Take Home assignment - Crypto.com

Hugo MOREL

October 14, 2023

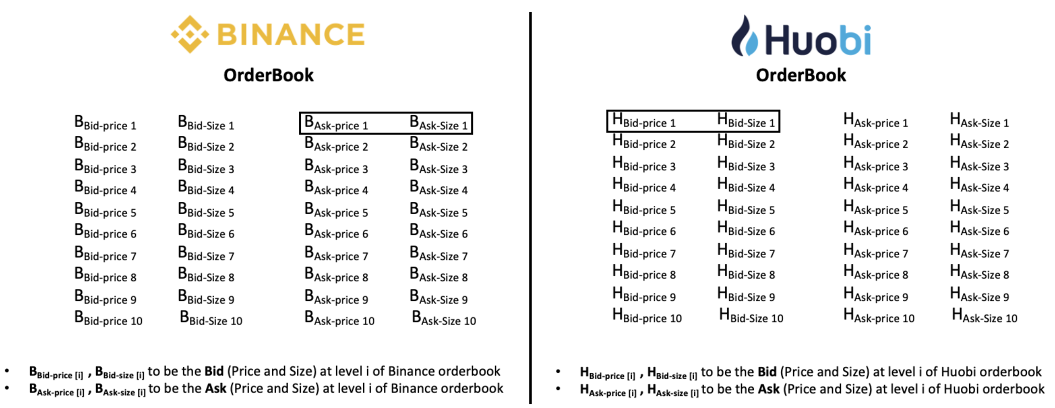
1. Design an arbitrage strategy on LINK/USDT between Binance and Huobi based on provided data.
2. Code a backtesting program by yourself with Python to test your strategy. Open source backtesting framework is not allowed.
3. Present your backtest result with risk metrics and charts.
4. Submit your risk metrics, charts, trade logs with backtesting Python script.
5. Post trade analysis will be a plus.

1 Design an arbitrage strategy on LINK/USDT between Binance and Huobi based on provided data

1.1 Context

Strategy : Based on Binance and Huobi orderbooks, our strategy will be to exploit price deffentials between tow platforms, by executing simultaneous buy and sell operations when arbitrage opportunities emerge.

How the code works : I developed a recursive function that delves into order books, analyzing and comparing various depth levels [i] to maximize the arbitrage quantity when an opportunity occurs. The below graph shows an example of transaction between platforms where we buy on Binance and Sell on Huobi (the opposite transaction may accurs as well).



$$B_{\text{Ask-price [i]}} < H_{\text{Bid-price [i]}}$$

Order 1 :

- Side : **Buy**
- Size : **SizeOrder *(*)***
- Price : $\mathbf{B}_{\textbf{Ask-price [i]}}$

Order 2 :

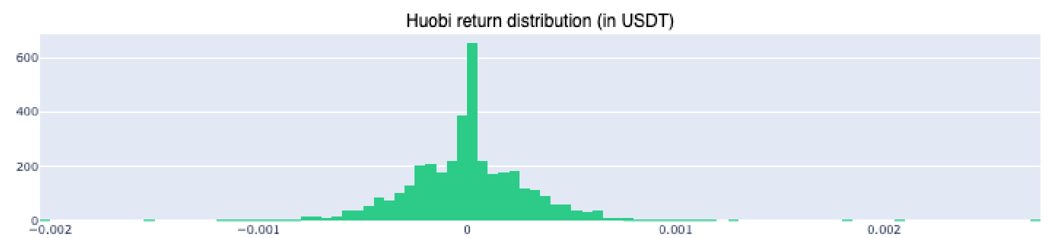
- Side : **Sell**
- Size : **SizeOrder *(*)***
- Price : $\mathbf{H}_{\textbf{Bid-price [i]}}$

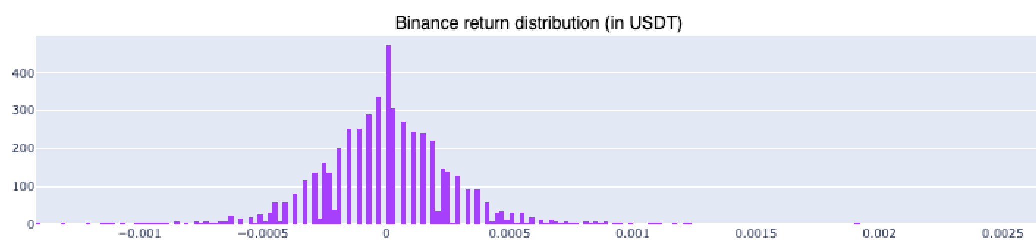
*(*) **SizeOrder** = min[$B_{\text{Ask-size [i]}}$, $H_{\text{Bid-size [i]}}$]*

Profit : ($B_{\text{Ask-price [i]}} - H_{\text{Bid-price [i]}}$) * SizeOrder

Data Structure :

The data utilized in our strategy contains order books and execution data sourced from Binance and Huobi, below the distributions and key metrics of the time-series.






```
bb.Check_Metrics()
```

```
-- Huobi Metrics --
        Historical Mean return : 1.4643865347555675e-06
        Historical Volatility : 0.00029795876058763895
-- Binance Metrics --
        Historical Mean return : 1.1710827337546865e-06
        Historical Volatility : 0.00027651141332626266
```

1.2 Scenarios

I created two scenarios, each representing different approaches one idealistic and the other realistic and then proceeded to compare them.

Scenario 1 - Idealistic scenario - Assumptions :

1. We assume that the transaction time for transferring LINK between platforms is instantaneous.
2. We assume to be the sole participant engaged in the orderbook, executing both buying and selling orders.
3. We assume we will always have sufficient funds to capitalize on arbitrage opportunities of any size that may arise
4. In the initial phase, we assume there are no transaction fees (subsequently, we will backtest to determine the maximum fees that can be reached)

Scenario 2 - Realistic scenario - Assumptions :

1. To achieve more realistic outcomes, I chose to introduce a random distribution to the generated profits, simulating a scenario where our profit is lost 20% of the time. This approach will enable us to introduce occasional random minor losses into our strategy, potentially arising from spread fluctuations during our arbitrage opportunities. Ideally, we would compute the expected value of our strategy by running it 100 times or more.
2. The rest of the assumtions remain unchanged

1.3 Risk metrics

What risk metrics are selected for our backtest ?

1. Sharpe Ratio : is used to quantify the return per unit of risk normal distribution, based on the assumption that the return distribution follows a normal distribution. However our arbitrage strategy targets consistent small returns, with occasional losses, it has a negative impact on the risk metric.

$$Sharpe\ Ratio = \frac{Expected\ portfolio\ return\ -\ Risk\ free\ rate}{Standard\ deviation\ of\ portfolio\ return}$$

2. Sortino Ratio : is a ratio which does not require the assumption that the return distribution follows a normal distribution, instead it uses the standard deviation of the negative returns. Which in our case might be a better indicator of risk for our arbitrage strategy

$$Sortino\ Ratio = \frac{Expected\ portfolio\ return\ -\ Risk\ free\ rate}{Downside\ ortfolio}$$

1.4 Data quality controls

The first step, has been to go through the data ensuring accurate data as input in our algorithm - model.

What measures are taken to validate the data ?
1. First, we verify that the datasets align in terms of data structure.
2. We ensure the quality of the data within our datasets

1.4.1 OrderBooks - list of the controls implemented

```python
# Binance and Huobi Orderbook Match
if orderbook_hb.shape == orderbook_bnc.shape:
    print('Result : Orderbooks shape are matching')
else:
    print('Result : Orderbooks shape do not match')
```

```
Result : Orderbooks shape are matching
```

```python
# Timestamp match between Binance and Huobi Orderbooks
if not False in pd.Series(orderbook_hb.index.values == orderbook_bnc.index.values).value_counts().keys():
    print('Result : Timestamps match between Binance and Huodi Orderbookds')
else:
    print('Result : Timestamps do not match between Binance and Huodi Orderbookds')
```

```
Result : Timestamps match between Binance and Huodi Orderbookds
```

```python
# Binance and Huobi Orderbook Columns Match
if not False in (orderbook_hb.columns == orderbook_bnc.columns):
    print('Result : Columns match between datasets')
else:
    print('Result : Columns do not match between datasets')
```

```
Result : Columns match between datasets
```

```python
# No null value into Huobi Orderbook
if orderbook_hb[orderbook_hb.isnull() == False].shape == orderbook_hb.shape:
    print('Result : No null values in Huobi dataset')
else:
    print('Result : There are null values in Huobi dataset')
```

```
Result : No null values in Huobi dataset
```

```python
# No null value into Binance Orderbook
if orderbook_bnc[orderbook_bnc.isnull() == False].shape == orderbook_bnc.shape:
    print('Result : No null values in Binance dataset')
else:
    print('Result : There are null values in Binance dataset')
```

```
Result : No null values in Binance dataset
```

1.4.2 Bar data - list of the controls implemented

```python
# Only Link/USDT is filled into data
if bar_hb['symbol'].unique()[0][:9] == bar_bnc['symbol'].unique()[0][:9]:
    print('Resut : Only Link/USDT is filled in our data')
else:
    print('Resut : Not only Link/USDT is filled in our data')
```

```
Resut : Only Link/USDT is filled in our data
```

```python
# Binance and Huobi Bat do not Match, more value into bar_bnc
if bar_hb.shape == bar_bnc.shape:
    print('Result : Bar data shapes match')
elif bar_bnc.shape[0] > bar_hb.shape[0]:
    print('Result : More Bar date in Binance dataset')
else:
    print('Result : More Bar date in Huobi dataset')
```

```
Result : More Bar date in Binance dataset
```

```python
# Binance and Huobi Bar Columns Match
if not False in bar_hb.columns == bar_bnc.columns:
    print('Result : Binance and Huobi bar columns match')
else:
    print('Result : Binance and Huobi bar columns do not match')
```

```
Result : Binance and Huobi bar columns match
```

```python
# No null value into Huobi bar
if bar_hb[bar_hb.isnull() == False].shape == bar_hb.shape:
    print('Result : No null values in Huobi dataset')
else:
    print('Result : There are null values in Huobi dataset')
```

```
Result : No null values in Huobi dataset
```

```python
# No null value into Binance bar
if bar_bnc[bar_bnc.isnull() == False].shape == bar_bnc.shape:
    print('Result : No null values in Binance dataset')
else:
    print('Result : There are null values in Binance dataset')
```

```
Result : No null values in Binance dataset
```

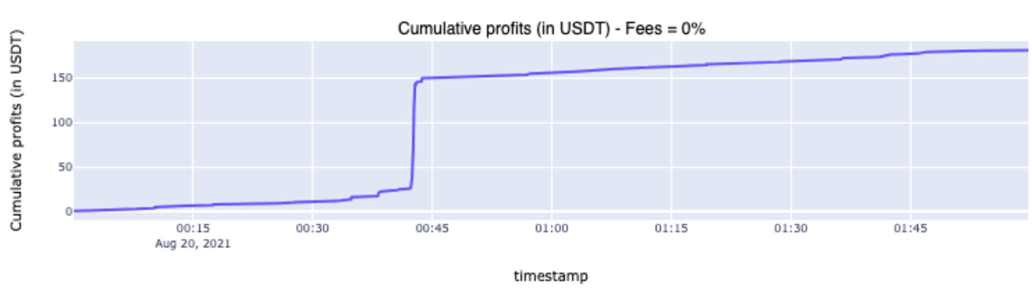
1.5 Results

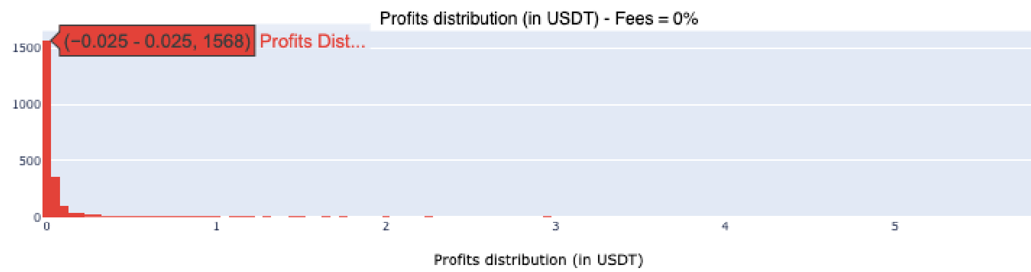
1.5.1 Scenario 1 - Idealistic scenario

Reminder of the assumptions :

1. We assume that the transaction time for transferring LINK between platforms is instantaneous.
2. We assume to be the sole participant engaged in the orderbook, executing both buying and selling orders.
3. We assume we will always have sufficient funds to capitalize on arbitrage opportunities of any size that may arise
4. In the initial phase, we assume there are no transaction fees (subsequently, we will backtest to determine the maximum fees that can be reached)

As observed below, the outcomes from our initial scenario are extremely positive, yielding a total profit of **181.955631 USDT** across **2283** transactions.
(you can access the strategy logs in the file named "scenario_1.csv.")





Identifying the risks associate to our strategy can be a complex task. In this realistic scenario, we assumed that we can subsequently buy and sell at a move favorable price on the alternate platform which imply that our risk is null.

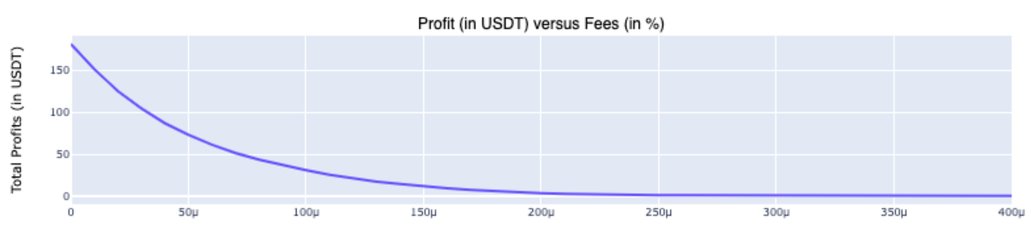
Bybit is offering a 2.5% yield staking for a 30-day period on USDT, I've chosen to use this rate as the risk-free rate, adjusting it to match our returns window, giving us a risk-free rate of 0.00009645% (earning by second on USDT).

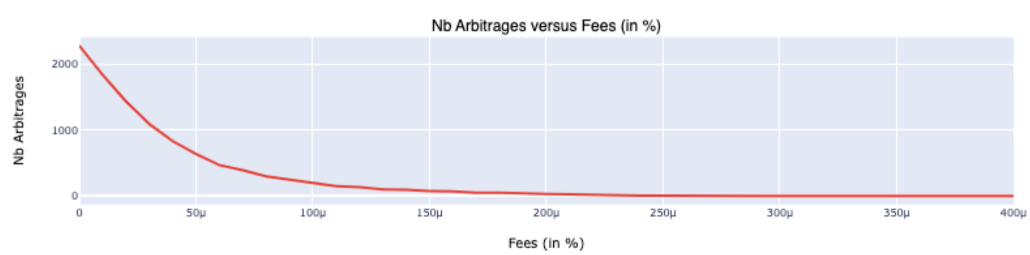
Furthermore, it's worth noting that our performance exhibits a positive Sharpe ratio and a Sortino ratio equal to "nan." This outcome was anticipated because the Sortino ratio considers losses to downgrade the ratio, and in our case, we had no losses.

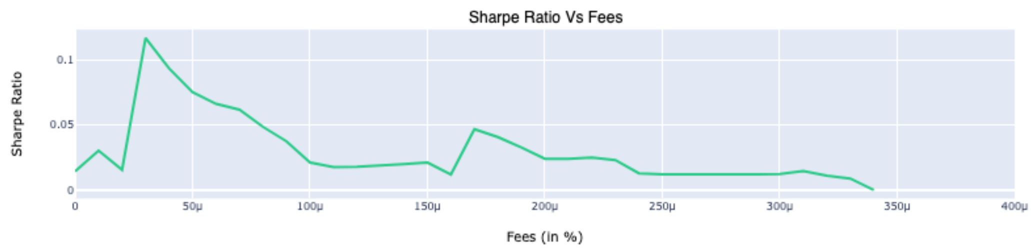
```
bb.run_risk_metrics(res)
```

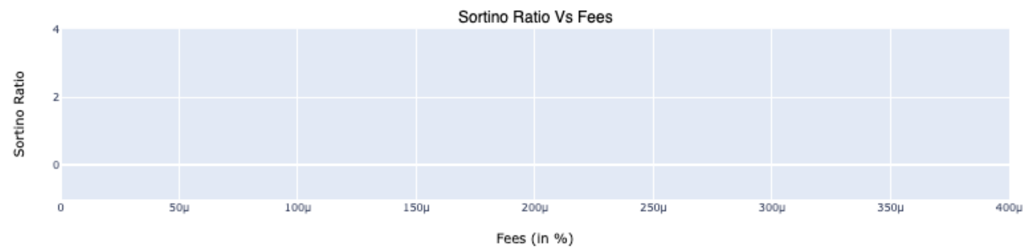
```
Sharpe Ratio :  0.014472100687053254
Sortino Ratio :  nan
```

As described below, I've decided to rerun the strategy, applying fees to determine the threshold fees that can be reached.
We can observe that our strategy is performing well up to **0.04%** fees, which is the fee limit beyond which profitable arbitrage opportunities cannot be found.









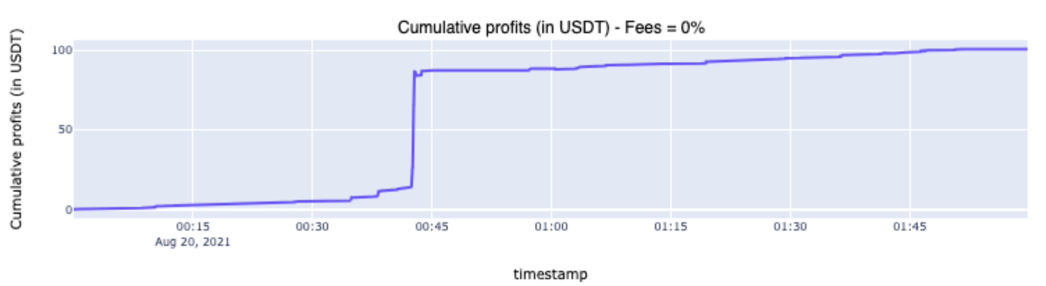
1.5.2 Scenario 2 - Realistic scenario

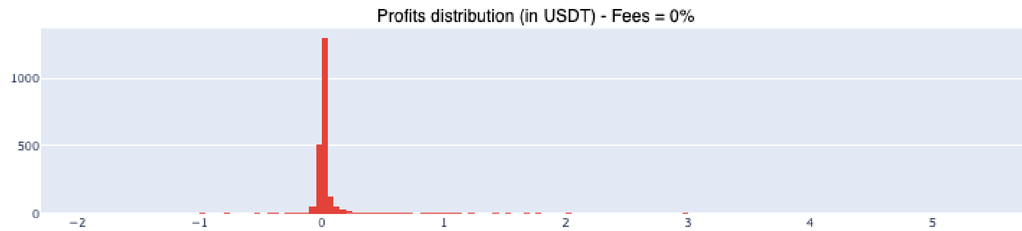
Reminder of the assumptions :

1. To achieve more realistic outcomes, I chose to introduce a random distribution to the generated profits, simulating a scenario where our profit is lost 20% of the time. This approach will enable us to introduce occasional random minor losses into our strategy, potentially arising from spread fluctuations during our arbitrage opportunities. Ideally, we would compute the expected value of our strategy by running it 100 times or more.
2. The rest of the assumtions remain unchanged

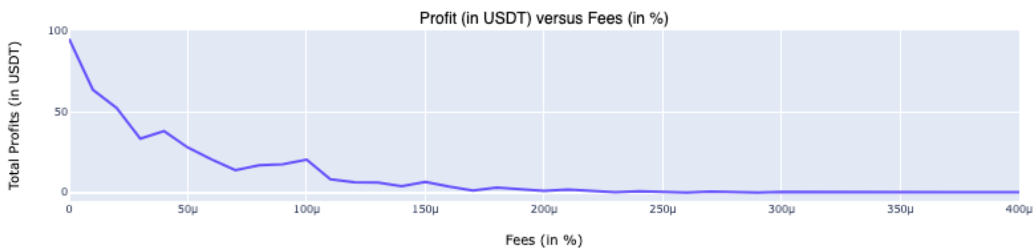
As observed below, the outcomes from our second scenario remain extremely positive, yielding a total profit of **101 USDT** across **2283** transactions. (positive and negative trades) (you can access the strategy logs in the file named "scenario 2.csv.")

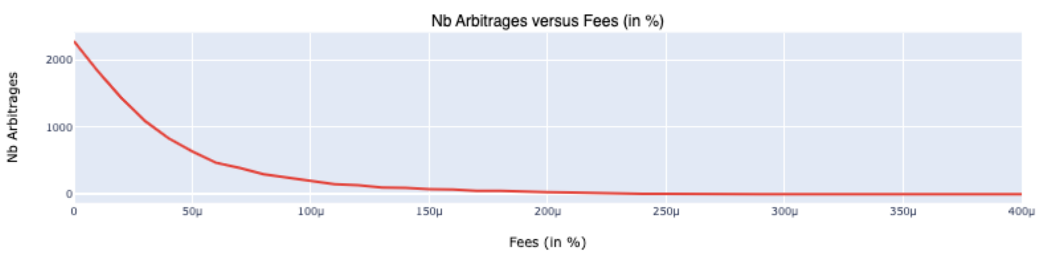
Even applying a 20% chance of loss to our arbitrage strategy remain profitable, however it would have been interesting to run the strategy 100 times or more to get better results computing the expected value. (due to time-computation I haven't had the time to run the backtest on more epochs).

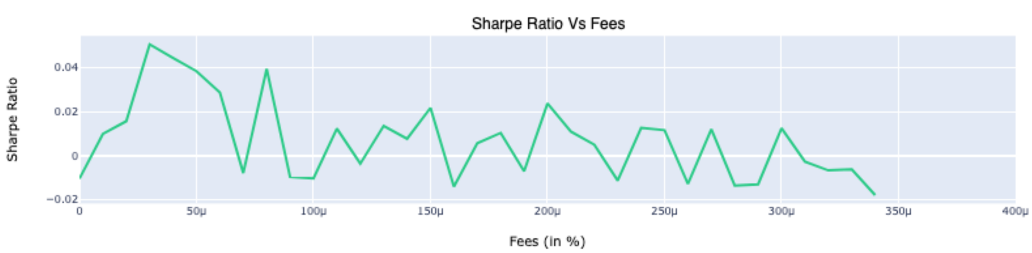


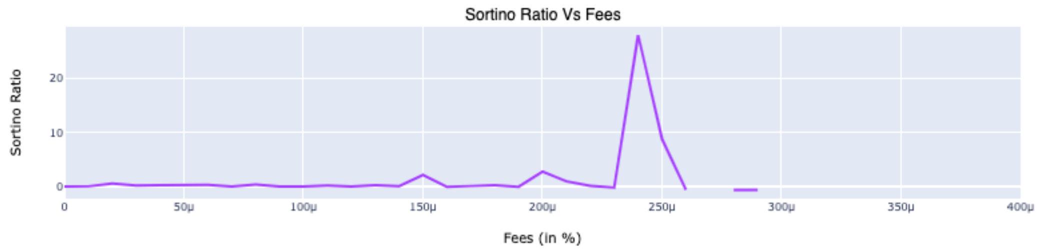


As applied to the first scenario, I backtested the strategy applying fees to determine the limit threshold fees to our strategy. We can see that the profits are declining more rapidly compared to the previous scenario, and our strategy is generating negative profits when the fees reach 0.0003%.








Sortino Ratio Vs Fees
Sortino Ratio
20
10
0
0
50μ
100μ
150μ
200μ
250μ
300μ
350μ
400μ
Fees (in %)

2 Appendix

Technical Test Crypto.com, 14/10/2023

Hugo Morel

Librairies

In [629…
```
import pandas as pd
from pandas.tseries.holiday import *
import numpy as np
import threading
import time
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.express as px
from random import seed
from random import randint
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Display All dataframe columns
pd.set_option('display.max_columns', None)
```

In [3]:
```
# Desable scrolling
```

In [4]:
```
%%javascript
IPython.OutputArea.auto_scroll_threshold = 9999
```

Q2 - Design an arbitrage strategy on LINK/USDT

Load Data

In [5]:
```
# bnc Data
bar_bnc                                                    = pd.read_csv('../bar_bnc.csv')
orderbook_bnc                                              = pd.read_csv('../orderbook_bnc.csv'
# Set 'timestamp' as index
bar_bnc.set_index('timestamp',inplace = True)
orderbook_bnc.set_index('timestamp',inplace = True)
# hb Data
bar_hb                                                     = pd.read_csv('../bar_hb.csv')
orderbook_hb                                               = pd.read_csv('../orderbook_hb.csv')
# Set 'timestamp' as index
bar_hb.set_index('timestamp',inplace = True)
orderbook_hb.set_index('timestamp',inplace = True)
```

1. Design an arbitrage strategy on LINK/USDT between Binance and Huobi based on provided data.

Analysis HB and BNC data

Analysis Orderbook Data

In [104…
```python
# Timestamp match between Binance and Huobi Orderbooks
if not False in pd.Series(orderbook_hb.index.values == orderbook_bnc.index.values).value
    print('Result : Timestamps match between Binance and Huodi Orderbookds')
else:
    print('Result : Timestamps do not match between Binance and Huodi Orderbookds')
```

```
Result : Timestamps match between Binance and Huodi Orderbookds
```

In [116…
```python
# Binance and Huobi Orderbook Match
if orderbook_hb.shape == orderbook_bnc.shape:
    print('Result : Orderbooks shape are matching')
else:
    print('Result : Orderbooks shape do not match')
```

```
Result : Orderbooks shape are matching
```

In [111…
```python
# Binance and Huobi Orderbook Columns Match
if not False in (orderbook_hb.columns == orderbook_bnc.columns):
    print('Result : Columns match between datasets')
else:
    print('Result : Columns do not match between datasets')
```

```
Result : Columns match between datasets
```

In [113…
```python
# No null value into Huobi Orderbook
if orderbook_hb[orderbook_hb.isnull() == False].shape == orderbook_hb.shape:
    print('Result : No null values in Huobi dataset')
else:
    print('Result : There are null values in Huobi dataset')
```

```
Result : No null values in Huobi dataset
```

In [115…
```python
# No null value into Binance Orderbook
if orderbook_bnc[orderbook_bnc.isnull() == False].shape == orderbook_bnc.shape:
    print('Result : No null values in Binance dataset')
else:
    print('Result : There are null values in Binance dataset')
```

```
Result : No null values in Binance dataset
```

Analysis Bar Data

In [131…
```python
# Only Link/USDT is filled into data
if bar_hb['symbol'].unique()[0][:9] == bar_bnc['symbol'].unique()[0][:9]:
    print('Resut : Only Link/USDT is filled in our data')
else:
    print('Resut : Not only Link/USDT is filled in our data')
```

```
Resut : Only Link/USDT is filled in our data
```

In [124…
```python
# Binance and Huobi Bat do not Match, more value into bar_bnc
if bar_hb.shape == bar_bnc.shape:
    print('Result : Bar data shapes match')
elif bar_bnc.shape[0] > bar_hb.shape[0]:
    print('Result : More Bar date in Binance dataset')
else:
    print('Result : More Bar date in Huobi dataset')
```

```
Result : More Bar date in Binance dataset
```

In [128…
```python
# Binance and Huobi Bar Columns Match
if not False in bar_hb.columns == bar_bnc.columns:
    print('Result : Binance and Huobi bar columns match')
```

```
else:
    print('Result : Binance and Huobi bar columns do not match')
```

```
Result : Binance and Huobi bar columns match
```

In [130…

```
# No null value into Huobi bar
if bar_hb[bar_hb.isnull() == False].shape == bar_hb.shape:
    print('Result : No null values in Huobi dataset')
else:
    print('Result : There are null values in Huobi dataset')
```

```
Result : No null values in Huobi dataset
```

In [129…

```
# No null value into Binance bar
if bar_bnc[bar_bnc.isnull() == False].shape == bar_bnc.shape:
    print('Result : No null values in Binance dataset')
else:
    print('Result : There are null values in Binance dataset')
```

```
Result : No null values in Binance dataset
```

2. Code a backtesting program by yourself with Python to test your strategy. Open source backtesting framework is not allowed.

In [748…

```
class Backtesting:

    def __init__(self,orderbook_hb,orderbook_bnc,bar_hb,bar_bnc, scenario):
        # Load Binance and Huobi Oderbooks
        self.orderbook_hb       = orderbook_hb
        self.orderbook_bnc      = orderbook_bnc

        # Load History price
        self.bar_bnc            = bar_bnc
        self.bar_hb             = bar_hb

        # Select scenario
        self.scenario = scenario

        # Set fees; % to real value
        self.Fees               = 0
        self.Ask_Fees           = (1+self.Fees)
        self.Bid_Fees           = (1-self.Fees)
        self.Fees_increment     = 0.00001

        # Set Columns name of data_Arbitrage
        self.columns                 = ['timestamp','Ask','Bid','By on','Profits by Link

        # Create data_arbitrage
        self.data_arbitrage     = pd.DataFrame(columns=self.columns)


        # Save Epocs Results
        self.epocs = []
        self.epocs_fees = []
        self.epocs_sharpe_ratio = []
        self.epocs_sortino_ratio = []


        # Risk Metrics
        self.risk_free_rate = 0.0000009645062


    def run_epocs_fees_limit(self):
```

```
        run = True

        while run==True:
            res = self.BacktestFunction()

            if res.empty:
                run = False
            else:
                self.epocs.append(res)
                self.epocs_fees.append(self.Fees)

                sharpe_ratio = self.Compute_sharpe_Ratio(res)

                sortino_Ratio = self.Compute_sortino_Ratio(res)


                self.epocs_sharpe_ratio.append(sharpe_ratio)
                self.epocs_sortino_ratio.append(sortino_Ratio)

                print("Fees : ", self.Fees, '- Profit :', self.epocs[-1]['Cumulative of

                self.Fees                = self.Fees + self.Fees_increment
                self.Ask_Fees            = (1+self.Fees)
                self.Bid_Fees            = (1-self.Fees)

        return self.epocs


    def run_risk_metrics(self,res):
        sharpe_ratio = self.Compute_sharpe_Ratio(res)

        sortino_Ratio = self.Compute_sortino_Ratio(res)

        print("Sharpe Ratio : ",sharpe_ratio)
        print("Sortino Ratio : ",sortino_Ratio)

    def Compute_sharpe_Ratio(self, res):

        returns_strategy = [ (res['Cumulative of Total Profits in USDT'].iloc[i+1] - res

        if len(returns_strategy) < 7200:
            returns_strategy = np.concatenate((returns_strategy, np.zeros(7200-len(retur

        mean_return = np.mean(returns_strategy)
        std_return = np.std(returns_strategy, axis=0)


        sharpe_ratio = (mean_return - self.risk_free_rate) / std_return


        return sharpe_ratio


    def Compute_sortino_Ratio(self,res):
        returns_strategy = [ (res['Cumulative of Total Profits in USDT'].iloc[i+1] - res

        returns_strategy = np.array(returns_strategy)
        expected_return = np.mean(returns_strategy)

        downside_returns = returns_strategy[returns_strategy < 0]
        downside_deviation = np.std(downside_returns)

        sortino_ratio = (expected_return - self.risk_free_rate) / downside_deviation

        return sortino_ratio
```

```python
def BacktestFunction(self):

    # Reset dataframe for epoc
    self.data_arbitrage = pd.DataFrame(columns=self.columns)

    for timestamp in self.orderbook_hb.index.values:
        # --- GET HB DATA ---
        # get Ask price
        self.Ask_hb            = orderbook_hb.loc[timestamp][['a1','a2','a3','a4',
        # get Ask Volume
        self.Ask_Volume_hb     = orderbook_hb.loc[timestamp][['av1','av2','av3','a
        # get Bid price
        self.Bid_hb            = orderbook_hb.loc[timestamp][['b1','b2','b3','b4',
        # get Bid Volume
        self.Bid_Volume_hb     = orderbook_hb.loc[timestamp][['bv1','bv2','bv3','b

        # --- GET BNC DATA ---
        # get Ask price
        self.Ask_bnc           = orderbook_bnc.loc[timestamp][['a1','a2','a3','a4'
        # get Ask Volume
        self.Ask_Volume_bnc    = orderbook_bnc.loc[timestamp][['av1','av2','av3','
        # get Bid price
        self.Bid_bnc           = orderbook_bnc.loc[timestamp][['b1','b2','b3','b4'
        # get Bid Volume
        self.Bid_Volume_bnc    = orderbook_bnc.loc[timestamp][['bv1','bv2','bv3','

        # --- RECURSIVE FUNCTIONS
        # Huobi to Binance Recursive function
        self.Check_Arbitrage_Hb_To_Bnc(0,0,timestamp)
        # Binance to Huobi Recursive function
        self.Check_Arbitrage_Bnc_To_Hb(0,0,timestamp)

    # Update data_arbitrage
    self.Update_Data_Arbitrage()


    return self.data_arbitrage


def Check_Arbitrage_Hb_To_Bnc(self,i_hb,i_bnc,timestamp):
    # Check if Ask of Huobi with fees is lower than Bid of Binance
    if (self.Ask_hb[i_hb]*self.Ask_Fees) < (self.Bid_bnc[i_bnc]*self.Bid_Fees):
        # Check if we can sell the Huobi Volume on Binance
        if self.Ask_Volume_hb[i_hb] < self.Bid_Volume_bnc[i_bnc]:
            # Set Profits
            profits                                  = (self.Bid_bnc[i_bnc]

            if self.scenario==2 and  randint(0, 10) <= 2:
                profits = -profits

            # Push data into "data_arbitrage"
            self.data_arbitrage.loc[len(self.data_arbitrage)] = [timestamp,self.Ask_

            # In we sell all the Huobi volume af first Ask we gonna check the next H
            if (i_hb+1) < len(self.Ask_Volume_bnc):
                # Update Volume
                self.Bid_Volume_bnc[i_bnc]           = self.Bid_Volume_bnc[
                # Recursive Call
                self.Check_Arbitrage_Hb_To_Bnc(i_hb+1,i_bnc,timestamp)

        # if we cant sell all the volume, we gonna check the next bid of binance
        elif (i_bnc+1) < len(self.Bid_Volume_hb):
            # Set Profits
            profits                                  = (self.Bid_bnc[i_bnc]
            if self.scenario==2 and randint(0, 10) <= 2:
                profits = -profits
```

```
            # Push data into "data_arbitrage"
            self.data_arbitrage.loc[len(self.data_arbitrage)] = [timestamp,self.Ask_
            # Update Volume
            self.Ask_Volume_hb[i_hb]                           = self.Ask_Volume_hb[i_
            # Recursive Call
            self.Check_Arbitrage_Hb_To_Bnc(i_hb,i_bnc+1,timestamp)


    def Check_Arbitrage_Bnc_To_Hb(self,i_hb,i_bnc,timestamp):
        if (self.Ask_bnc[i_bnc]*self.Ask_Fees) < (self.Bid_hb[i_hb]*self.Bid_Fees):
            if self.Ask_Volume_bnc[i_bnc] < self.Bid_Volume_hb[i_hb]:
                # Set Profits
                profits                                        = (self.Bid_hb[i_hb]*s
                if self.scenario==2 and randint(0, 10) <= 2:
                    profits = -profits
                # Push data into "data_arbitrage"
                self.data_arbitrage.loc[len(self.data_arbitrage)] = [timestamp,self.Ask_

                if (i_bnc+1) < len(self.Ask_Volume_bnc):
                    # Update Volume
                    self.Bid_Volume_hb[i_hb]                   = self.Bid_Volume_hb[i_

                    # Recursive Call
                    self.Check_Arbitrage_Bnc_To_Hb(i_hb,i_bnc+1,timestamp)

            elif (i_hb+1) < len(self.Bid_Volume_hb):
                # Set Profits
                profits                                        = (self.Bid_hb[i_hb]*s
                if self.scenario==2 and randint(0, 10) <= 2:
                    profits = -profits
                ## Push data into "data_arbitrage"
                self.data_arbitrage.loc[len(self.data_arbitrage)] = [timestamp,self.Ask_
                # Update Volume
                self.Ask_Volume_bnc[i_bnc]                     = self.Ask_Volume_bnc[
                # Recursive Call
                self.Check_Arbitrage_Bnc_To_Hb(i_hb+1,i_bnc,timestamp)


    def Update_Data_Arbitrage(self):
        # Sort Data Arbitrage
        self.data_arbitrage.sort_values(by='timestamp',inplace=True)
        # Create new columns with Cumulative Profits
        self.data_arbitrage['Cumulative of Total Profits in USDT'] = self.data_arbitrage

        self.data_arbitrage.reset_index(drop=True,inplace=True)

        # Add 'returns' columns to 'bar_hb'
        self.bar_hb['returns']                                  = (self.bar_hb['close
        # Add 'returns' columns to 'bar_bnc'
        self.bar_bnc['returns']                                 = (self.bar_bnc['clos


    def Display_Profits_fees_limit(self):
        # Set Title
        title_profit_fees               = "Profit (in USDT) versus Fees (in %)"
        title_nb_trades_fees            = "Nb Arbitrages versus Fees (in %)"
        title_sharpe_ratio_versus_fees  = "Sharpe Ratio Vs Fees"
        title_sortino_ratio_versus_fees = "Sortino Ratio Vs Fees"

        # Set graph
        fig                        = make_subplots(rows           = 4,
                                                   cols           = 1,
                                                   subplot_titles = [title_profit_fees, t


        # Add first graph
```

```python
    profits = [self.epocs[i]['Cumulative of Total Profits in USDT'].iloc[-1] for i i
    fig.add_trace(
            go.Scatter(x          = self.epocs_fees,
                       y          = profits,
                       name       = 'Total profits Vs Fees'),
                       row        = 1,
                       col        = 1
        )

    nb_trades = [len(self.epocs[i]['Cumulative of Total Profits in USDT']) for i in
    fig.add_trace(
            go.Scatter(x          = self.epocs_fees,
                       y          = nb_trades,
                       name       = 'Nb Arbitrages Vs Fees'),
                       row        = 2,
                       col        = 1
        )

    fig.add_trace(
            go.Scatter(x          = self.epocs_fees,
                       y          = self.epocs_sharpe_ratio,
                       name       = 'Sharpe Ratio Vs Fees'),
                       row        = 3,
                       col        = 1
        )

    fig.add_trace(
            go.Scatter(x          = self.epocs_fees,
                       y          = self.epocs_sortino_ratio,
                       name       = 'Sortino Ratio Vs Fees'),
                       row        = 4,
                       col        = 1
        )




    # Update axis Name, Size, Color
    fig.update_yaxes(title_text='Total Profits (in USDT)', row=1, col=1, title_font=
    fig.update_yaxes(title_text='Nb Arbitrages', row=2, col=1, title_font=dict(size=
    fig.update_yaxes(title_text='Sharpe Ratio', row=3, col=1, title_font=dict(size=1
    fig.update_yaxes(title_text='Sortino Ratio', row=4, col=1, title_font=dict(size=

    fig.update_xaxes(title_text='Fees (in %)', row=1, col=1, title_font=dict(size=10
    fig.update_xaxes(title_text='Fees (in %)', row=2, col=1, title_font=dict(size=10
    fig.update_xaxes(title_text='Fees (in %)', row=3, col=1, title_font=dict(size=10
    fig.update_xaxes(title_text='Fees (in %)', row=4, col=1, title_font=dict(size=10

    # Update tiles Size, Color
    fig.update_annotations(font=dict(family="Helvetica", size=12, color='black'))

    # Change charts size
    fig.update_layout(
                    autosize=False,
                    width=1000,
                    height=1000,
                )

    return fig

def Display_Profits(self):

    # Set title
    title_fig_1             = "Cumulative profits (in USDT) - Fees = "    + str(s
    title_fig_2             = "Profits distribution (in USDT) - Fees = "  + str(s
    title_fig_3             = "Huobi return distribution (in USDT)"
```

```
        title_fig_4                 = "Binance return distribution (in USDT)"


        # Set graph
        fig                         = make_subplots(rows            = 4,
                                                    cols            = 1,
                                                    subplot_titles = [title_fig_1,title_fi
        # Add first graph
        fig.add_trace(
            go.Scatter(x            = self.data_arbitrage['timestamp'].values,
                       y            = self.data_arbitrage['Cumulative of Total Profits in
                       name         = 'Cumulative Profits'),
                       row          = 1,
                       col          = 1
        )
        # Add Second graph
        fig.add_trace(
            go.Histogram(x          = self.data_arbitrage['Total Profits in USDT'].astype(
                         name       = 'Profits Distribution',
                         autobinx = True),
                         row        = 2,
                         col        = 1
        )
        # Add third graph
        fig.add_trace(
            go.Histogram(x          = self.bar_hb['returns'],
                         name       = 'Huobi return Distribution',
                         autobinx = True),
                         row        = 3,
                         col        = 1
        )
        # Add fourth graph
        fig.add_trace(
            go.Histogram(x          = self.bar_bnc['returns'],
                         name       = 'Binance return Distribution',
                         autobinx = True),
                         row        = 4,
                         col        = 1
        )


        # Update axis Name, Size, Color
        fig.update_yaxes(title_text='Cumulative profits (in USDT)', row=1, col=1, title_
        fig.update_yaxes(row=2, col=1, title_font=dict(size=10, color='black'), tickfont
        fig.update_yaxes(row=3, col=1, title_font=dict(size=10, color='black'), tickfont
        fig.update_yaxes(row=4, col=1, title_font=dict(size=10, color='black'), tickfont

        fig.update_xaxes(title_text='timestamp', row=1, col=1, title_font=dict(size=10,
        fig.update_xaxes(title_text='Profits distribution (in USDT)', row=2, col=1, titl
        fig.update_xaxes(title_text='Huobi return distribution (in USDT)', row=3, col=1,
        fig.update_xaxes(title_text='Binance return distribution (in USDT)', row=4, col=

        # Update tiles Size, Color
        fig.update_annotations(font=dict(family="Helvetica", size=12, color='black'))

        # Change charts size
        fig.update_layout(
                            autosize=False,
                            width=1000,
                            height=1000,
                         )

        return fig

    def Export_Data(self, name):
        name = name + ".csv"
```

```
        # Export data to csv
        self.data_arbitrage.to_csv(name)

    def Check_Metrics(self):
        print("-- Huobi Metrics --",
              "\n\tHistorical Mean return : " + str(np.mean(self.bar_hb['returns'])),
              "\n\tHistorical Volatility : "  + str(np.std(self.bar_hb['returns'])),
              "\n-- Binance Metrics --",
              "\n\tHistorical Mean return : " + str(np.mean(self.bar_bnc['returns'])),
              "\n\tHistorical Volatility : "  + str(np.std(self.bar_bnc['returns'])))
```

In [750…

```
def run_scenario_(orderbook_hb, orderbook_bnc, bar_hb,bar_bnc, scenario):
    # Init Backtesting Class
    bb = Backtesting(orderbook_hb,orderbook_bnc,bar_hb,bar_bnc,scenario)
    # Run BacktestFunction and display output DataFrame "data_arbitrage"
    display(bb.BacktestFunction())
    # Display graphs
    display(bb.Display_Profits())
    # Display(Metrics)
    bb.Check_Metrics()
    # Export Data Arbritage
    name = 'scenario_' + str(scenario)
    bb.Export_Data(name)

    # Run BacktestFunction to get the limit fees and their impacts on the performances
    bb.run_epocs_fees_limit()
    # Display graph from Backtest fees Function
    display(bb.Display_Profits_fees_limit())
```

In [ ]:

```
if __name__ == '__main__':
    scenario = 1
    run_scenario_(orderbook_hb, orderbook_bnc, bar_hb,bar_bnc, scenario)
```