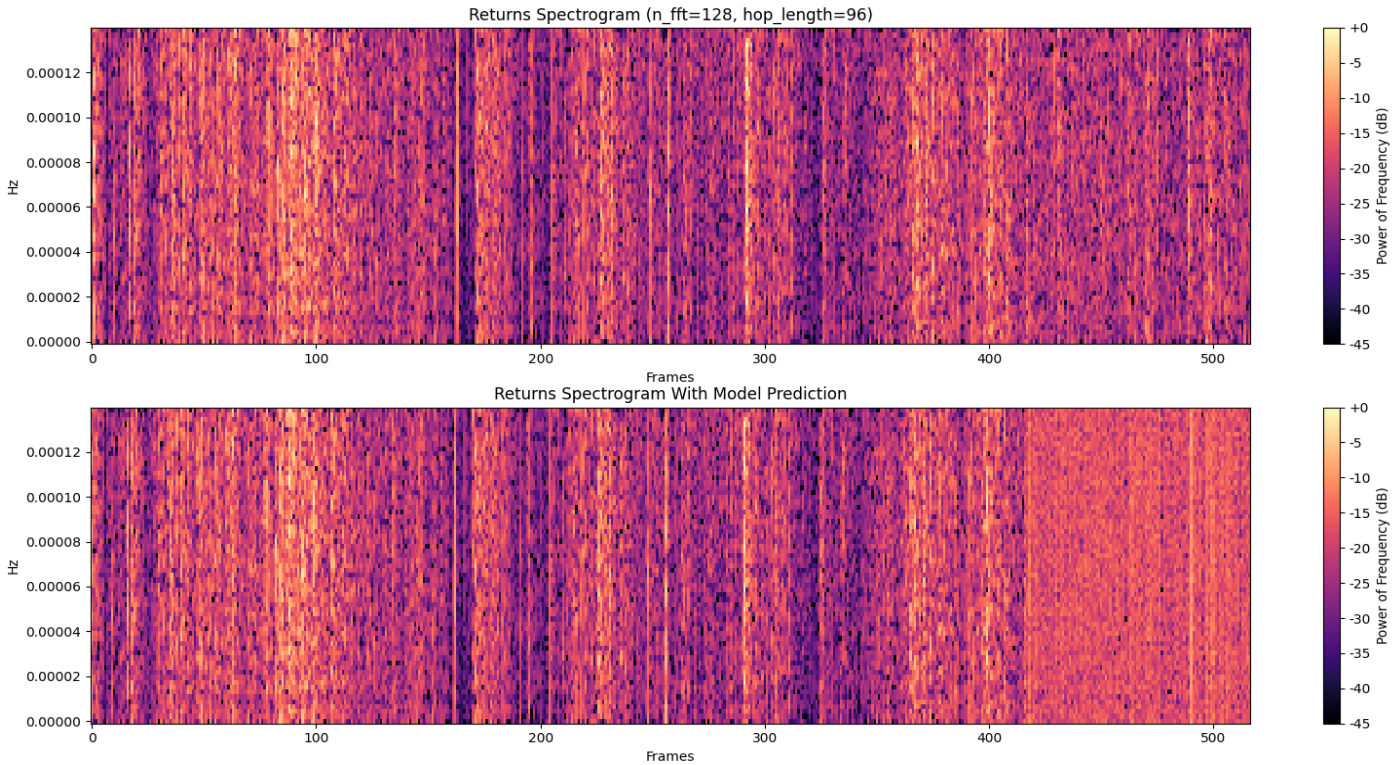


Figure 18 Spectrogram Prediction Results

Upon receiving this result, we go frame-by-frame and compute the inverse STFT to bring our prediction into the time domain:

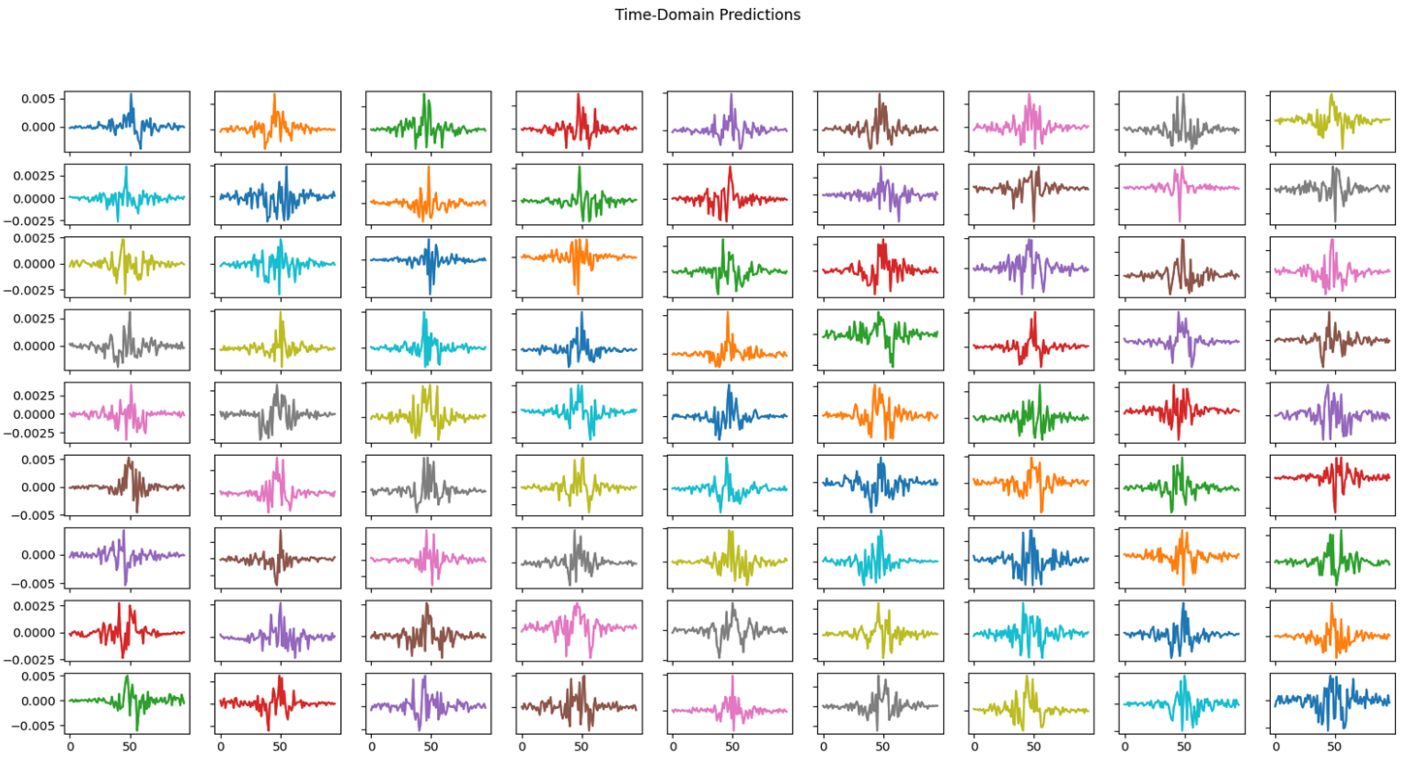


Figure 19 Time Domain Representation of Spectrogram Predictions

Each of the above waves is a prediction of returns movement, although not localized in terms of phase. Because of this lack of phase information, these have almost no value as predictors of returns or price. However, variance doesn't really care about phase, and we're predicting variance. Thus, we calculate the variance of each slice and arrive at our actual prediction result.

Results

Model Results

In an out-of-sample test, the model achieves the following performance:

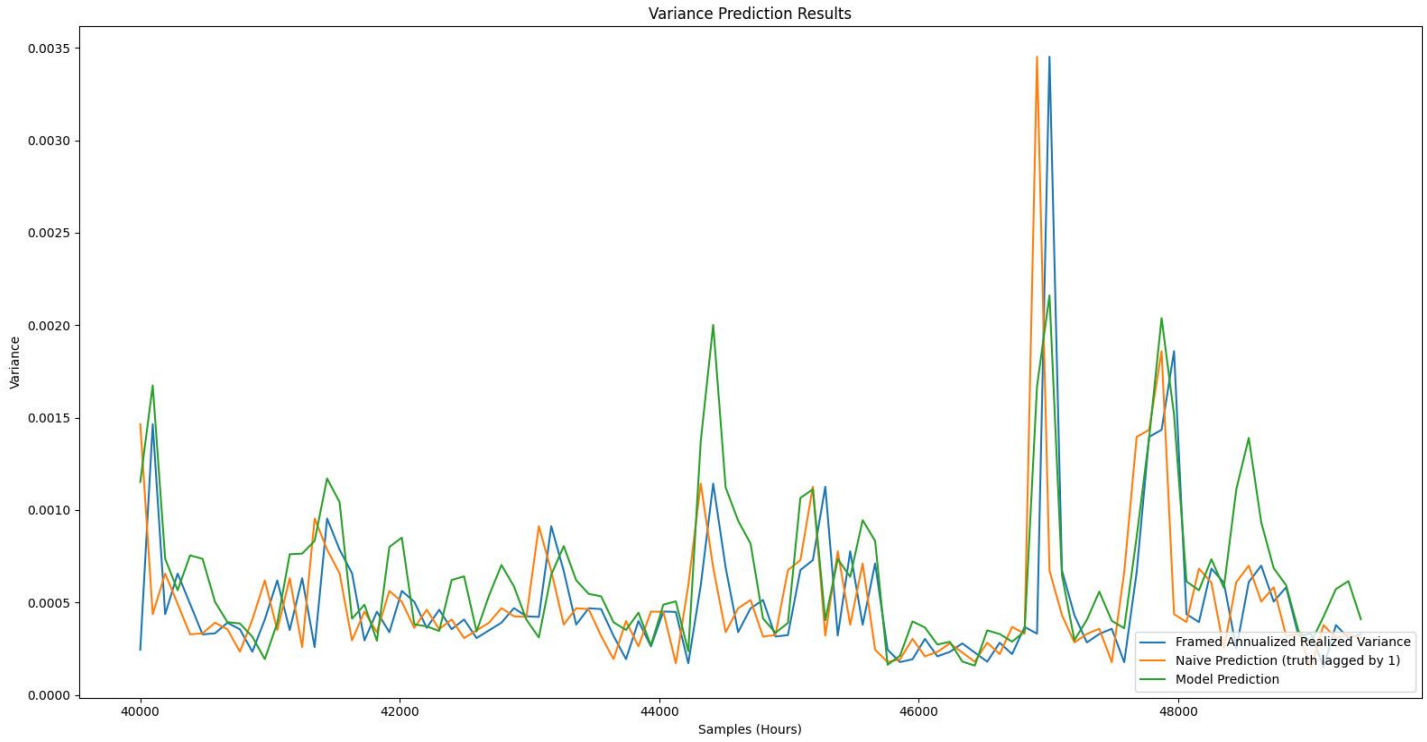


Figure 20 Variance Prediction Result

This test encompasses little over a year of OOS data.

The model's prediction achieves an **R2 Score: 0.2955**

The naïve approach of guessing tomorrow's variance to be equal to today's achieves an **R2 Score: -0.6237**

Trading Strategy Test

To further test whether the model produces useful insights, a simple trading strategy was tested based on its prediction. The trading strategy is as follows:

1. Upon receiving a prediction that the variance over the next 4 days will increase by more than **up_threshold**, purchase a 4DTE at-the-money Straddle on BTC
2. Hold the straddle to expiration

Due to limited availability of option chain data for BTC, a very significant approximation had to be made. In order to properly evaluate strategy profitability – we need to know the entry cost. However, there is no available data on option pricing for the OOS test period. Even estimating the pricing using implied volatility is impossible as no such data is available to the author.

The only data available was the Deribit exchange's option trading tape for 2020. This does not contain quotes or any information on the greeks of the contracts, and includes all possible expirations and levels of moneyness. Thus, as a broad estimate, the author calculated the average ratio of put/call price to underlying BTC price. Using this information, he found that a 4DTE Straddle in 2020 would have cost around 1,000$ for the vast majority of the time.

When running the strategy, the entry cost per trade was assumed to be fixed, and was tested with values of 1000$, 2000$, 3000$, 4000$, 5000$. Since such a huge assumption is being made on cost of the straddle, it is assumed that the imperfection introduced by this is greater than the impact of fees (0.4% for Deribit), so they were ignored.

It must be stressed, that a “fixed entry cost” assumption is extremely detached from reality. Straddle prices depend on volatility, but we completely ignore this phenomenon. Trading strategy simulation results have to be taken with a very large grain of salt.

Since positions are held to expiration, the payoff is modeled simply as the absolute value of the difference between BTC price at position open and position expiry.

With this in mind, the trading strategy achieved the following results:

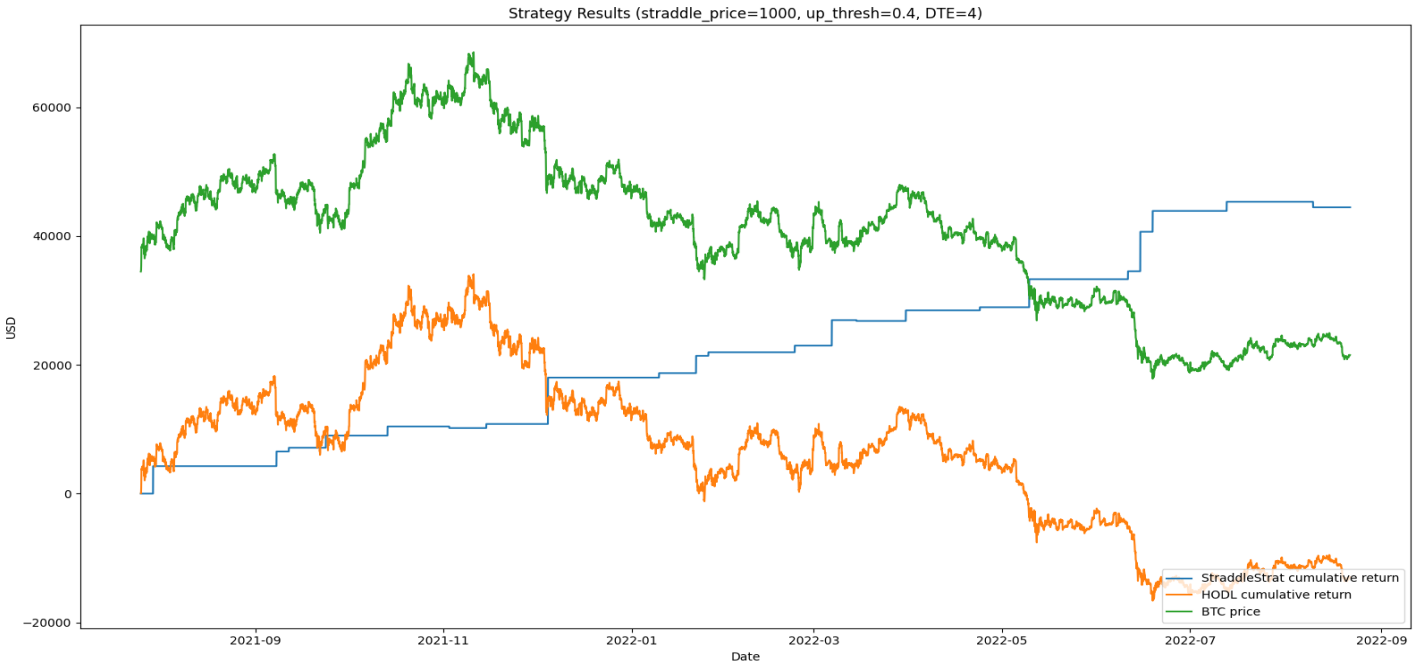


Figure 21 Run 1: Price=1000, up_thresh=0.4, DTE=4

Strategy Return: 44,472$ | HODL Return: -12,957$

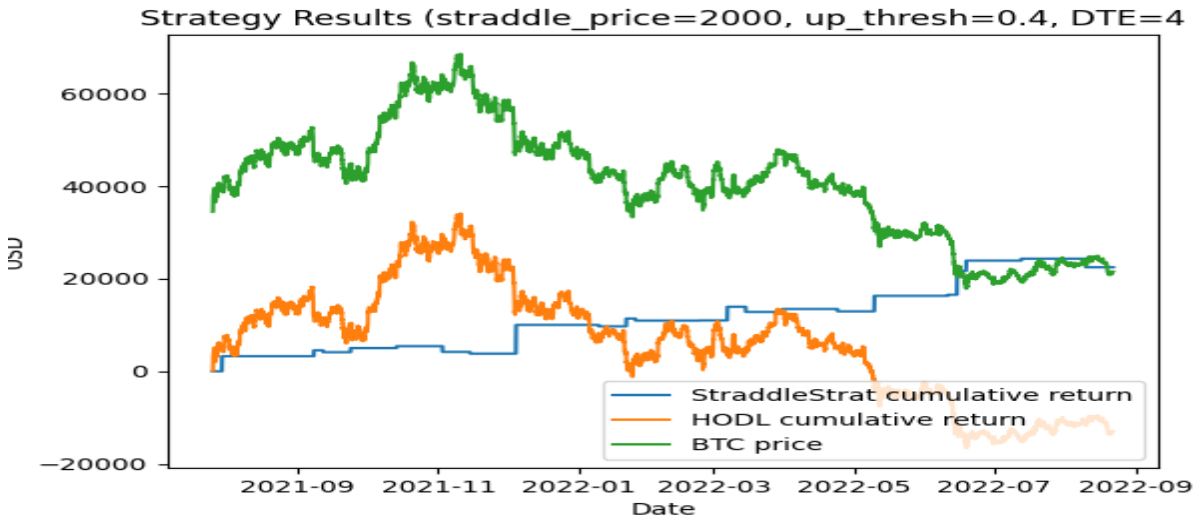


Figure 22 Run 2: Price=2000, up_thresh=0.4, DTE=4

Strategy Return: 22,472$ | HODL Return: -12,957$

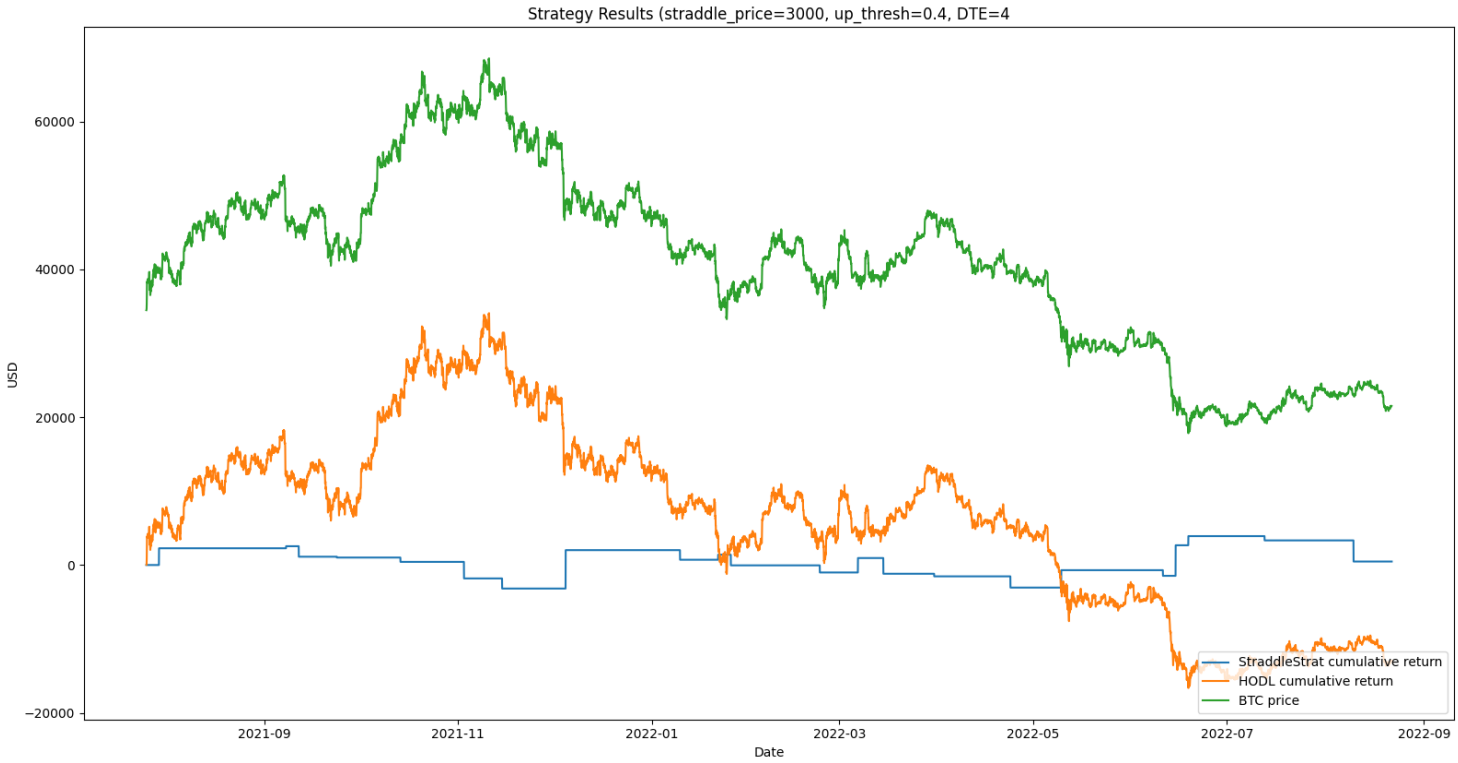


Figure 23 Run 3: Price=3000, up_thresh=0.4, DTE=4

Strategy Return: 472$ | HODL Return: -12,957$

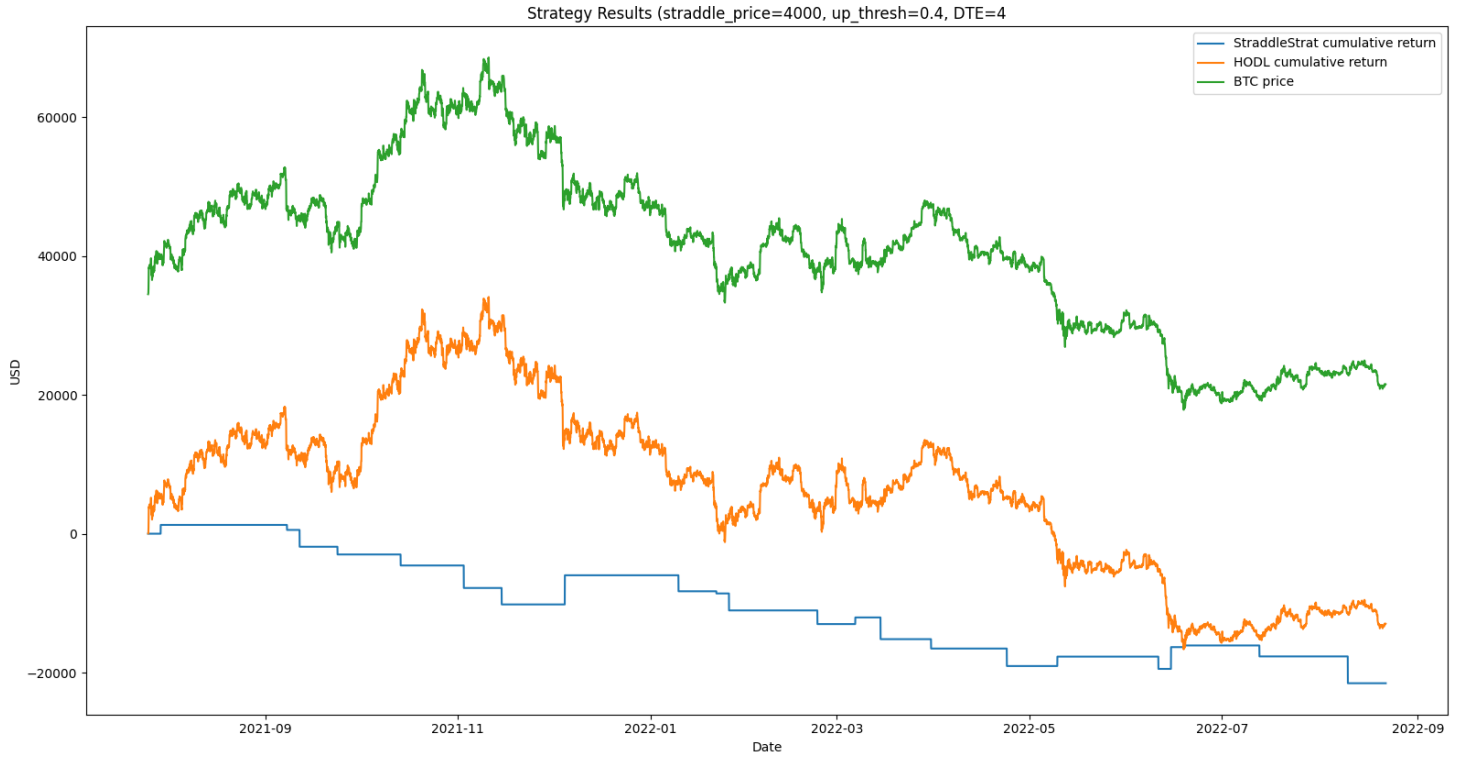


Figure 24 Run 4: Price=4000, up_thresh=0.4, DTE=4

Strategy Return: -21,527$ | HODL Return: -12,957$

As we can see, the strategy returns *drastically* reduce as Straddle prices increase, but can be quite stellar if prices stay low.

Further, this strategy can work even under very expensive options, if we increase the days to expiry and threshold:

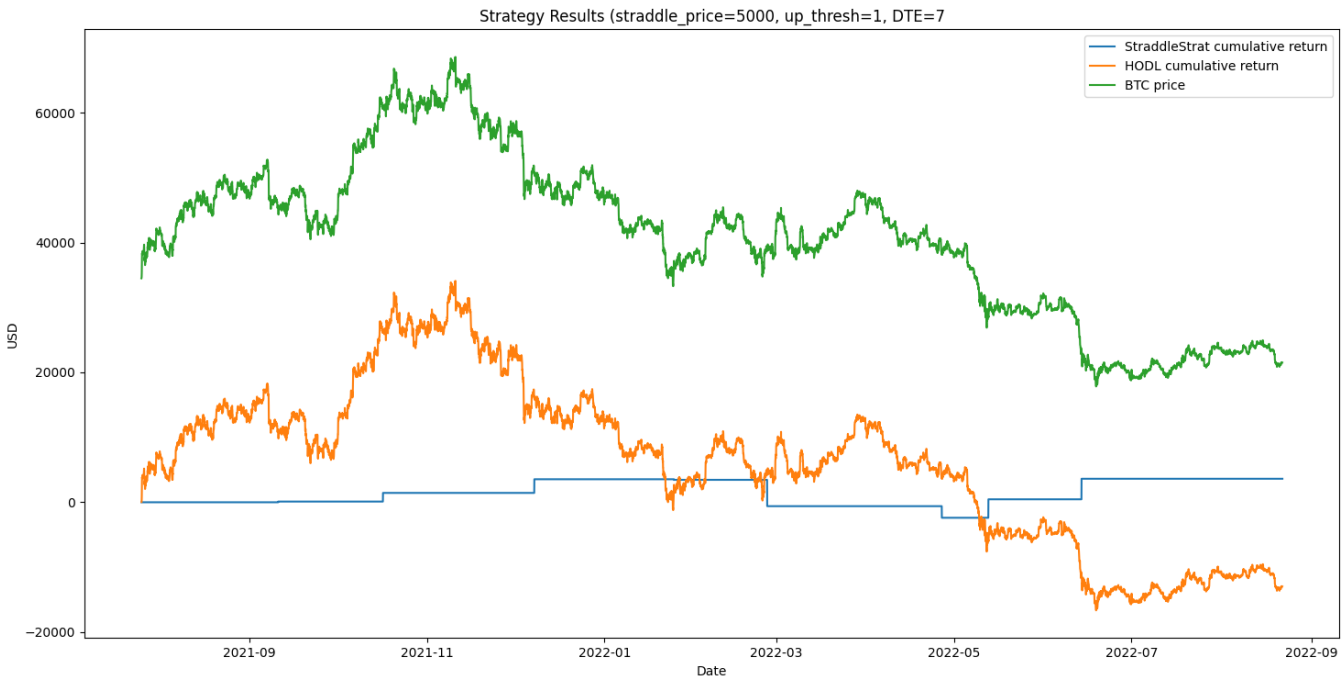


Figure 25 Run 5: Price=5000, up_thresh=1, DTE=7

Strategy Return: 3,620$ | HODL Return: -12,957$

Discussion

Results obtained in this work are surprisingly good. The model achieves strong predictive power in OOS testing, which is further backed by high potential profitability of the proposed trading strategy (with some caveats). It has an R2 of 0.2955, which is very good for this type of data.

Since the model was not trained on the OOS data, there is no danger of overfitting, especially considering that the OOS segment contains information which is significantly different from the training data, as it includes both a steep runup and crash of price. Further, the very purpose of using ElasticNet over a standard OLS is to combat overfitting. Given that we have a cross-validated ElasticNet model with successful OOS performance, it is likely safe to conclude that there is no overfitting problem.

One interesting point about our results can be seen in Figure 18 Spectrogram Prediction Results. We can observe that once our prediction begins, there is much less granularity in the spectrogram – it looks more uniform. This suggests that many of our 65 ElasticNets converged to the same choice of weights. This is not necessarily a problem, as it is possible that only a few select frequencies actually have predictive power, although this is definitely unexpected. This raises a concern that either we have too much frequency resolution, or could benefit from reducing the number of frequencies from 65, or that the majority of frequencies simply don't carry useful predictive information. Due to sklearn's ElasticNet, there is no default representation for the results, and as such the author was unable to view the coefficient selections directly, nor would this be feasible for analysis due to the large number of ElasticNets. As such, the only way to alleviate the aforementioned concern would be to train a new model under a different configuration and benchmark it against existing results.

Another surprising outcome was the result of cross validation, which suggested an alpha of 0.001, and an l1_ratio of 0.005. With these parameters, our ElasticNet is almost a Ridge. Ridge reduces the magnitudes of coefficients, but avoids forcing them to 0. Given the large number of factors in the model, it would have been expected that a large number of them carry redundant or useless information, so if anything, we should have gravitated to Lasso.

Finally, the trading strategy testing achieves shockingly good results. Yes, very daring assumptions were made regarding the pricing of the straddles, but even when this price was assumed to be very high (3,000$ at 4DTE or 5000$ at 7DTE) we still significantly outperformed simply holding Bitcoin. The author expected that any outperformance of the strategy would be minimal, and would have also expected significant drawdowns. Neither of this occurred – the strategy seems to have "printed money" for the majority of the test cases. Again, this must be taken with a grain of salt since the price of real options is nowhere near static.

Overall, the model achieves respectable results, and strongly suggests that there is valuable information present in the spectrum of returns, and that this information could potentially be used to predict their variance.

Future Work

There are several directions of improvement for this model.

First, it would be very interesting to experiment with spectrogram parameters. Current results suggest that we have over-tuned in favor of frequency resolution, when it does not provide that much additional value (given the uniformity of our predictions). The idea here is that by reducing the spectral resolution we will claw-back some time-localization, which will allow us to make shorter-scale predictions (on the order of 1day) and therefore trade lower DTE straddles at cheaper cost.

It would also be promising to try out a pure-lasso model, with a new round of cross validation. Despite current cross-validation results, the author is not convinced that Ridge is more suitable than Lasso given the large number of factors, and is more inclined to believe that an error was made somewhere in the cross-validation process.

Another interesting development would be to shift the entire analysis back in time, training on even older data and predicting a different OOS segment. This would establish whether the model's apparent success is repeatable across other timeframes.

Finally, and most interestingly – the trading strategy needs further revision. Assuming that the model itself is re-verified and found to be without implementation error, the trading strategy should be tested on real-time data. This would allow a smarter strategy definition than "buy straddles for fixed price upon fixed threshold, and hold till expiration". For example, the purchase threshold can be modified to take into account implied volatility, so that the strategy only buys when the projected move exceeds the implied move. Alternatively, we could rewrite the threshold in terms of a breakeven price move, only buying when the projected variance is a certain multiple greater than the minimal breakeven move in the underlying. We could also introduce a downward threshold, and use it to go short on straddles (or long iron condors) when we predict a significant fall in volatility, therefore profiting from volatility crush. Such an updated strategy could be tested "on paper" using real-time data, assuming such data is available.

In addition, several minor things could also benefit from attention:

- Experiment with using multiple spectrograms of different parameters as inputs. For example, a frequency-focused and a time-focused instance
- Attempt per-model cross validation again
- Expand model to other cryptocurrencies, investigate cross-currency return variance dependencies

Appendix

Trading Logs from Strategy Tests

Cost=1000 DTE=4 Thresh=0.4:

| Date 2021-09-03 21:00:00 | BTC price at entry 50151.0825271 | BTC price at exit 46873.0 | Payoff 3278.0825270999994 | entry cost 1000 | profit 2278.0825270999994
| Date 2021-09-07 21:00:00 | BTC price at entry 46873.0 | BTC price at exit 45294.02660435 | Payoff 1578.9733956499986 | entry cost 1000 | profit 578.9733956499986
| Date 2021-09-19 21:00:00 | BTC price at entry 47626.53101293 | BTC price at exit 44739.0 | Payoff 2887.531012929998 | entry cost 1000 | profit 1887.5310129299978
| Date 2021-10-09 21:00:00 | BTC price at entry 54658.0 | BTC price at exit 57073.0 | Payoff 2415.0 | entry cost 1000 | profit 1415.0
| Date 2021-10-29 21:00:00 | BTC price at entry 62374.0 | BTC price at exit 63133.46819531 | Payoff 759.4681953100007 | entry cost 1000 | profit -240.53180468999926
| Date 2021-11-10 21:00:00 | BTC price at entry 65768.0 | BTC price at exit 64136.2185172 | Payoff 1631.7814827999973 | entry cost 1000 | profit 631.7814827999973
| Date 2021-11-30 21:00:00 | BTC price at entry 57551.0 | BTC price at exit 49352.64463109 | Payoff 8198.35536891 | entry cost 1000 | profit 7198.355368910001
| Date 2022-01-05 21:00:00 | BTC price at entry 44161.2795584 | BTC price at exit 42464.0 | Payoff 1697.2795583999978 | entry cost 1000 | profit 697.2795583999978
| Date 2022-01-17 21:00:00 | BTC price at entry 42082.0 | BTC price at exit 38395.62670477 | Payoff 3686.373295229998 | entry cost 1000 | profit 2686.373295229998
| Date 2022-01-21 21:00:00 | BTC price at entry 38395.62670477 | BTC price at exit 36840.0 | Payoff 1555.626704770002 | entry cost 1000 | profit 555.6267047700021
| Date 2022-02-18 21:00:00 | BTC price at entry 40089.0 | BTC price at exit 38053.0 | Payoff 2036.0 | entry cost 1000 | profit 1036.0
| Date 2022-03-02 21:00:00 | BTC price at entry 43815.0 | BTC price at exit 38862.0 | Payoff 4953.0 | entry cost 1000 | profit 3953.0
| Date 2022-03-10 21:00:00 | BTC price at entry 39655.0 | BTC price at exit 38780.0 | Payoff 875.0 | entry cost 1000 | profit -125.0
| Date 2022-03-26 21:00:00 | BTC price at entry 44573.0 | BTC price at exit 47221.2302473 | Payoff 2648.230247300002 | entry cost 1000 | profit 1648.2302473000018
| Date 2022-04-19 21:00:00 | BTC price at entry 41314.0 | BTC price at exit 39834.0 | Payoff 1480.0 | entry cost 1000 | profit 480.0
| Date 2022-05-05 21:00:00 | BTC price at entry 36438.0 | BTC price at exit 31082.0 | Payoff 5356.0 | entry cost 1000 | profit 4356.0
| Date 2022-06-06 21:00:00 | BTC price at entry 31443.99893077 | BTC price at exit 29209.0 | Payoff 2234.9989307700016 | entry cost 1000 | profit 1234.9989307700016
| Date 2022-06-10 21:00:00 | BTC price at entry 29209.0 | BTC price at exit 22075.0 | Payoff 7134.0 | entry cost 1000 | profit 6134.0
| Date 2022-06-14 21:00:00 | BTC price at entry 22075.0 | BTC price at exit 17838.0 | Payoff 4237.0 | entry cost 1000 | profit 3237.0
| Date 2022-07-08 21:00:00 | BTC price at entry 21844.0 | BTC price at exit 19421.0 | Payoff 2423.0 | entry cost 1000 | profit 1423.0
| Date 2022-08-05 21:00:00 | BTC price at entry 22990.0 | BTC price at exit 23131.0 | Payoff 141.0 | entry cost 1000 | profit -859.0

total profit 44472.3660644
HODL profit -12957.26952437

Cost=2000 DTE=4 Thresh=0.4:

| Date 2021-09-03 21:00:00 | BTC price at entry 50151.0825271 | BTC price at exit 46873.0 | Payoff 3278.0825270999994 | entry cost 2000 | profit 1278.0825270999994

| Date 2021-09-07 21:00:00 | BTC price at entry 46873.0 | BTC price at exit 45294.02660435 | Payoff 1578.9733956499986 | entry cost 2000 | profit -421.02660435000143
| Date 2021-09-19 21:00:00 | BTC price at entry 47626.53101293 | BTC price at exit 44739.0 | Payoff 2887.531012929998 | entry cost 2000 | profit 887.5310129299978
| Date 2021-10-09 21:00:00 | BTC price at entry 54658.0 | BTC price at exit 57073.0 | Payoff 2415.0 | entry cost 2000 | profit 415.0
| Date 2021-10-29 21:00:00 | BTC price at entry 62374.0 | BTC price at exit 63133.46819531 | Payoff 759.4681953100007 | entry cost 2000 | profit -1240.5318046899993
| Date 2021-11-10 21:00:00 | BTC price at entry 65768.0 | BTC price at exit 64136.2185172 | Payoff 1631.7814827999973 | entry cost 2000 | profit -368.2185172000027
| Date 2021-11-30 21:00:00 | BTC price at entry 57551.0 | BTC price at exit 49352.64463109 | Payoff 8198.35536891 | entry cost 2000 | profit 6198.355368910001
| Date 2022-01-05 21:00:00 | BTC price at entry 44161.2795584 | BTC price at exit 42464.0 | Payoff 1697.2795583999978 | entry cost 2000 | profit -302.72044160000223
| Date 2022-01-17 21:00:00 | BTC price at entry 42082.0 | BTC price at exit 38395.62670477 | Payoff 3686.373295229998 | entry cost 2000 | profit 1686.373295229998
| Date 2022-01-21 21:00:00 | BTC price at entry 38395.62670477 | BTC price at exit 36840.0 | Payoff 1555.626704770002 | entry cost 2000 | profit -444.3732952299979
| Date 2022-02-18 21:00:00 | BTC price at entry 40089.0 | BTC price at exit 38053.0 | Payoff 2036.0 | entry cost 2000 | profit 36.0
| Date 2022-03-02 21:00:00 | BTC price at entry 43815.0 | BTC price at exit 38862.0 | Payoff 4953.0 | entry cost 2000 | profit 2953.0
| Date 2022-03-10 21:00:00 | BTC price at entry 39655.0 | BTC price at exit 38780.0 | Payoff 875.0 | entry cost 2000 | profit -1125.0
| Date 2022-03-26 21:00:00 | BTC price at entry 44573.0 | BTC price at exit 47221.2302473 | Payoff 2648.230247300002 | entry cost 2000 | profit 648.2302473000018
| Date 2022-04-19 21:00:00 | BTC price at entry 41314.0 | BTC price at exit 39834.0 | Payoff 1480.0 | entry cost 2000 | profit -520.0
| Date 2022-05-05 21:00:00 | BTC price at entry 36438.0 | BTC price at exit 31082.0 | Payoff 5356.0 | entry cost 2000 | profit 3356.0
| Date 2022-06-06 21:00:00 | BTC price at entry 31443.99893077 | BTC price at exit 29209.0 | Payoff 2234.9989307700016 | entry cost 2000 | profit 234.9989307700016
| Date 2022-06-10 21:00:00 | BTC price at entry 29209.0 | BTC price at exit 22075.0 | Payoff 7134.0 | entry cost 2000 | profit 5134.0
| Date 2022-06-14 21:00:00 | BTC price at entry 22075.0 | BTC price at exit 17838.0 | Payoff 4237.0 | entry cost 2000 | profit 2237.0
| Date 2022-07-08 21:00:00 | BTC price at entry 21844.0 | BTC price at exit 19421.0 | Payoff 2423.0 | entry cost 2000 | profit 423.0
| Date 2022-08-05 21:00:00 | BTC price at entry 22990.0 | BTC price at exit 23131.0 | Payoff 141.0 | entry cost 2000 | profit -1859.0

total profit 22472.366064399997
HODL profit -12957.26952437

Cost=3000 DTE=4 Thresh=0.4:
| Date 2021-09-03 21:00:00 | BTC price at entry 50151.0825271 | BTC price at exit 46873.0 | Payoff 3278.0825270999994 | entry cost 3000 | profit 278.0825270999994
| Date 2021-09-07 21:00:00 | BTC price at entry 46873.0 | BTC price at exit 45294.02660435 | Payoff 1578.9733956499986 | entry cost 3000 | profit -1421.0266043500014
| Date 2021-09-19 21:00:00 | BTC price at entry 47626.53101293 | BTC price at exit 44739.0 | Payoff 2887.531012929998 | entry cost 3000 | profit -112.46898707000219
| Date 2021-10-09 21:00:00 | BTC price at entry 54658.0 | BTC price at exit 57073.0 | Payoff 2415.0 | entry cost 3000 | profit -585.0

| Date 2021-10-29 21:00:00 | BTC price at entry 62374.0 | BTC price at exit 63133.46819531 | Payoff 759.4681953100007 | entry cost 3000 | profit -2240.5318046899993
| Date 2021-11-10 21:00:00 | BTC price at entry 65768.0 | BTC price at exit 64136.2185172 | Payoff 1631.7814827999973 | entry cost 3000 | profit -1368.2185172000027
| Date 2021-11-30 21:00:00 | BTC price at entry 57551.0 | BTC price at exit 49352.64463109 | Payoff 8198.35536891 | entry cost 3000 | profit 5198.355368910001
| Date 2022-01-05 21:00:00 | BTC price at entry 44161.2795584 | BTC price at exit 42464.0 | Payoff 1697.2795583999978 | entry cost 3000 | profit -1302.7204416000022
| Date 2022-01-17 21:00:00 | BTC price at entry 42082.0 | BTC price at exit 38395.62670477 | Payoff 3686.373295229998 | entry cost 3000 | profit 686.3732952299979
| Date 2022-01-21 21:00:00 | BTC price at entry 38395.62670477 | BTC price at exit 36840.0 | Payoff 1555.626704770002 | entry cost 3000 | profit -1444.373295229998
| Date 2022-02-18 21:00:00 | BTC price at entry 40089.0 | BTC price at exit 38053.0 | Payoff 2036.0 | entry cost 3000 | profit -964.0
| Date 2022-03-02 21:00:00 | BTC price at entry 43815.0 | BTC price at exit 38862.0 | Payoff 4953.0 | entry cost 3000 | profit 1953.0
| Date 2022-03-10 21:00:00 | BTC price at entry 39655.0 | BTC price at exit 38780.0 | Payoff 875.0 | entry cost 3000 | profit -2125.0
| Date 2022-03-26 21:00:00 | BTC price at entry 44573.0 | BTC price at exit 47221.2302473 | Payoff 2648.230247300002 | entry cost 3000 | profit -351.7697526999982
| Date 2022-04-19 21:00:00 | BTC price at entry 41314.0 | BTC price at exit 39834.0 | Payoff 1480.0 | entry cost 3000 | profit -1520.0
| Date 2022-05-05 21:00:00 | BTC price at entry 36438.0 | BTC price at exit 31082.0 | Payoff 5356.0 | entry cost 3000 | profit 2356.0
| Date 2022-06-06 21:00:00 | BTC price at entry 31443.99893077 | BTC price at exit 29209.0 | Payoff 2234.9989307700016 | entry cost 3000 | profit -765.0010692299984
| Date 2022-06-10 21:00:00 | BTC price at entry 29209.0 | BTC price at exit 22075.0 | Payoff 7134.0 | entry cost 3000 | profit 4134.0
| Date 2022-06-14 21:00:00 | BTC price at entry 22075.0 | BTC price at exit 17838.0 | Payoff 4237.0 | entry cost 3000 | profit 1237.0
| Date 2022-07-08 21:00:00 | BTC price at entry 21844.0 | BTC price at exit 19421.0 | Payoff 2423.0 | entry cost 3000 | profit -577.0
| Date 2022-08-05 21:00:00 | BTC price at entry 22990.0 | BTC price at exit 23131.0 | Payoff 141.0 | entry cost 3000 | profit -2859.0

total profit 472.3660643999974
HODL profit -12957.26952437

Cost=4000 DTE=4 Thresh=0.4:
| Date 2021-09-03 21:00:00 | BTC price at entry 50151.0825271 | BTC price at exit 46873.0 | Payoff 3278.0825270999994 | entry cost 4000 | profit -721.9174729000006
| Date 2021-09-07 21:00:00 | BTC price at entry 46873.0 | BTC price at exit 45294.02660435 | Payoff 1578.9733956499986 | entry cost 4000 | profit -2421.0266043500014
| Date 2021-09-19 21:00:00 | BTC price at entry 47626.53101293 | BTC price at exit 44739.0 | Payoff 2887.531012929998 | entry cost 4000 | profit -1112.4689870700022
| Date 2021-10-09 21:00:00 | BTC price at entry 54658.0 | BTC price at exit 57073.0 | Payoff 2415.0 | entry cost 4000 | profit -1585.0
| Date 2021-10-29 21:00:00 | BTC price at entry 62374.0 | BTC price at exit 63133.46819531 | Payoff 759.4681953100007 | entry cost 4000 | profit -3240.5318046899993
| Date 2021-11-10 21:00:00 | BTC price at entry 65768.0 | BTC price at exit 64136.2185172 | Payoff 1631.7814827999973 | entry cost 4000 | profit -2368.2185172000027
| Date 2021-11-30 21:00:00 | BTC price at entry 57551.0 | BTC price at exit 49352.64463109 | Payoff 8198.35536891 | entry cost 4000 | profit 4198.355368910001

| Date 2022-01-05 21:00:00 | BTC price at entry 44161.2795584 | BTC price at exit 42464.0 | Payoff 1697.2795583999978 | entry cost 4000 | profit -2302.7204416000022
| Date 2022-01-17 21:00:00 | BTC price at entry 42082.0 | BTC price at exit 38395.62670477 | Payoff 3686.373295229998 | entry cost 4000 | profit -313.6267047700021
| Date 2022-01-21 21:00:00 | BTC price at entry 38395.62670477 | BTC price at exit 36840.0 | Payoff 1555.626704770002 | entry cost 4000 | profit -2444.373295229998
| Date 2022-02-18 21:00:00 | BTC price at entry 40089.0 | BTC price at exit 38053.0 | Payoff 2036.0 | entry cost 4000 | profit -1964.0
| Date 2022-03-02 21:00:00 | BTC price at entry 43815.0 | BTC price at exit 38862.0 | Payoff 4953.0 | entry cost 4000 | profit 953.0
| Date 2022-03-10 21:00:00 | BTC price at entry 39655.0 | BTC price at exit 38780.0 | Payoff 875.0 | entry cost 4000 | profit -3125.0
| Date 2022-03-26 21:00:00 | BTC price at entry 44573.0 | BTC price at exit 47221.2302473 | Payoff 2648.230247300002 | entry cost 4000 | profit -1351.7697526999982
| Date 2022-04-19 21:00:00 | BTC price at entry 41314.0 | BTC price at exit 39834.0 | Payoff 1480.0 | entry cost 4000 | profit -2520.0
| Date 2022-05-05 21:00:00 | BTC price at entry 36438.0 | BTC price at exit 31082.0 | Payoff 5356.0 | entry cost 4000 | profit 1356.0
| Date 2022-06-06 21:00:00 | BTC price at entry 31443.99893077 | BTC price at exit 29209.0 | Payoff 2234.9989307700016 | entry cost 4000 | profit -1765.0010692299984
| Date 2022-06-10 21:00:00 | BTC price at entry 29209.0 | BTC price at exit 22075.0 | Payoff 7134.0 | entry cost 4000 | profit 3134.0
| Date 2022-06-14 21:00:00 | BTC price at entry 22075.0 | BTC price at exit 17838.0 | Payoff 4237.0 | entry cost 4000 | profit 237.0
| Date 2022-07-08 21:00:00 | BTC price at entry 21844.0 | BTC price at exit 19421.0 | Payoff 2423.0 | entry cost 4000 | profit -1577.0
| Date 2022-08-05 21:00:00 | BTC price at entry 22990.0 | BTC price at exit 23131.0 | Payoff 141.0 | entry cost 4000 | profit -3859.0

total profit -21527.633935600003
HODL profit -12957.26952437

Cost=4000 DTE=7 Thresh=1:
| Date 2021-10-09 21:00:00 | BTC price at entry 54658.0 | BTC price at exit 60990.0 | Payoff 6332.0 | entry cost 4000 | profit 2332.0
| Date 2021-11-30 21:00:00 | BTC price at entry 57551.0 | BTC price at exit 50447.0 | Payoff 7104.0 | entry cost 4000 | profit 3104.0
| Date 2022-01-17 21:00:00 | BTC price at entry 42082.0 | BTC price at exit 37172.0 | Payoff 4910.0 | entry cost 4000 | profit 910.0
| Date 2022-02-18 21:00:00 | BTC price at entry 40089.0 | BTC price at exit 39144.0 | Payoff 945.0 | entry cost 4000 | profit -3055.0
| Date 2022-04-19 21:00:00 | BTC price at entry 41314.0 | BTC price at exit 38101.0 | Payoff 3213.0 | entry cost 4000 | profit -787.0
| Date 2022-05-05 21:00:00 | BTC price at entry 36438.0 | BTC price at exit 28594.0 | Payoff 7844.0 | entry cost 4000 | profit 3844.0
| Date 2022-06-06 21:00:00 | BTC price at entry 31443.99893077 | BTC price at exit 23279.41543842 | Payoff 8164.583492350001 | entry cost 4000 | profit 4164.583492350001

total profit 11620.66601945
HODL profit -12957.26952437

Cost=5000 DTE=7 Thresh=1:

| Date 2021-10-09 21:00:00 | BTC price at entry 54658.0 | BTC price at exit 60990.0 | Payoff 6332.0 | entry cost 5000 | profit 1332.0
| Date 2021-11-30 21:00:00 | BTC price at entry 57551.0 | BTC price at exit 50447.0 | Payoff 7104.0 | entry cost 5000 | profit 2104.0
| Date 2022-01-17 21:00:00 | BTC price at entry 42082.0 | BTC price at exit 37172.0 | Payoff 4910.0 | entry cost 5000 | profit -90.0
| Date 2022-02-18 21:00:00 | BTC price at entry 40089.0 | BTC price at exit 39144.0 | Payoff 945.0 | entry cost 5000 | profit -4055.0
| Date 2022-04-19 21:00:00 | BTC price at entry 41314.0 | BTC price at exit 38101.0 | Payoff 3213.0 | entry cost 5000 | profit -1787.0
| Date 2022-05-05 21:00:00 | BTC price at entry 36438.0 | BTC price at exit 28594.0 | Payoff 7844.0 | entry cost 5000 | profit 2844.0
| Date 2022-06-06 21:00:00 | BTC price at entry 31443.99893077 | BTC price at exit 23279.41543842 | Payoff 8164.583492350001 | entry cost 5000 | profit 3164.5834923500006

total profit 3620.66601945
HODL profit -12957.26952437

Code

```
import numpy as np
import pandas as pd
from sklearn.linear_model import ElasticNet,ElasticNetCV
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from statistics import variance
import math
import librosa
import librosa.display
from sklearn.model_selection import train_test_split, cross_val_score

#----------PREP DATA------------
# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)

# load data & fix index
df = pd.read_csv("btcusd.csv", index_col='time')
df.index = pd.to_datetime(df.index, unit='ms')
df = df[~df.index.duplicated(keep='first')]
df=df[~(df.index < '2017-01-01')]
df = df.resample('1H').pad()
btcusd=df

#add returns
btcusd['returns']=btcusd['close'].pct_change()
btcusd=btcusd.dropna(axis=0)

#----------CALCULATE WINDOWED VARIANCE------------
PREDICTION_WINDOW=4
sc=500
ret_list = btcusd['returns'].tolist()
prc_list=btcusd['close'].tolist()
var_list=[]
for i in range(len(ret_list)):
    try:
        if i>=PREDICTION_WINDOW:
            var_list.append(variance(ret_list[i-PREDICTION_WINDOW: i]))
```

```
        else:
            var_list.append(np.nan)
    except Exception as e:
        print("EXCEPTION OCCURED:")
        print(e)
btcusd['ret_var']= var_list
btcusd=btcusd.dropna(axis=0)

#ANNUALIZE VARIANCE:
factor = math.sqrt(365/PREDICTION_WINDOW)
btcusd['ret_var_annualized']=btcusd['ret_var']*factor

def var_ret_var_plot():
    """
    Optional function to call for visualization of
    :return:
    """
    fig, ax = plt.subplots(3, 1, figsize=(18, 5))
    ax[0].plot(btcusd['returns'], color='blue', label='returns')
    ax[0].set_ylabel('returns', fontsize=20, color='blue')
    ax[0].set_xlabel('date', fontsize=20)
    ax[0].set_title('BTCUSD Return', fontsize=23)
    ax[0].set_xlim([btcusd.index[0], btcusd.index[-1]])
    ax[0].grid()

    ax[1].plot(btcusd['ret_var'], color='blue', label='var')
    ax[1].set_ylabel('variance', fontsize=16, color='blue')
    ax[1].set_xlabel('date', fontsize=16)
    ax[1].set_title('Returns Variance', fontsize=16)
    ax[1].set_xlim([btcusd.index[0], btcusd.index[-1]])
    ax[1].grid()

    ax[2].plot(btcusd['ret_var_annualized'], color='blue', label='var')
    ax[2].set_ylabel('price [USD]', fontsize=16, color='blue')
    ax[2].set_xlabel('date', fontsize=16)
    ax[2].set_title('Return Variance Annualized', fontsize=16)
    ax[2].set_xlim([btcusd.index[0], btcusd.index[-1]])
    ax[2].grid()

    fig.tight_layout()
    plt.show()

#----------MAKE SPECTROGRAM------------
spec=np.abs(librosa.stft(btcusd['returns'].iloc[:-128].to_numpy(), n_fft=128,
hop_length=96))


#----------CREATE DATAFRAME FOR REGRESSION------------
#transpose spectrogram
data = pd.DataFrame(spec).transpose()

#convert index to samples instead of frames
idx = data.index.to_numpy() #extract index
idx_samples=librosa.frames_to_samples(idx, hop_length=96, n_fft=128) #convert from
"Frame" notation to "Sample"
data.index=idx_samples #reassign index

#CALCULATE FEATURES
rms = librosa.feature.rms(S=spec, hop_length=96, frame_length=128 ) #RMS Energy
features = pd.DataFrame(rms).transpose()
features= features.rename(columns={0: 'rms'})
features.index=idx_samples #reassign index
```

```
roloff_01 = librosa.feature.spectral_rolloff(S=spec, hop_length=96, n_fft=128,
roll_percent=0.01, sr=1/3600).transpose()
roloff_99 = librosa.feature.spectral_rolloff(S=spec, hop_length=96, n_fft=128,
roll_percent=0.99, sr=1/3600).transpose()
roloff_50 = librosa.feature.spectral_rolloff(S=spec, hop_length=96, n_fft=128,
roll_percent=0.5, sr=1/3600).transpose()

features['roloff_01']=roloff_01
features['roloff_99']=roloff_99
features['roloff_50']=roloff_50

def showspecs():
    """
    Plot spectrogram and derived features
    :return:
    """
    fig1, ax1 = plt.subplots()
    img1 = librosa.display.specshow(librosa.amplitude_to_db(spec,
                                                            ref=np.max),
                                    y_axis='linear', x_axis='frames', ax=ax1,
                                    hop_length=96, n_fft=128, sr=1/3600, vmin=-45)
    ax1.set_title('Returns Spectrogram (n_fft=128, hop_length=96)')
    fig1.colorbar(img1, ax=ax1, format="%+2.0f")
    # PLOT SPEC:
    fig, ax = plt.subplots(2)
    librosa.display.specshow(librosa.amplitude_to_db(spec,
                                                     ref=np.max),
                             y_axis='linear', x_axis='frames', ax=ax[0],
                             hop_length=96, n_fft=128,sr=1/3600, vmin=-45)
    ax[0].set_title('Power spectrogram')
    # fig.colorbar(img, ax=ax, format="%+2.0f dB")
    ax[0].plot(idx, roloff_01, label='Roll-off frequency (0.01)')
    ax[0].plot(idx, roloff_99, color='w', label='Roll-off frequency (0.99)')
    ax[0].plot(idx, roloff_50, color='g', label='Roll-off frequency (0.5)')
    ax[0].legend(loc='lower right')
    ax[1].plot(features['rms'], color='blue', label='var')
    ax[1].set_ylabel('RMS Energy')
    ax[1].set_xlabel('sample (hours)')
    ax[1].set_title('RMS Energy of Spectrogram', fontsize=16)
    ax[1].set_xlim([features.index[0], features.index[-1]])
    ax[1].grid()

    data.plot(subplots=True, sharex=True, use_index=True, stacked=True, legend=False,
              title="Frequency Components vs Time", sharey=True)

#----------MAKE MODEL------------
#create a lead-target dataframe
lead=data.shift(1, fill_value=0)

framed_variance=[]
framed_annualized_variance=[]
j=0
for i in idx_samples:
    try:
        target=ret_list[j: i]
        var = variance(target)
        framed_variance.append(var)
        framed_annualized_variance.append(var*factor)
        j=i
    except Exception as e:
        print("EXCEPTION OCCURED:")
        print(e)
j=0
```

```
prc_framed_variance=[]
prc_framed_annualized_variance=[]
for i in idx_samples:
    try:
        target=prc_list[j: i]
        var = variance(target)
        prc_framed_variance.append(var)
        prc_framed_annualized_variance.append(var*factor)
        j=i
    except Exception as e:
        print("EXCEPTION OCCURED:")
        print(e)
variance_df = pd.DataFrame(framed_annualized_variance)
variance_df= variance_df.rename(columns={0: 'framed_annualized_variance'})
variance_df.index=idx_samples #reassign index
variance_df['prc_framed_annualized_variance']=prc_framed_annualized_variance


#map to X y
X=pd.merge(data, features['rms'], left_index=True, right_index=True)
#X=sm.add_constant(X)
X['rms_lag1']=X['rms'].shift(-1)
y=lead
#y=variance_df.shift(1)
#model = sm.OLS.fit_regularized( method='elastic_net', alpha=1, L1_wt=0.5).fit()

X=X.iloc[1:-1 , :]
y=y.iloc[1:-1 , :]
model_list=[]


def crossval():
    """
    module used for cross validation
    Currently non-functional due to further experimentation
    :return:
    """
    alphas = [0.0001, 0.001, 0.01, 0.1, 0.3, 0.5, 0.7, 1]
    l1s = [0, 0.02, 0.04, 0.06, 0.08, 0.10, 0.12, 0.14,0.16]
    for a in alphas:
        #for j in l1s:
        model = ElasticNet(alpha=a, max_iter=5000).fit(X, y)
        score = model.score(X, y)
        pred_y = model.predict(X)
        mse = mean_squared_error(y, pred_y)
        print("Alpha:{0:.4f}, R2:{1:.2f}, MSE:{2:.2f}, RMSE:{3:.2f}"
              .format(a, score, mse, np.sqrt(mse)))


#----------TRAIN MODEL & MAKE FREQUENCY DOMAIN PREDICTION------------

xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.15)

scores=[]
mses=[]
correls=[]
models=[]
predict_stft = pd.DataFrame(columns=range(100))
predict_inverse=pd.DataFrame(columns=range(96))
#for j in range(100):
XX = X.iloc[0:-100, :]
yy = y.iloc[0:-100, :]
xtest = X.iloc[-100:, :]
```

```
predict=[]
for i in y:
    print("REGRESSION FOR FREQUENCY COMPONENT: ",i)
    elastic = ElasticNet(alpha=0.001, l1_ratio=0.005, normalize=True,
fit_intercept=False).fit(XX, yy[i])
    models.append(elastic)
    ypred = elastic.predict(xtest)
    predict_stft=predict_stft.append(pd.DataFrame(ypred).transpose())

#----------CALCULATE IFFT OF PREDICTION (GO TO TIME DOMAIN)------------
for i in predict_stft:
    try:
        inv=librosa.istft(predict_stft[[i, i + 1]].to_numpy(), n_fft=128,
hop_length=96)
        predict_inverse=predict_inverse.append(pd.DataFrame(inv).transpose())
    except KeyError:
        pass
predict_inverse=predict_inverse.reset_index(drop=True)
predict_inverse=predict_inverse.transpose() #ROWS=FRAMES, COLUMNS=SAMPLES IN FRAME

#----------CALCULATE PREDICTED VARIANCE------------
pred_var=[]
variance_df=variance_df.iloc[1:-1 , :]
test_variance = variance_df.iloc[-100:-1, :]
for i in predict_inverse:
    pred_var.append(variance(predict_inverse[i])*sc)

test_variance['predicted_var']=pred_var


#----------EVALUATE VARIANCE PREDICTION------------
#CREATE NAIVE STRATEGY
test_variance['naive_var']=test_variance['framed_annualized_variance'].shift(-1,
fill_value=test_variance['framed_annualized_variance'].to_list()[-2]).to_numpy()

#PLOT RESULT
fig, ax= plt.subplots(1,1)
ind = test_variance.index
ax.plot(ind, test_variance['framed_annualized_variance'], label='Framed Annualized
Realized Variance')
ax.plot(ind, test_variance['naive_var'],  label='Naive Prediction (truth lagged by
1)')
ax.plot(ind, test_variance['predicted_var'],  label='Model Prediction')
ax.legend(loc='lower right')

#CALCULATE R2
from sklearn.metrics import r2_score
r2_naive = r2_score(test_variance['framed_annualized_variance'],
test_variance['naive_var'])
r2_model = r2_score(test_variance['framed_annualized_variance'],
test_variance['predicted_var'])
print("naive R2:", r2_naive, "model R2:", r2_model)

plt.show()



#----------TEST A TRADING STRATEGY------------

#PARAMETERS
nr_days=99*4
straddle_price=3000 #assumed price of straddle
```

```
up_thresh = 0.4 #predicted volatility increase % threshold
DTE=4 #days to expiry of straddle



#compute signal indicator
test_variance['pct_change_predicted']=test_variance['predicted_var'].pct_change()


#create decision making signal
test_variance['signal']=np.where(test_variance['pct_change_predicted'].shift(-
1)>=up_thresh, 1, 0)

#create evaluation data (sync price to variance prediction)
test = btcusd[40000:49408]
indx = [test.index[0]]
for i in range(99):
    indx.append(indx[0]+pd.DateOffset(days=(i+1)*4))
test_variance.index=indx[0:99]

test = test.merge(test_variance, left_index=True, right_index=True, how='outer')
test['strat_profit']=0
prof_list=[]

#run the strategy
for ind in test.index:
    stat=test['signal'][ind]
    if stat==1:
        price_at_entry=test['close'][ind]
        price_at_exit=test['close'][ind+pd.DateOffset(days=DTE)] #exit
        entry=straddle_price
        payoff = abs(price_at_exit-price_at_entry)
        profit=payoff-entry
        print("| Date", ind, "| BTC price at entry", price_at_entry, "| BTC price at
exit", price_at_exit, "| Payoff", payoff, "| entry cost", entry, "| profit", profit )
        prof_list.append(profit)
        test['strat_profit'][ind+pd.DateOffset(days=DTE)] = profit
print('total profit', sum(prof_list))
print("HODL profit", test['close'].iloc[-2]-test['close'].iloc[0])

#graphically display results
test['strat_cumulative_profit']=test['strat_profit'].cumsum()
test['hodl_daily_profit']=test['close']-test['close'].shift(1)
test['hodl_cumulative_profit']=test['hodl_daily_profit'].cumsum()
fig, ax= plt.subplots(1,1)
ind = test.index
ax.plot(ind, test['strat_cumulative_profit'], label='StraddleStrat cumulative
return')
ax.plot(ind, test['hodl_cumulative_profit'],  label='HODL cumulative return')
ax.plot(ind, test['close'],  label='BTC price')
ax.legend(loc='upper right')
plt.title("Strategy Results (straddle_price="+str(straddle_price)+ ",
up_thresh="+str(up_thresh)+ ", DTE="+str(DTE))
plt.xlabel("Date")
plt.ylabel("USD")
plt.show()
```