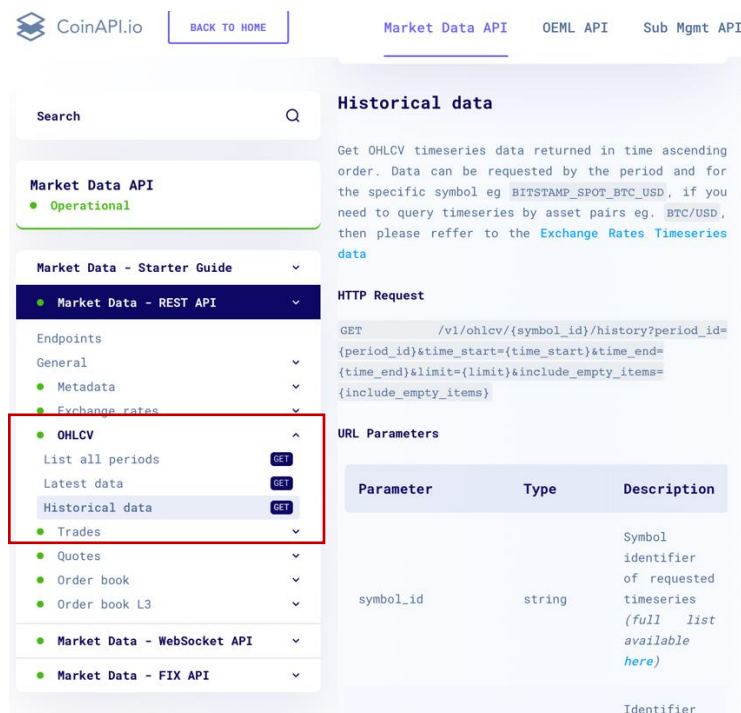


## INTERVIEW ASSESSMENT

Build a Monte Carlo simulation to predict Bitcoin prices for the next 7 days using 100 simulation iterations. There are 8 tasks to be completed. Scripting to be accomplished in Python. Submit your code in Jupyter Notebook. Time Limit: 72hrs from email receipt.

### 1. Retrieve Historical Bitcoin Prices via API

Use the free API available at <https://www.coinapi.io> to retrieve historical Bitcoin prices starting 01-01-2021. Use OHLCV API link as provided in documentation with a periodicity of 1 day.



The screenshot shows the CoinAPI.io website. On the left, a sidebar menu lists various API endpoints under 'Market Data - REST API'. The 'OHLCV' endpoint is highlighted with a red box. On the right, the 'Historical data' section provides a description of the API, an example HTTP request, and a table of URL parameters.

**Historical data**

Get OHLCV timeseries data returned in time ascending order. Data can be requested by the period and for the specific symbol eg BITSTAMP\_SPOT\_BTC\_USD, if you need to query timeseries by asset pairs eg. BTC/USD, then please refer to the [Exchange Rates Timeseries data](#)

**HTTP Request**

```
GET /v1/ohlcv/{symbol_id}/history?period_id={period_id}&time_start={time_start}&time_end={time_end}&limit={limit}&include_empty_items={include_empty_items}
```

**URL Parameters**

Parameter	Type	Description
symbol_id	string	Symbol identifier of requested timeseries (full list available <a href="#">here</a> )
period_id	string	Identifier

### 2. Calculate Logarithmic Returns using the formula as below using close prices

Log Returns =  $\log(1 + [\text{Price}(t) - \text{Price}(t-1)] / \text{Price}(t-1))$ , t refers to time

### 3. Calculate the following parameters using the Log Returns

- Mean
- Variance
- Drift = Mean -  $\frac{1}{2} \times \text{Variance}$
- Standard Deviation

### 4. Create a function - monte\_carlo\_simulation() and use *for loops* to simulate price paths

- Use inverse of cdf (i.e. ppf) to create random values from a normal distribution for all days in each simulations and store it in Z

- b. Calculate daily returns =  $e^{(\text{drift} + \text{standard deviation} * Z)}$
- c.  $\text{Price}(t) = \text{Price}(t-1) * \text{daily return}(t)$ , t refers to time

Part of the skeleton for this function is provided as below:

This is just a skeleton function provided to assist you in developing the algorithm. It has missing code snippets which will be needed to be updated by you or you can attempt to write an entirely new function with your own algorithm.

```
def monte_carlo_simulation(periods , sims, latest_hist_close):

    #####
    for s in range (0, sims):
        sim_paths = [0]*sims

        Z = #####
        daily_returns = #####
        price_paths = #####
        price_paths[0] = latest_hist_close

        for t in range(1, periods):
            price_paths[t] = #####

        #####

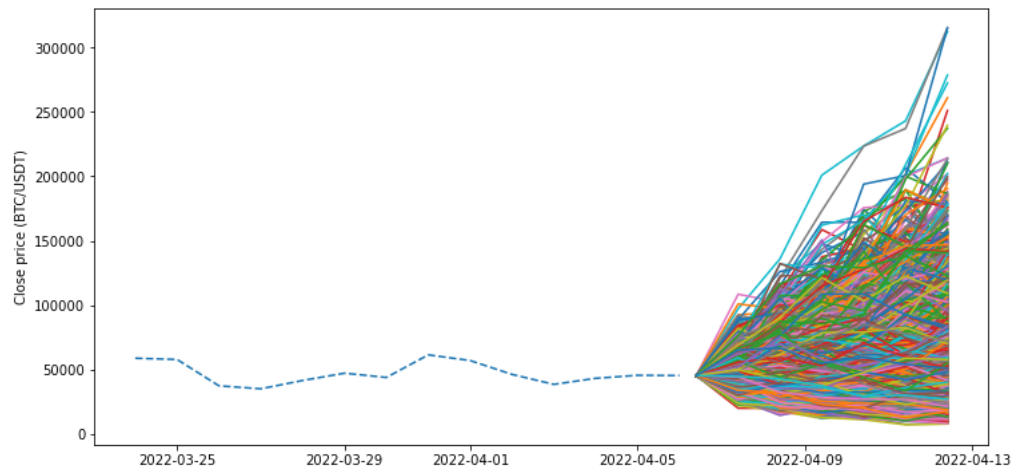
    return sim_paths
```

\*Use timeit to show the execution time of this function

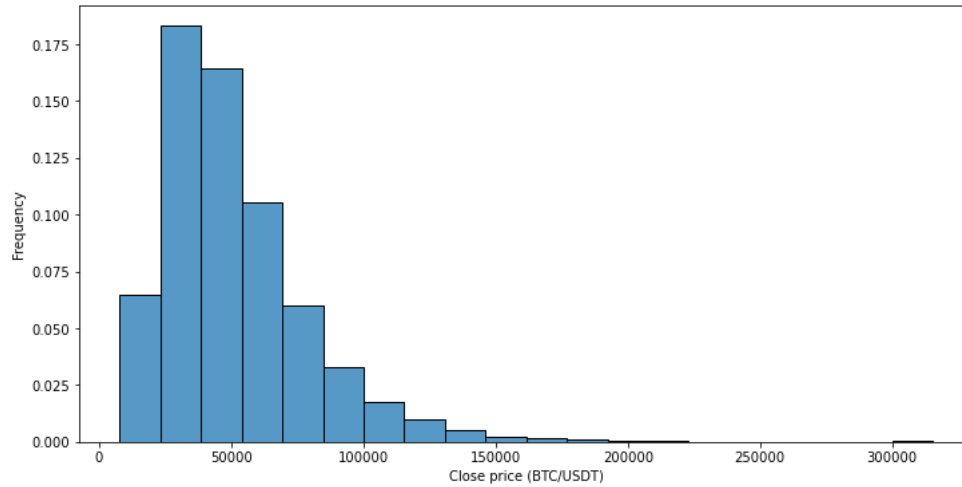
5. Build similar function as above but vectorize the function using numpy

\*Use timeit to show the execution time of this function

6. Plot the simulated paths with historical as shown below



7. Plot the distribution of the simulated paths as shown below:



8. Can you use numba package to build the same monte carlo simulation function? Is there any limitation you found to use numba?