STAT4601 Project: A Pairs Trading Strategy Study with ARIMA Model Forecasting

Group 6 Members:
San Wai Tung 3035388528
Sit Wai Tang 3035761603
Ye Mao 3035535066
Young Hok Ming Elwin 3035462568

May 4, 2022

Abstract

In this project we examine 27 pairs of securities in the Hong Kong Stocks and Future market, aiming at applying time series forecasting techniques to build pairs trading strategies. We'll first go through the basic mechanisms and statistical tests applied in identifying available trading pairs, followed by the process of identifying the proper ARIMA model for target security pairs after preliminary filtration. Afterward, we analyze the potential for pairs trading for the 27 identified pairs and end the report with possible risks and real-life implications of the project.

Contents

1 Part 1: Background Knowledge and Introduction

1.1 ETFs and Pairs Trading

An exchange-traded fund (ETF) is a type of pooled investment security that operates much like a mutual fund, while it can be easily bought and sold on the stock exchange just like stock with fewer limitations compared to a mutual fund. Index ETFs try to track the movement of the underlying index, with an annual tracking error normally less than 2%[1].

The introduction of ETFs began with the S&P 500 Depositary Receipts (SPDR, symbol SPY), which was incepted in 1993 by the American Stock Exchange - allowing an investor to buy or sell the entire S&P 500 index within a single security. The exponential growth of SPDR trading volume caught the attention of investors and institutions worldwide, and consequently served as a template for products that replicated individual stock indexes and more recently, specific sectors and variations within global financial markets. ETFs are simply baskets of securities designed to track or mimic the performance of an index, designed to provide exposure to broad-based indexes at a low cost.

Pairs trading is a common type of investment strategy that has been widely used in financial industry for decades. For any two assets with autoregressive behavior in their historical spread, arbitrageurs seek investment opportunities when the spread diverges from its recent history or it's autoregressive model predicted value. When the spread diverges from its recent history, the arbitrageur takes a long position in the undervalued asset and finances the position by taking a short position in the overvalued one. When the spread converges in times of rational equilibrium, the arbitrageur can close both positions to generate a profit.

Much recent research shows that despite the advent of algorithmic and high-frequency trading (HFT), pairs trading can still be a profitable venture for professionals in a less efficient market. As argued by Anders and Evans in 2021[2], pairs trading augmented by unsupervised machine learning approach yielded 22% of excess return and 0.84 Sharpe ratio after adjusting for transaction cost in OSE(Japan Market), while failing to earn significant profit in S&P 500 components. This is what intrigued us to carry out this following preliminary research on whether such trading opportunities exist in the HK makret.

1.2 Data Collection

In this study, we conduct research into 13 securities, names listed below. They're all ETFs or futures tracing three different indexes: Hang Seng Index, Hang Seng Technology Index, and Hang Seng Chinese Enterprise Index.

[1] `https://www.icicipruamc.com/blog-details/blogs/tracking-error-and-its-implication-on-your-etf-`

[2] Anders & Evans, 2021: Statistical Arbitrage using an unsupervised machine learning approach: is liquidity a predictor of profitability?

Start date and end date of each security in the source data are listed below. Some have later start-dates as they're not listed until then.

code	start-date	end-date	English Name
HK.02800	2015-01-02	2022-04-20	Tracker Fund of Hong Kong
HK.02828	2015-01-02	2022-04-20	Hang Seng China Enterprises Index ETF
HK.03032	2020-09-04	2022-04-20	Hang Seng TECH Index ETF
HK.03033	2020-08-28	2022-04-20	CSOP Hang Seng TECH Index ETF
HK.03067	2020-09-17	2022-04-20	iShares Hang Seng TECH ETF
HK.03088	2020-09-03	2022-04-20	ChinaAMC Hang Seng TECH Index ETF
HK.07200	2017-03-14	2022-04-20	CSOP Hang Seng Index Daily (2x)
HK.07226	2020-12-10	2022-04-20	CSOP Hang Seng TECH Index Daily (2x)
HK.07500	2019-05-28	2022-04-20	CSOP Hang Seng Index Daily (-2x)
HK.07552	2020-12-10	2022-04-20	CSOP Hang Seng TECH Index Daily (-2x)
HK.800000	2015-01-02	2022-04-20	Hang Seng Index
HK.800100	2015-01-02	2022-04-20	Hang Seng Chinese Enterprise Index
HK.HHImain	2017-08-14	2022-04-20	Hang Seng Chinese Enterprise Index Future Main
HK.HSImain	2015-01-02	2022-04-20	Hang Seng Index Future Main
HK.HTImain	2020-11-23	2022-04-20	Hang Seng Technology Index Future Main

Table 1: Information of each security under investigation. HK.80000(Hang Seng Index) and HK.800100(Hang Seng Chinese Enterprise Index) are not directly tradable, thus needs to be represented by their futures, HK.HSImain and HK.HHImain.

2 Part 2: Identify tradable paris by cointegration tests

2.1 Cointegration tests

The first thing that naturally comes in mind for possible pairs trading target selection is correlation. However, correlations between financial time series, especially for higher frequency data(e.g. daily log return series), though seems a good measure for potential of pairs trading, are notoriously unstable when the examination period is not long enough. A supplementary statistical measure to correlation is cointegration. If two or more series are individually integrated (in the time series sense) but some linear combination of them has a lower order of integration, then the two series are said to be cointegrated. Two stocks may be perfectly correlated over short time scales, yet diverge in the long run, with one growing and the other decaying. Conversely, two stocks may follow each other, never being more than a certain distance apart, but with any correlation, positive, negative or varying. For cointegration, three main testing methods exists: Johansen, Engle-Granger, and Phillips-Ouliaris. Engle-Granger cointegration testing will be applied here for its popularity and convenience in application.

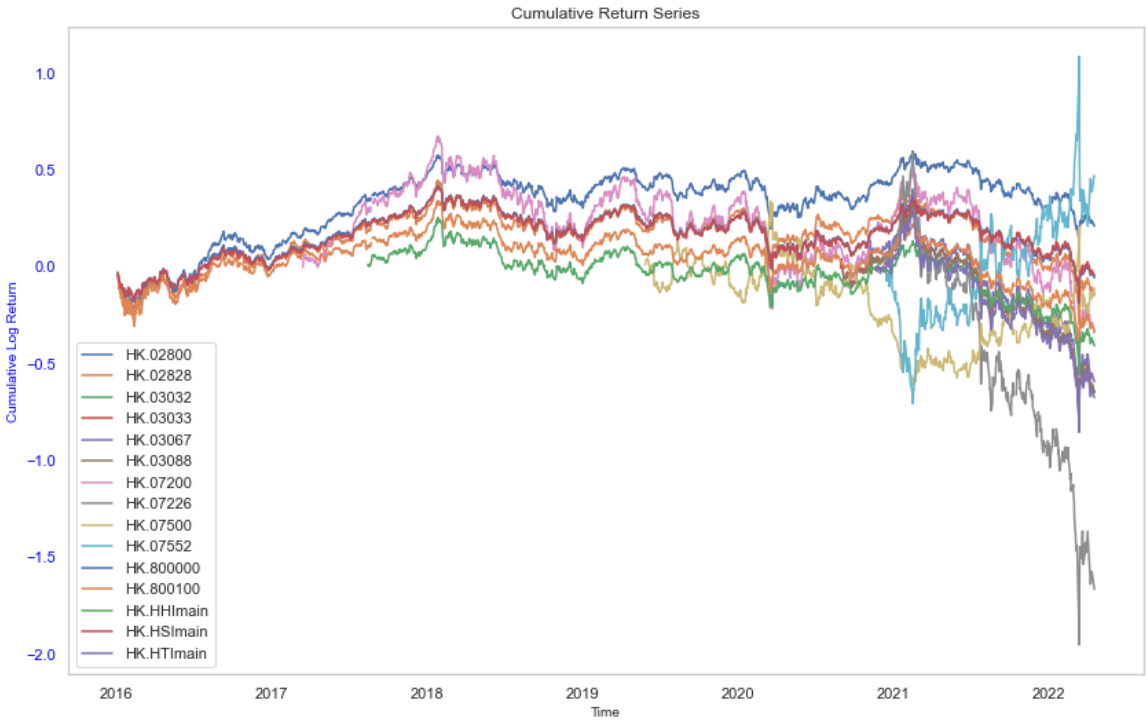


Figure 1: Cumulative log return of 13 securities and 2 indexes

To carry out Engle-Granger cointegration test, let's consider the regression model for y_t:

$$y_{1t} = \delta D_t + \phi_{1t} y_{2t} + \phi_{m-1} y_{mt} + \varepsilon_t$$

D_t is the deterministic term. From there, we can test whether ε_t is $I(1)$ or $I(0)$. The hypothesis test is as follows:

$$H_0 : \varepsilon_t \sim I(1) \implies y_t \ (no\ cointegration)$$
$$H_1 : \varepsilon_t \sim I(0) \implies y_t \ (cointegration)$$

y_t is cointegrated with a normalized cointegration vector $\alpha = (1, \phi_1, \ldots, \phi_{m-1})$.

We may test the hypothesis by checking wehther unit root exists in the residuals ε_t. The hypothesis test model is :

$$\Delta\varepsilon_t = \lambda\varepsilon_{t-1} + \sum_{j=1}^{p-1} \varphi \Delta\varepsilon_{t-j} + \alpha_t$$

$$H_0 : \lambda = 0 \ (Unit\ Root)$$
$$H_1 : \lambda < 1 \ (Stationary)$$

The test statistic is:

$$t_\lambda = \frac{\hat{\lambda}}{s_{\hat{\lambda}}}$$

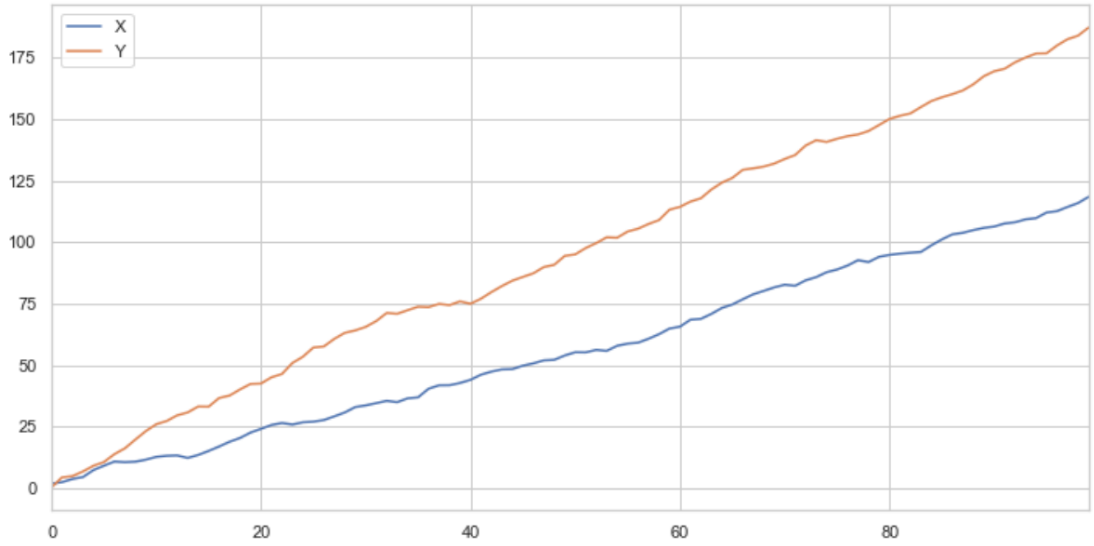


Figure 2: Example of correlated but uncointegrated series

By applying hypothesis testing in this way we're able to identified cointegration relationship between two time series. To further illustrate why correlation itself is not enought and cointegration is needed in the identification of pairs trading targets, we may take a look at the following two examples:

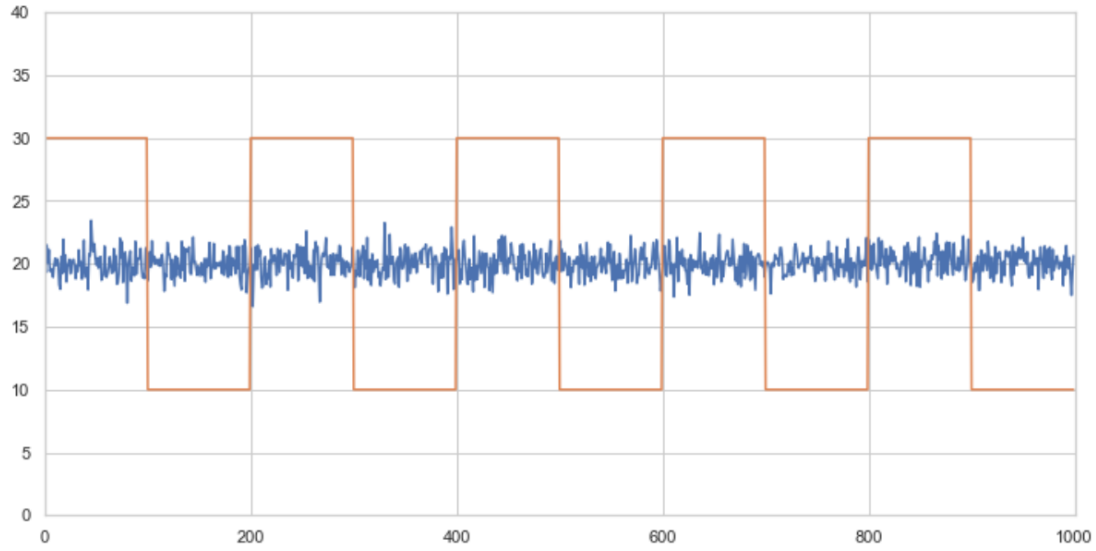


Figure 3: Example of uncorrelated but cointegrated series

In this example, two stocks have a perfect positive correlation. Correlation coefficient is 0.9957 while the p-value for Engle-Granger test is 0.2623, showing a poor indication of cointegration. From the graph, we can see that we may not be able to profit from the convergence of spread since it may never converge.

In this hypothetical example, there're two time series, one is normally distributed series and the other is a square wave. Poor correlation and strong cointegration are observed, with correlation coefficient being 0.0183, while p-value for Engle-Granger test being 0. Profits can be earned by longing the undervalued security and shorting the overvalued securities when two time series diverge with each other and unwind the position when the spread converges.

2.2 Selected pairs with high cointegration

Through cointegration test we're able to identify cointegrated pairs for potential pairs trading opportunity screening. The market is highly competitive, so without doubt not every cointegrated pairs that we screen out is going to yield high returns. After the Engle-Granger cointegration test is carried out between each pair of securities, we may screen possible opportunities in a heatmap.

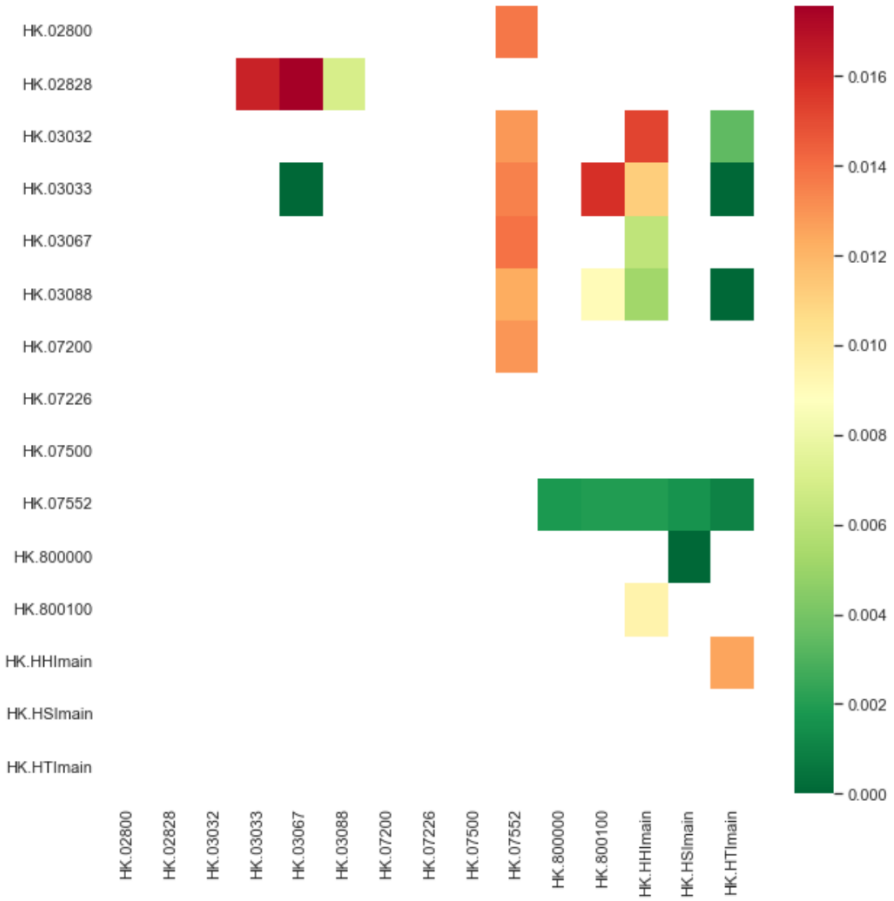


Figure 4: Heatmap of p-value of Engle-Granger test

If we set the threshold p-value for picking possible pairs to be 0.05, we will get 34 pairs of possible pairs trading targets. On the graph, pairs with Engle-Granger test p-value less than 0.02 are masked. As can be seen, 27 pairs are filered out.

3 Part 3: Model Identification and Adjustments

In total there're 34 pairs identified as cointegrated at 5% p-value threshold. Due to the space constraint we will focus on the model identification process of a selected pair HK.3033 and HK.3067. A similar process is carried out for other pairs. Both ETFs are ETFs tracing the Hangseng Technology Index, which explains for the high cointegration between the two securities.

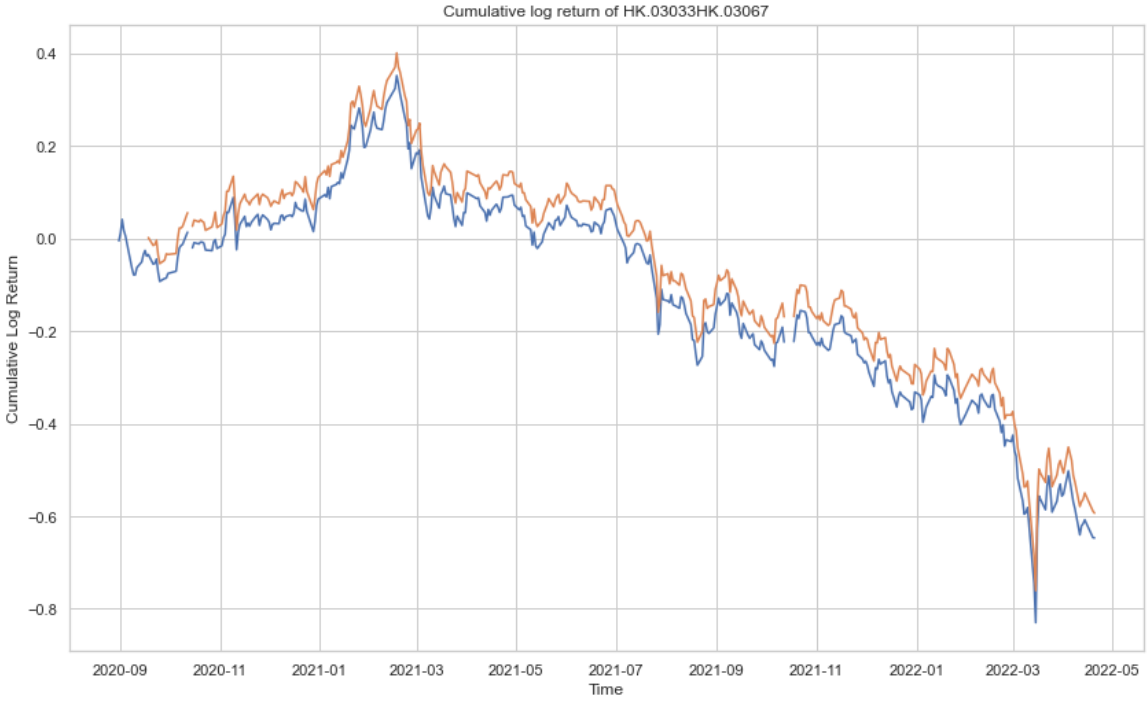


Figure 5: Cumulative Log Return of HK.03033 and HK.03067

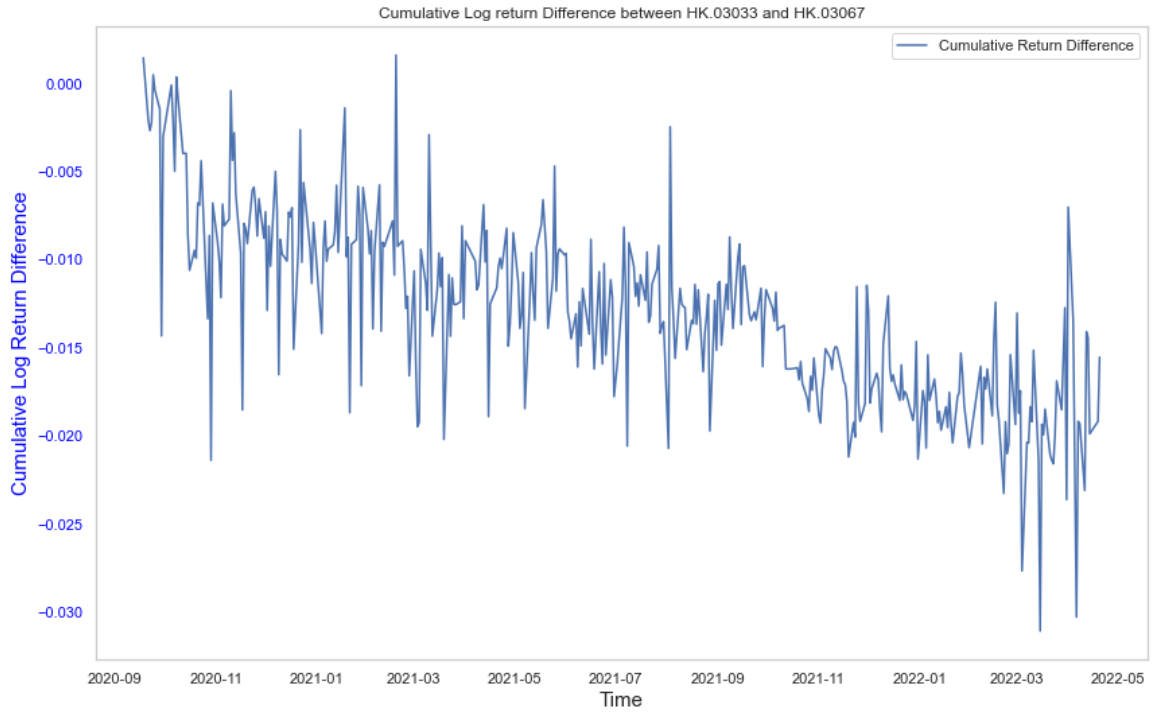


Figure 6: Cumulative Log Return Difference between HK.03033 and HK.03067

]

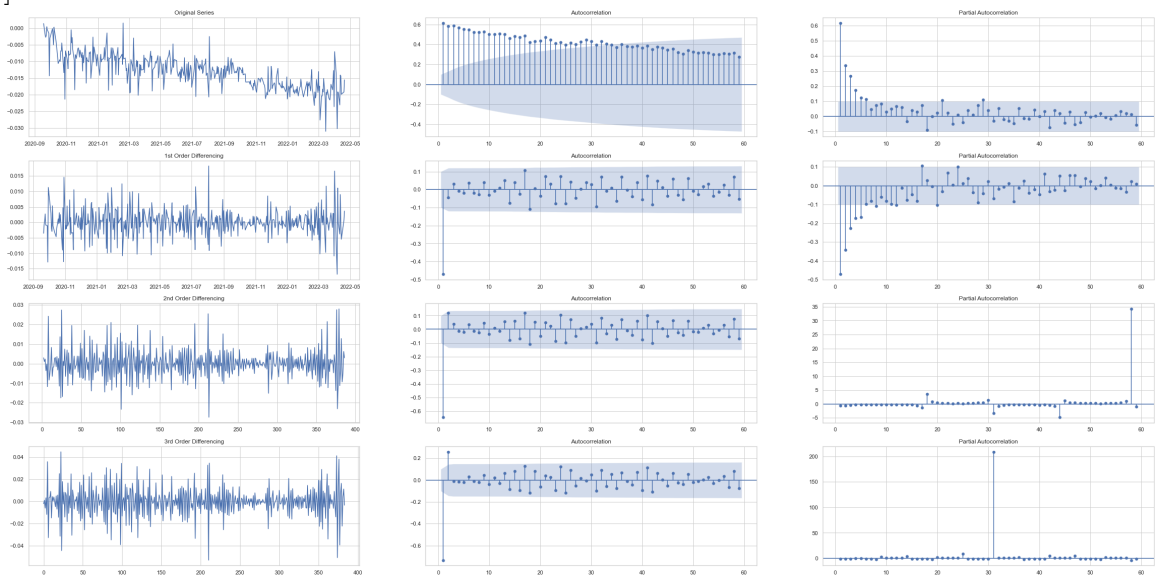


Figure 7: Time Series ACF and PACF plot of cumulative return log difference between the HK.03033 and HK.03067

From the ACF and PACF graphs we can clearly see that first-order differencing is required as consecutive positive significant autocorrelations are exhibited in the ACF graph of the original series. The existence of unit root may also be proved

by Augmented-Dickey-Fuller test. Results of adfuller test on cumulative log return difference and its 1st order difference.

	ADF value	ADF p-value
Cumulative Log Return Difference	-1.79147	0.384704
1st order difference	-9.05844	4.63298e-15

Table 2: Augmented Dickey-Fuller Test Results

From hereafter, we will focus on studying the 1st order differenced time series, which is essentially the daily log return difference between HK.03033 and HK.03067. The d terms will be automatically set to 1 to illustrate the need of 1st order difference.

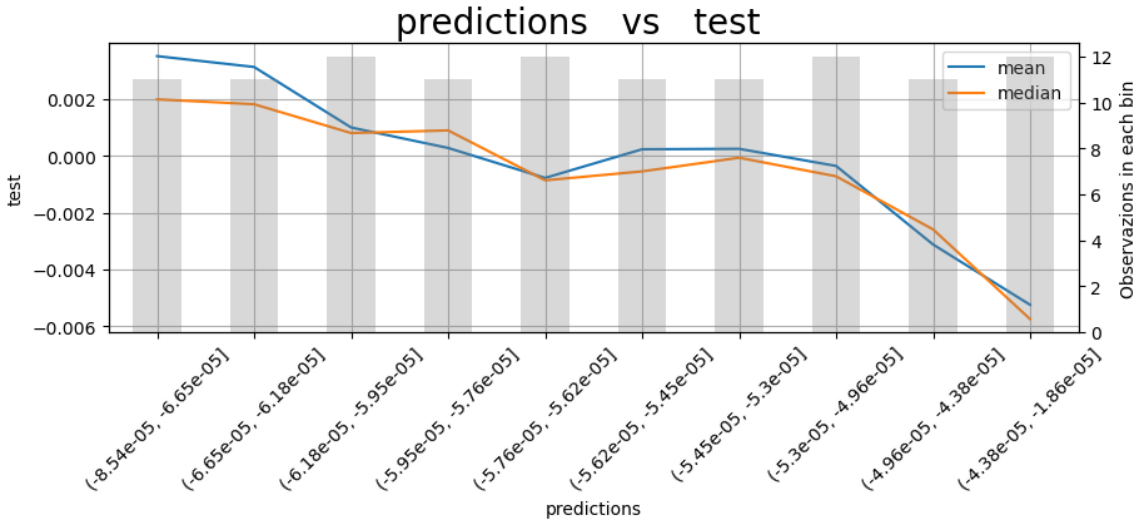


Figure 8: Null Model bin plot

3.1 Before Model Setup: Trading Rules

Before we pick the model, we'd like to discuss about how we determine our model's trading rule so as to use conduct backtesting to evaluate model's performance under transaction cost. While ARIMA provides predictions of future value of daily log return, it needs additional transformation to be turned into actual transactional orders. A typical distribution of ARIMA predictions is shown in the graph beneath.

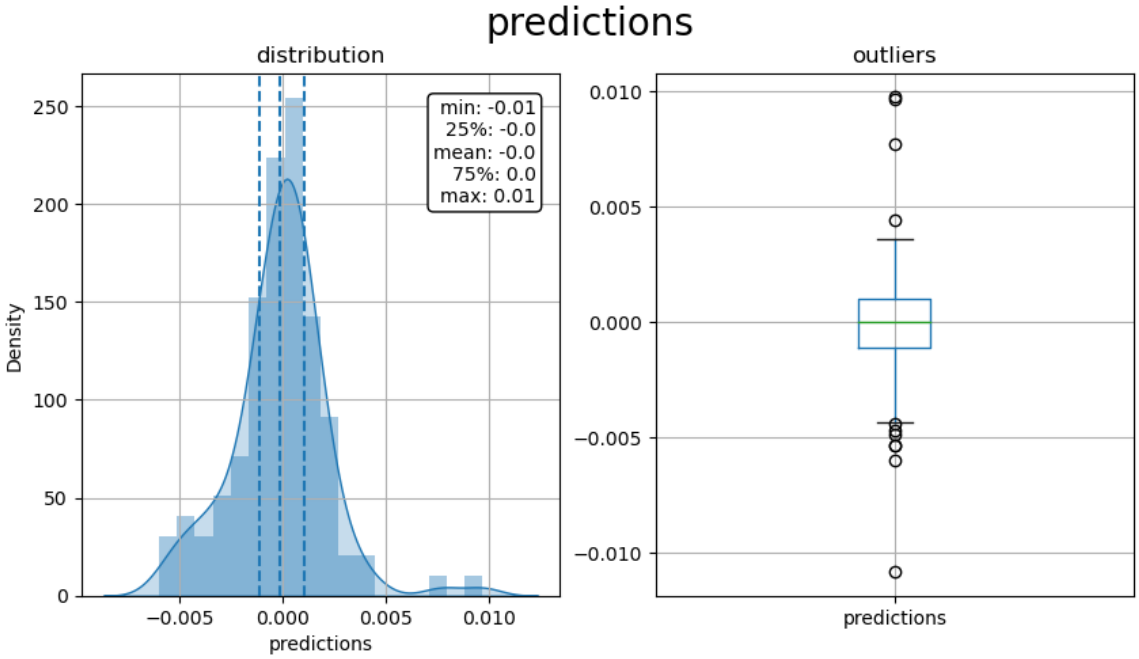


Figure 9: A typical plot of ARIMA predictions, as can be seen most values are between ± 0.01

The following transformation rules are proposed.

1. It needs to be transformed into values between -2 and 2, representing the effect of maximum leverage in real life situation. Financial meaning behind this ± 2 bound is that, while we're longing or shorting the spread, the maximum leverage we can take under normal circumstances is 100% more than our own money, and that means the maximum position we can trade on is 200% of our available assets.

2. Predictions with low absolute value should be filtered out. According to Kelly's criterion(Kelly, 1956) we should place the bets based on our confidence level and potential gain/loss from the bet. Predictions with lower absolute value should be filtered out so as to avoid noise trading and its corresponding null trading costs. Though we will not apply exactly the Kelly's Criterion, we will use an approximation, by normalizing the signals first and filter out 70% of signals in the middle and only trade on 30% of signals on tails.

The general formula for position is formulated as:

$$Norm = \frac{ARIMA\ prediction\ signal - signal\ rolling\ mean_{30days}}{signal\ rolling\ standard\ deviation_{30days}}$$

$$Position = \begin{cases} 0, & \text{if } Norm < 0.7 \\ sign(Norm) * min(abs(Norm), 2), & \text{if } Norm \geq 0.7 \end{cases}$$

Norm stands for normalized prediction signals. After transformation above, distribution of the target positions have the following distribution. When the position signal is positive, we long the spread by the exact amount. Transaction cost, leverage cost and slippage is estimated at 12bps per trade. [3]

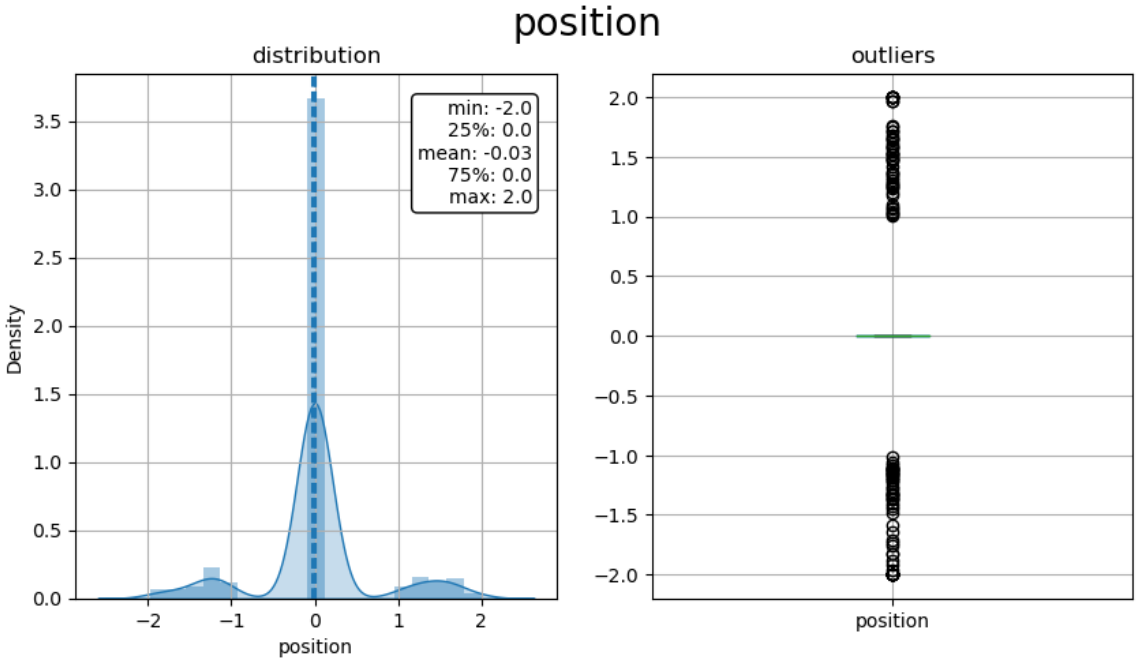


Figure 10: Distribution plot of the transformed positions

3.2 Model 0: Null Model: ARIMA(0,1,0)

Data Range for HK.03033 and HK.03067 are available from 2020/09/18 to 2022/04/11. We set the first 70% of data as training set, and the rest 30% as test set.

For each date in test set, ARIMA model uses all data available up to that timepoint to predict next day's log return difference, and sequentially make predictions for all data inside the test set from the earliest to the latest. The model then compares its prediction value with the true value to evaluate its performance.

For Null Model ARIMA(0,1,0), performance is underwhelming, as expected.

Clearly null model is not a good model for prediction, as can be proved by significant negative slope in the scatter plot and negative backtesting returns.

[3]Source: https://www.futuhk.com/en/support/topic335?lang=en-us

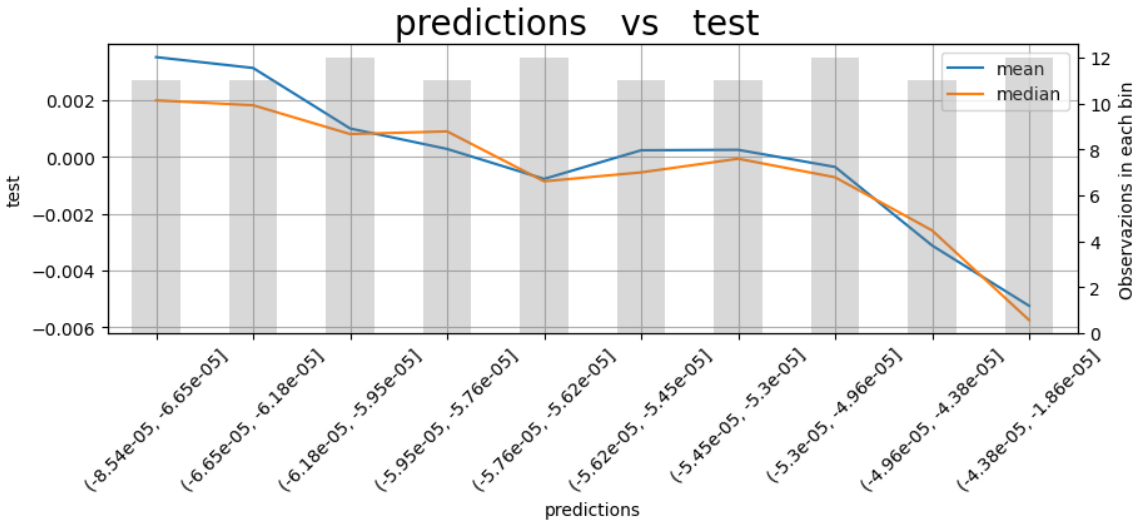


Figure 11: Null Model ARIMA(0,1,0)-Bin plot

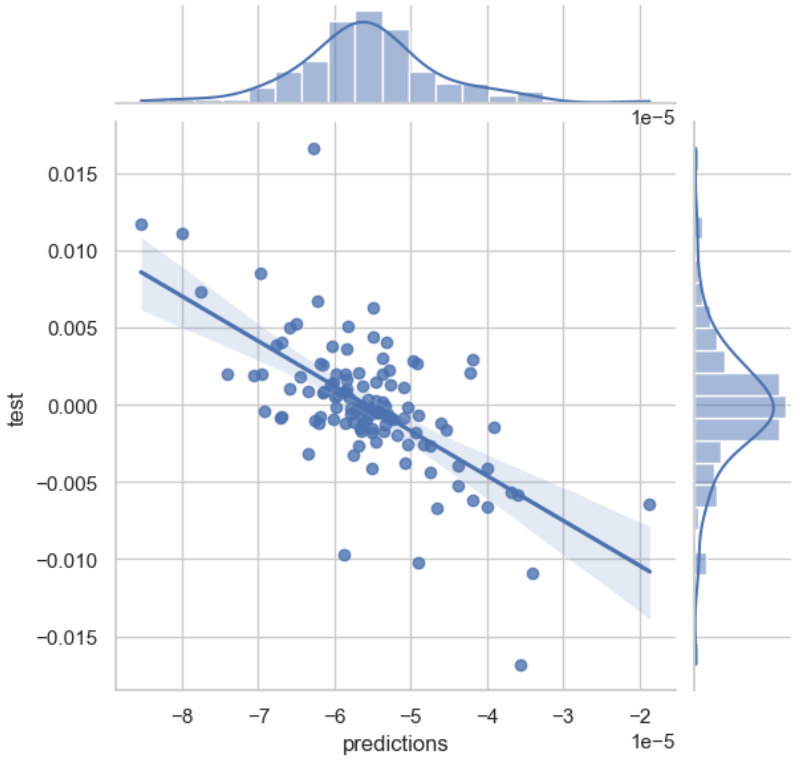


Figure 12: Null Model ARIMA(0,1,0)-Scatter plot

3.3 Model 1: ARIMA(5,1,1)

By re-examining Figure 7, our intuitive model concluded from the ACF and PACF graph for the 1st order difference is that an ARIMA(5,1,1) model may be suitable for modeling the process. Since seasonality is against the commonly observed character-

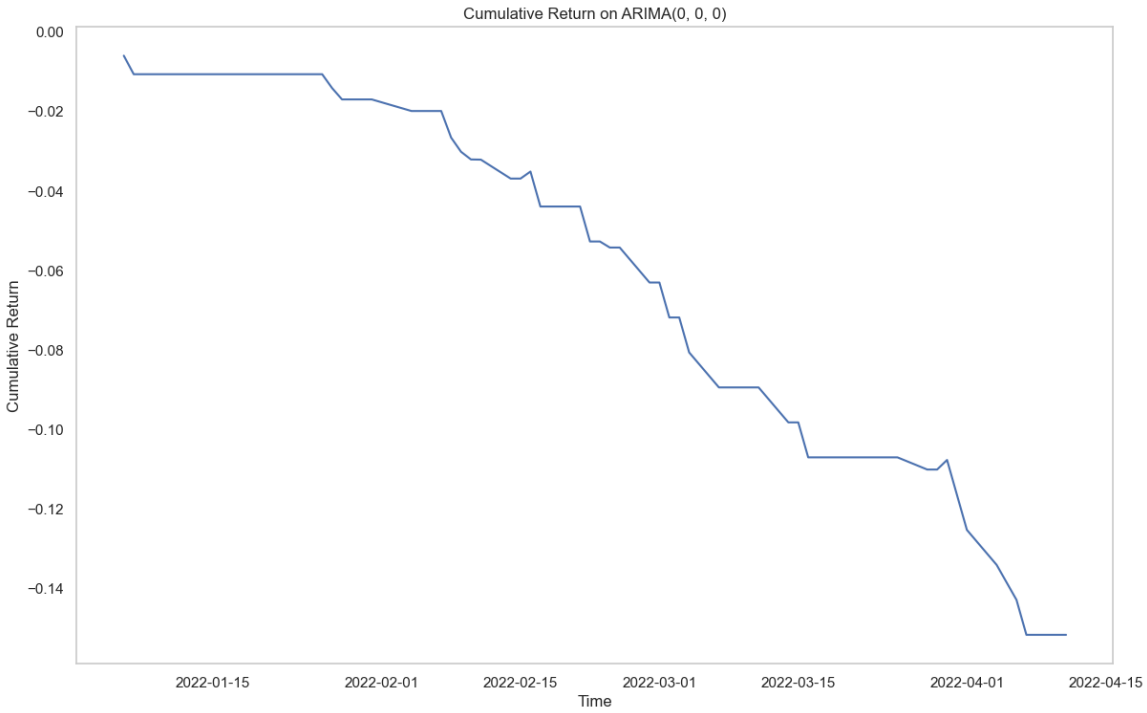


Figure 13: Null Model ARIMA(0,1,0)-Backtesting Cumulative Return

istics of the financial market and our analysis of the data shows no sign of seasonality, we will not include seasonality components in the ARIMA model we use. The model's performance is satisfying in general.

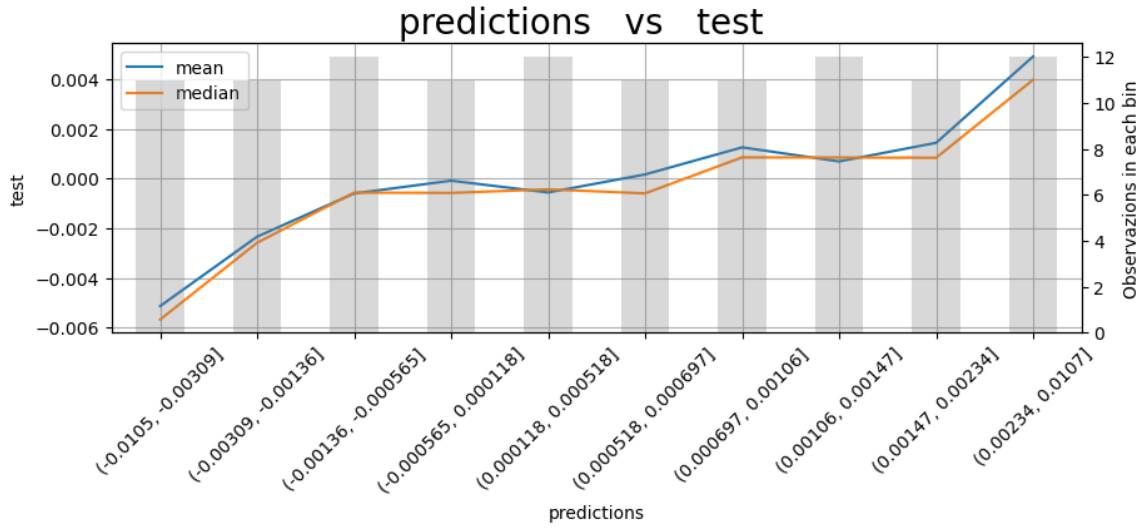


Figure 14: Model1 ARIMA(5,1,1)-Bin Plot

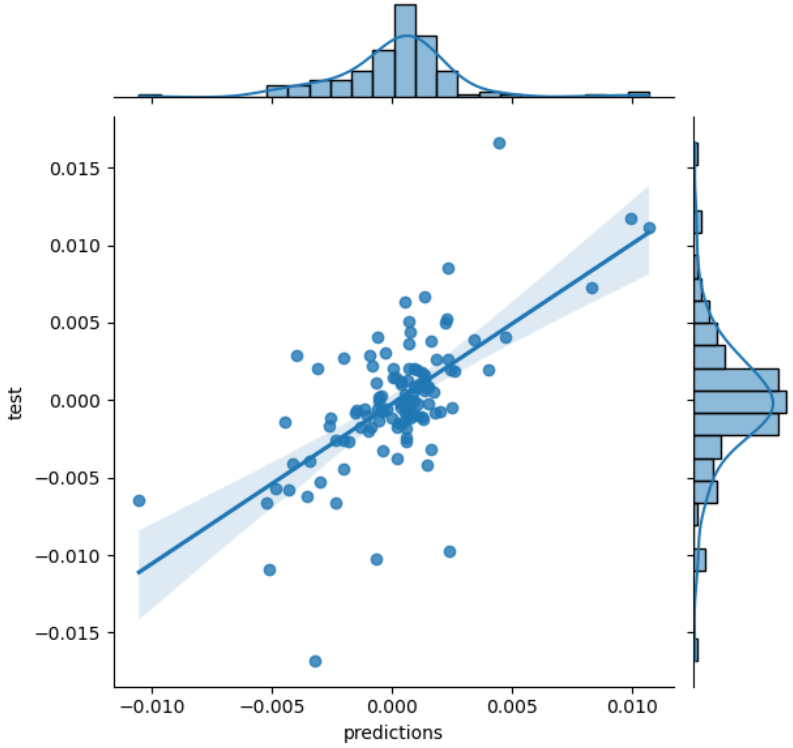


Figure 15: Model1 ARIMA(5,1,1)-Scatter Plot

	1	2	3	4	5
ljungbox stat	0.21179	2.10644	2.57267	5.11	6.20894
ljungbox p-value	0.645368	0.348813	0.462301	0.276196	0.286416

Table 3: Ljung-Box test statistics for ARIMA (5,1,1)

However, when we examine the ARIMA(5,1,1) model summaries, we see that parameters for AR terms at lag5, lag4 and lag3 are all insignificant at 5% level, indicating an overfitting problem. Therefore, more models should be tested.

ARMA Model Results

Dep. Variable:	y	No. Observations:	394
Model:	ARMA(5, 1)	Log Likelihood	1693.506
Method:	css-mle	S.D. of innovations	0.003
Date:	Tue, 26 Apr 2022	AIC	-3371.013
Time:	12:01:57	BIC	-3339.202
Sample:	0	HQIC	-3358.408

	coef	std err	z	P>|z|	[0.025	0.975]
const	-2.398e-05	1.71e-06	-14.039	0.000	-2.73e-05	-2.06e-05
ar.L1.y	0.0600	0.051	1.171	0.242	-0.040	0.161
ar.L2.y	-0.0012	0.052	-0.023	0.982	-0.102	0.100
ar.L3.y	0.0386	0.052	0.746	0.455	-0.063	0.140
ar.L4.y	0.0250	0.052	0.483	0.629	-0.076	0.126
ar.L5.y	0.0367	0.052	0.708	0.479	-0.065	0.138
ma.L1.y	-1.0000	0.009	-111.224	0.000	-1.018	-0.982

Roots

Figure 16: Model1 ARIMA(5,1,1)-Summary

3.4 Model 2 & 3: ARIMA(4,1,1) & ARIMA(3,1,1)

Given the insignificant parameter estimation for lag5, lag4 and lag3 AR terms, we try to recursively fit the ARIMA model with reduced p parameter. We first test Model2: ARIMA(4,1,1) model.

ARMA Model Results

Dep. Variable:	y	No. Observations:	394
Model:	ARMA(4, 1)	Log Likelihood	1691.781
Method:	css-mle	S.D. of innovations	0.003
Date:	Tue, 26 Apr 2022	AIC	-3369.562
Time:	12:04:53	BIC	-3341.728
Sample:	0	HQIC	-3358.533

	coef	std err	z	P>|z|	[0.025	0.975]
const	-2.543e-05	6.18e-06	-4.115	0.000	-3.75e-05	-1.33e-05
ar.L1.y	0.0596	nan	nan	nan	nan	nan
ar.L2.y	-0.0070	0.014	-0.518	0.605	-0.034	0.020
ar.L3.y	0.0415	0.026	1.583	0.113	-0.010	0.093
ar.L4.y	0.0068	0.034	0.196	0.845	-0.061	0.074
ma.L1.y	-0.9647	nan	nan	nan	nan	nan

Figure 17: Model2 ARIMA(4,1,1)-Summary

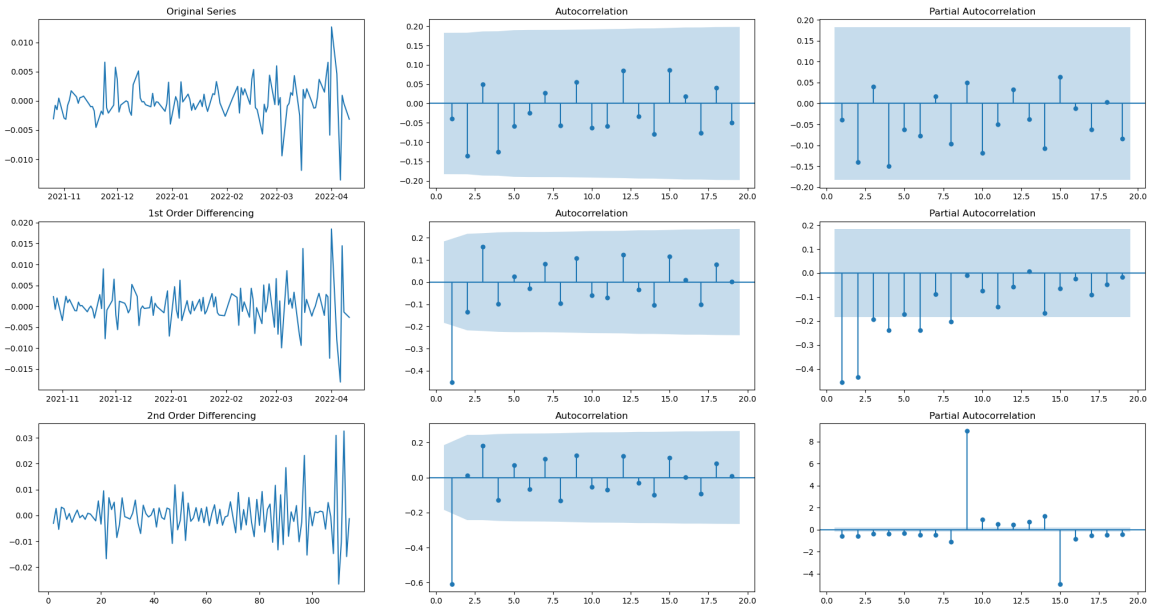


Figure 18: Model2 ARIMA(4,1,1)-Residual Analysis

As can be seen, the model is adequate while the p-value for AR4 is still high, so we further reduce the model to Model3: ARIMA(3,1,1) and examine its performance.

ARMA Model Results

Dep. Variable:	y	No. Observations:	394
Model:	ARMA(3, 1)	Log Likelihood	1691.695
Method:	css-mle	S.D. of innovations	0.003
Date:	Tue, 26 Apr 2022	AIC	-3371.390
Time:	12:06:23	BIC	-3347.532
Sample:	0	HQIC	-3361.937

	coef	std err	z	P>|z|	[0.025	0.975]
const	-2.44e-05	3.51e-06	-6.961	0.000	-3.13e-05	-1.75e-05
ar.L1.y	0.0691	0.052	1.316	0.188	-0.034	0.172
ar.L2.y	0.0376	0.053	0.715	0.475	-0.066	0.141
ar.L3.y	0.1351	0.053	2.548	0.011	0.031	0.239
ma.L1.y	-0.9890	0.024	-41.164	0.000	-1.036	-0.942

Figure 19: Model3 ARIMA(3,1,1)-ARIMA summary

Luckily, this time we can see that the model's AR3 parameter is significant at 5%

level and the model is adequate in residual analysis.

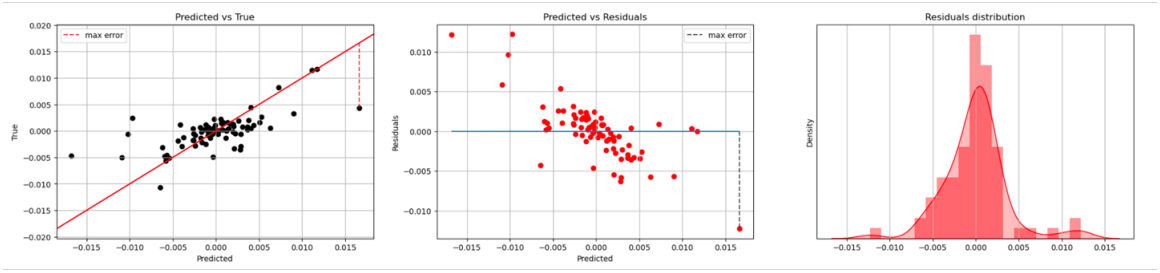


Figure 20: Model3 ARIMA(3,1,1)-Residual Analysis

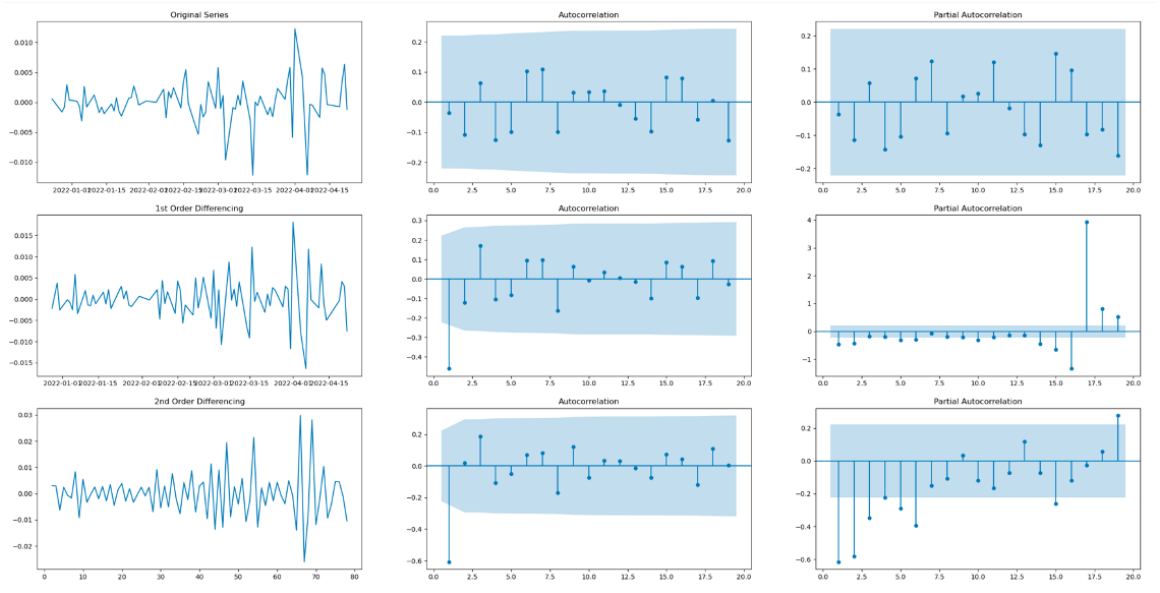


Figure 21: Model3 ARIMA(3,1,1)-Residual ACF and PACF

	1	2	3	4	5
ljungbox stat	0.160185	3.02287	3.35734	3.78048	4.05582
ljungbox p-value	0.688986	0.220593	0.339741	0.436529	0.541408

Table 4: Ljung-Box test statistics for ARIMA (3,1,1)

Prediction performance of the model is satisfying, as shown below.

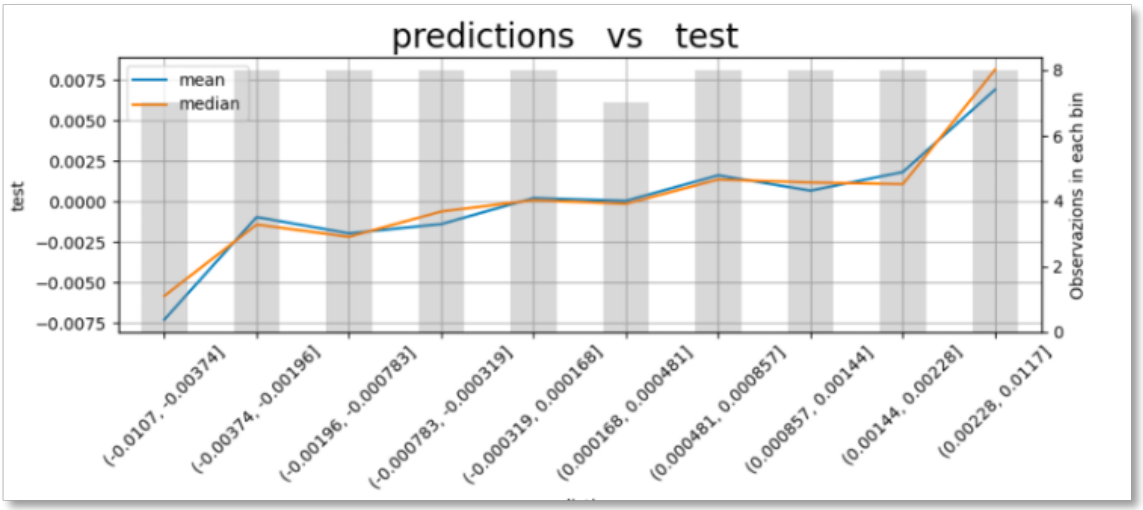


Figure 22: Model3 ARIMA(3,1,1)-Bin Plot

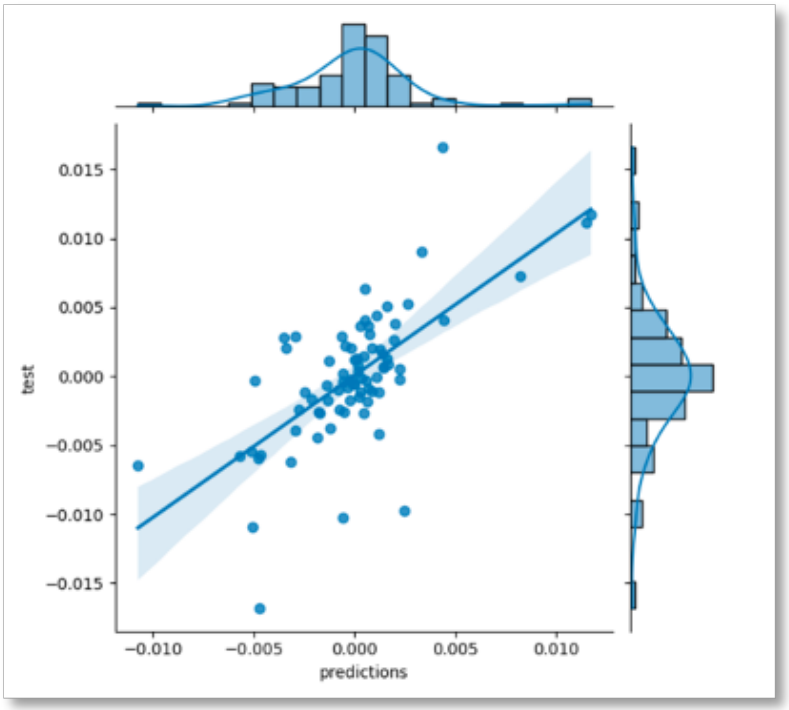


Figure 23: Model3 ARIMA(3,1,1)-Scatter Plot

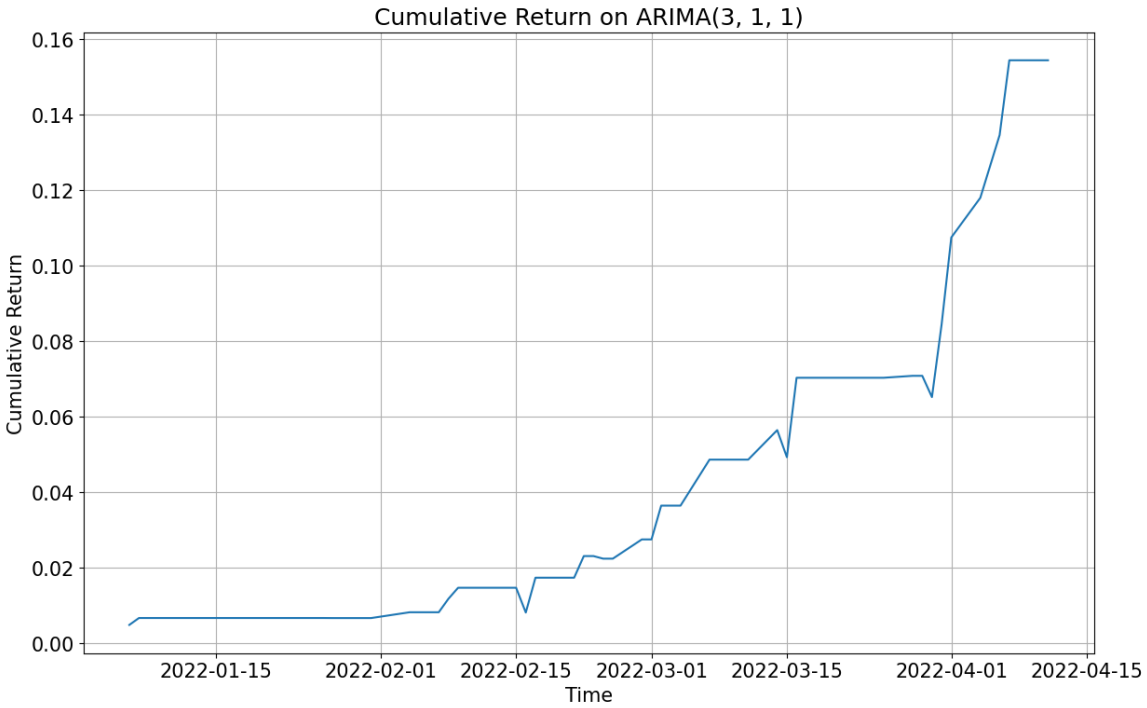


Figure 24: Model3 ARIMA(3,1,1)-Backtesting

	Parameter
correlation	0.6526
rmse	0.0032
ljung-box test(1)-p-value	0.689
Shapiro-Wilk Test p-value	0
R-squared	0.4255
Annualized Return	0.3425
Sharpe Ratio	2.4393
winrate	0.7826

Table 5: Strategy Performance for ARIMA (3,1,1)

What's worth noticing is that the residuals have a rather low Shapiro-Wilk test p-value, indicating non-normality in the residual distributions. One of the possible reasons for this is that the backtesting dates are relatively short, from 2022/01 to 2022/04, and this period is relatively special compared to other times. There're only around a hundred data points in the test set, and during this period HK market is relatively volatile, due to the delisting crisis of Chinese concept stocks in the US market and the Russo-Ukrainian war. If further log-transformation or square-transformation is applied, data-distortion may occur and the financial meaning of the original series is lost. Since the model is performing quite well itself, and changing p and q order of the model doesn't seem to improve residual normality, we believe that the non-normality in the residuals can be accepted in this case.

3.5 Model 4: Auto-ARIMA yielded model-ARIMA(1,1,1)

By using auto-arima module, it exhausts p and q in a given range and choose the optimal model based on the AIC criteria. By applying the module and restricting p and q less than 10, auto-arima suggested model of ARIMA(1,1,1). The model is also adequate and the performance of the two models is quite close.

ARMA Model Results

Dep. Variable:	y	No. Observations:	394
Model:	ARMA(1, 1)	Log Likelihood	1691.109
Method:	css-mle	S.D. of innovations	0.003
Date:	Tue, 26 Apr 2022	AIC	-3374.219
Time:	12:36:39	BIC	-3358.313
Sample:	0	HQIC	-3367.916

	coef	std err	z	P>|z|	[0.025	0.975]
const	-2.552e-05	7.04e-06	-3.626	0.000	-3.93e-05	-1.17e-05
ar.L1.y	0.0454	nan	nan	nan	nan	nan
ma.L1.y	-0.9596	nan	nan	nan	nan	nan

Figure 25: Model4 ARIMA(1,1,1)-ARIMA summary

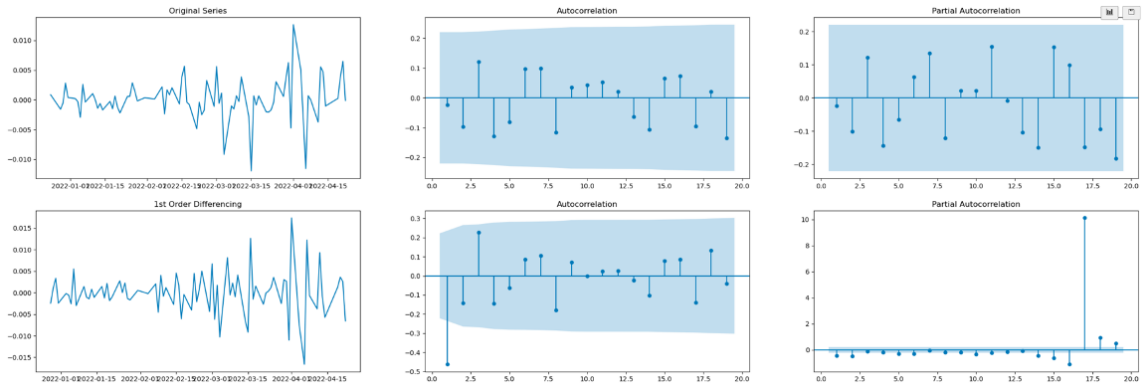


Figure 26: Model4 ARIMA(1,1,1)-Residual ACF and PACF graph

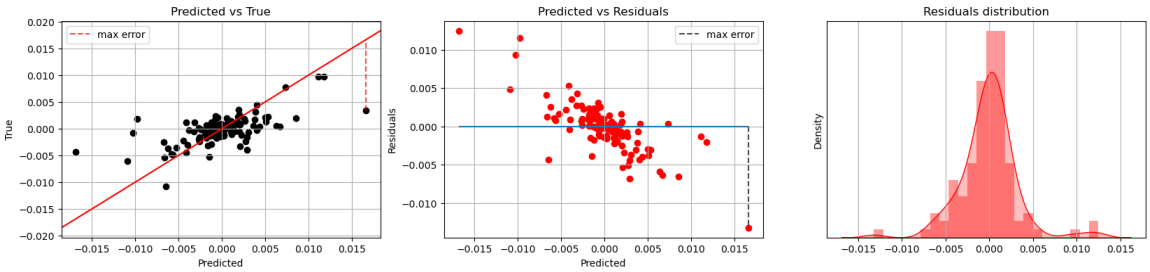


Figure 27: Model4 ARIMA(1,1,1)-Residual Distributions

	1	2	3	4	5
ljungbox stat	0.22654	1.73613	2.37615	3.1181	3.67378
ljungbox p-value	0.634101	0.419763	0.498089	0.53826	0.597271

Table 6: Ljung-Box test statistics for ARIMA (1,1,1)

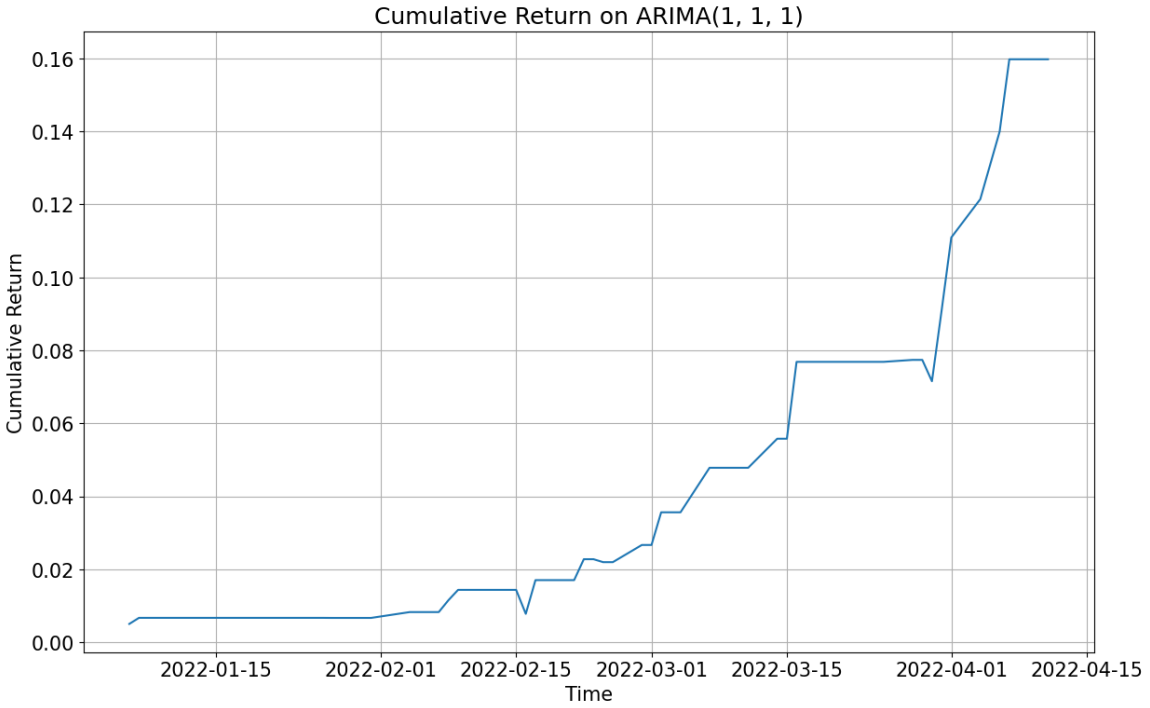


Figure 28: Model4 ARIMA(1,1,1)-Backtest

As we compare the performance of ARIMA(3,1,1) and ARIMA(1,1,1), we may see that they have quite close performance metrics. Based on AIC, we'd prefer ARIMA (1,1,1) as it has higher AIC over the entire set and has less parameters, thus may be less likely to overfit and more robust in out-of-sample data.

	Parameter
correlation	0.6526
rmse	0.0032
ljung-box test(1)-p-value	0.689
Shapiro-Wilk Test p-value	0
R-squared	0.4255
Annualized Return	0.3425
Sharpe Ratio	2.4393
winrate	0.7826

Table 7: Strategy Performance for ARIMA (1,1,1)

3.6 Model 5: Overparametarized model-ARIMA(2,1,1)

We then overparameterize ARIMA(1,1,1) and check the adequacy and level of fit for ARIMA(2,1,1).

ARMA Model Results

Dep. Variable:	y	No. Observations:	394
Model:	ARMA(2, 1)	Log Likelihood	1691.344
Method:	css-mle	S.D. of innovations	0.003
Date:	Tue, 26 Apr 2022	AIC	-3372.687
Time:	12:51:35	BIC	-3352.806
Sample:	0	HQIC	-3364.809

	coef	std err	z	P>|z|	[0.025	0.975]
const	-2.445e-05	3.52e-06	-6.945	0.000	-3.14e-05	-1.76e-05
ar.L1.y	0.0782	0.053	1.482	0.138	-0.025	0.182
ar.L2.y	0.0889	0.053	1.663	0.096	-0.016	0.194
ma.L1.y	-0.9875	0.025	-39.218	0.000	-1.037	-0.938

Figure 29: Model5 ARIMA(2,1,1)-ARIMA summary

As can be seen, the model is still adequate while the AR2 parameter is insignificant at 5% threshold. We thus conclude that ARIMA(2,1,1) is overfitted and less suitable than ARIMA(1,1,1). Similar tests are conducted for ARIMA(1,1,2) and the results are similar. **To sum up, we conclude that ARIMA(1,1,1) is our recommended model for pairs trading between HK.3033 and HK.3067.**

3.7 Predictions for suggested model ARIMA(1,1,1)

To test the performance of ARIMA(1,1,1), we plot out the prediction performance of ARIMA(1,1,1) for comparison purpose. Detailed residual analysis and prediction performance can be found in section 3.5.

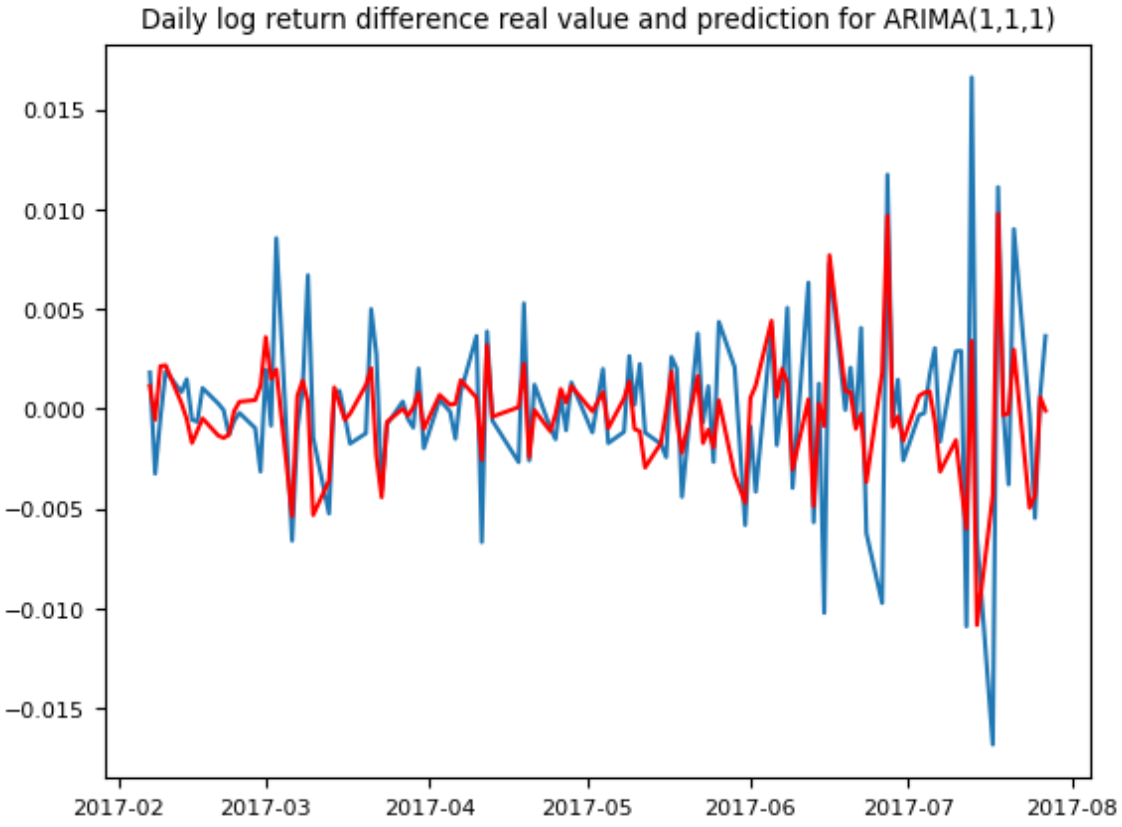


Figure 30: Suggested Model ARIMA(1,1,1)-Daily log return difference comparison

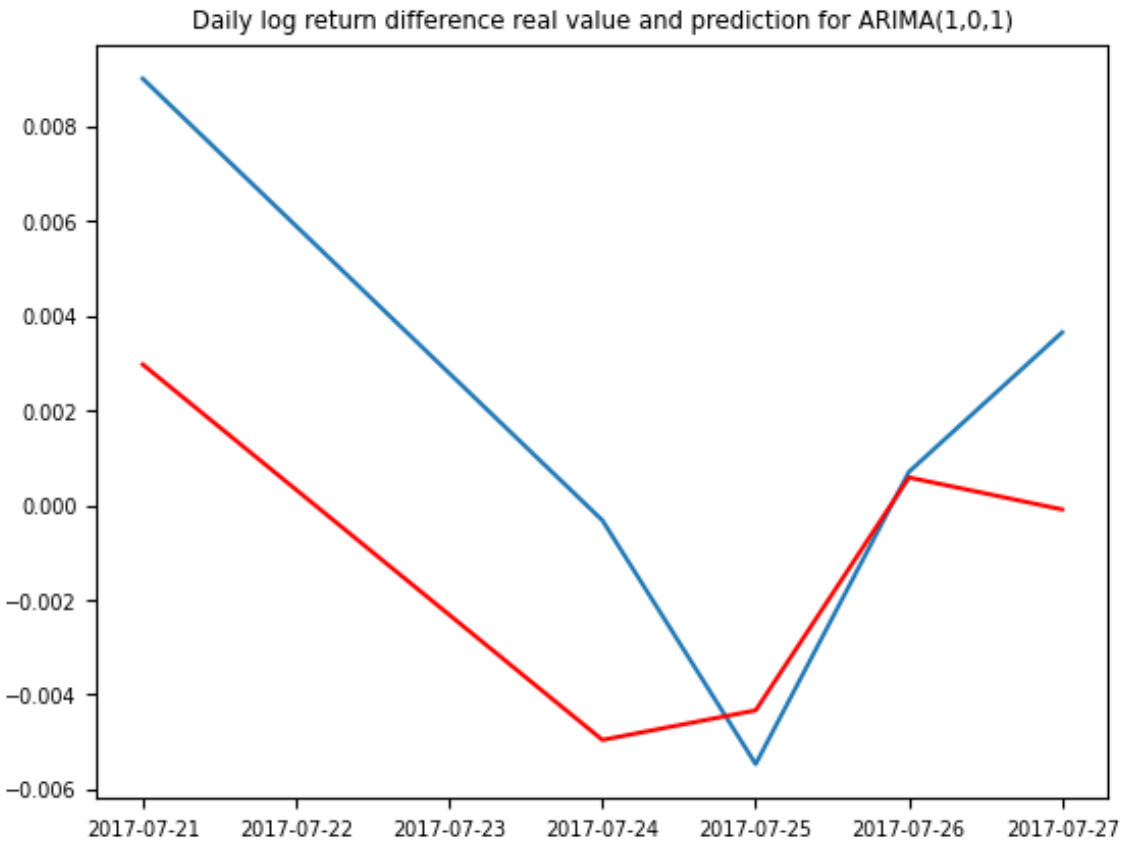


Figure 31: Suggested Model ARIMA(1,1,1)-Daily log return difference last 5 value comparison

4 Part 4: Other examined models and Conclusions

As mentioned in the first part, there're other possible opportunities for pairs trading. We examine all other models by direclty using auto-arima module, select the sub-optimal ARIMA model with highest AIC on the training set, and use them to run backtesting on the test set. In order to reflect the long-term performance of each model, we set 50% of available data for each pair to be training set and the remaining 50% of data as test set. For any pair, before each prediction is made, all known realized values up to one time point is used to train the ARIMA model and produce the forecast for the next timepoint. Below are the top-performers and their performance metrics among the 27 pairs that we filter out earlier in Part 2. All below returns take into account of 12 bps two sided transaction costs. Not every pair in the list generates steady and promising alphas, but it provides an approximate range for picking pairs trading targets in the HK market.

pairs	correlation	rmse	Shapiro-Wilk Test p-value	R-squared	Annualized Return	Sharpe Ratio
('HK.03033', 'HK.HTImain')	0.692	0.004	0.206	0.462	0.565	4.699
('HK.03088', 'HK.HHImain')	0.238	0.013	0	0.046	0.521	1.409
('HK.02828', 'HK.03088')	0.22	0.013	0	0.037	0.498	1.675
('HK.03032', 'HK.HTImain')	0.628	0.004	0.338	0.39	0.46	3.698
('HK.HHImain', 'HK.HTImain')	0.174	0.013	0	0.016	0.421	1.155
('HK.03088', 'HK.HTImain')	0.646	0.004	0.251	0.408	0.419	3.616
('HK.03033', 'HK.HHImain')	0.212	0.012	0	0.034	0.409	1.173
('HK.03032', 'HK.HHImain')	0.243	0.012	0	0.05	0.402	1.109
('HK.02828', 'HK.03033')	0.214	0.012	0	0.033	0.402	1.516
('HK.03088', 'HK.800100')	0.202	0.012	0	0.026	0.396	1.395
('HK.03033', 'HK.800100')	0.186	0.012	0	0.017	0.345	1.382
('HK.800100', 'HK.HHImain')	0.62	0.003	0	0.383	0.221	2.774
('HK.800000', 'HK.HSImain')	0.611	0.002	0	0.368	0.194	2.742

Table 8: Performance Metrics for different pairs identified

If we take a close look at the top performing pair, HK.03033 and HK.HTImain, the ETF and future tracing Hang Seng technology index, its prediction performance is satisfying in general. Others have varying results, but the list provides a rough selection range for pairs trading targets.

	1	2	3	4	5
lb_stat	2.87186	3.11787	3.31729	6.79419	6.92214
lb_p-value	0.0901408	0.21036	0.345243	0.147172	0.226496

Table 9: Ljungbox Test Statistics for residuals

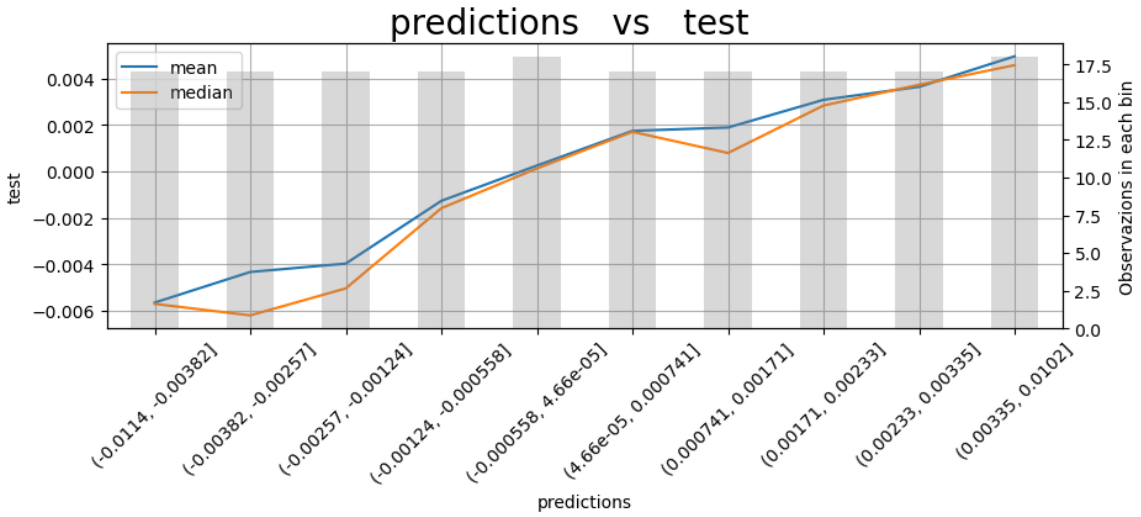


Figure 32: Auto-Arima specified model prediction and real spread-HK.03033 and HK.HTImain-Bin Plot

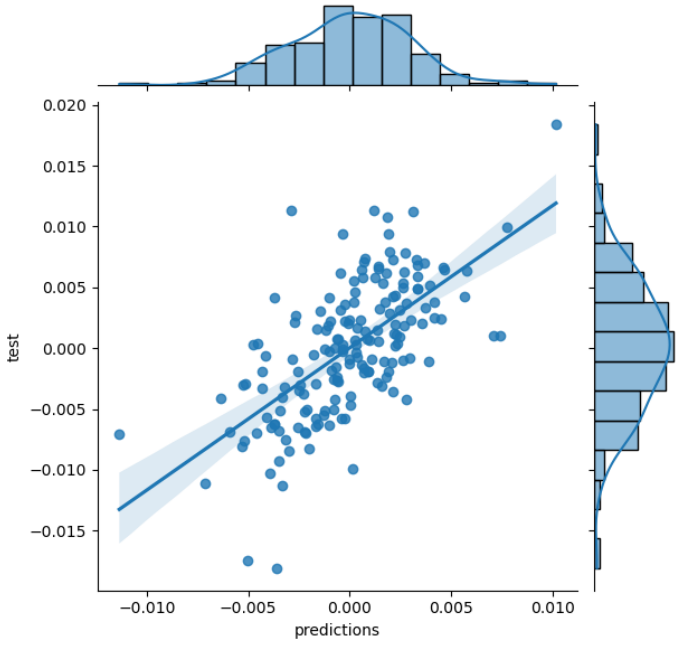


Figure 33: Auto-Arima specified model prediction and real spread-HK.03033 and HK.HTImain-Scatter Plot

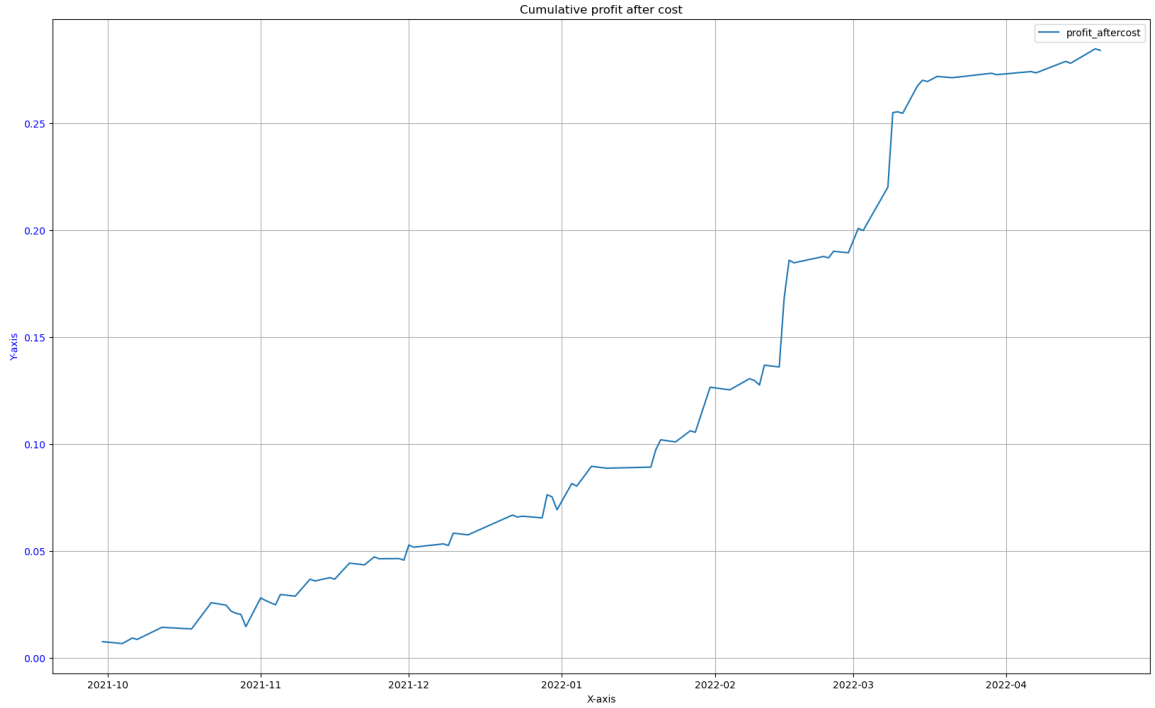


Figure 34: Auto-Arima specified model prediction and real spread-HK.03033 and HK.HTImain-Backtesting result

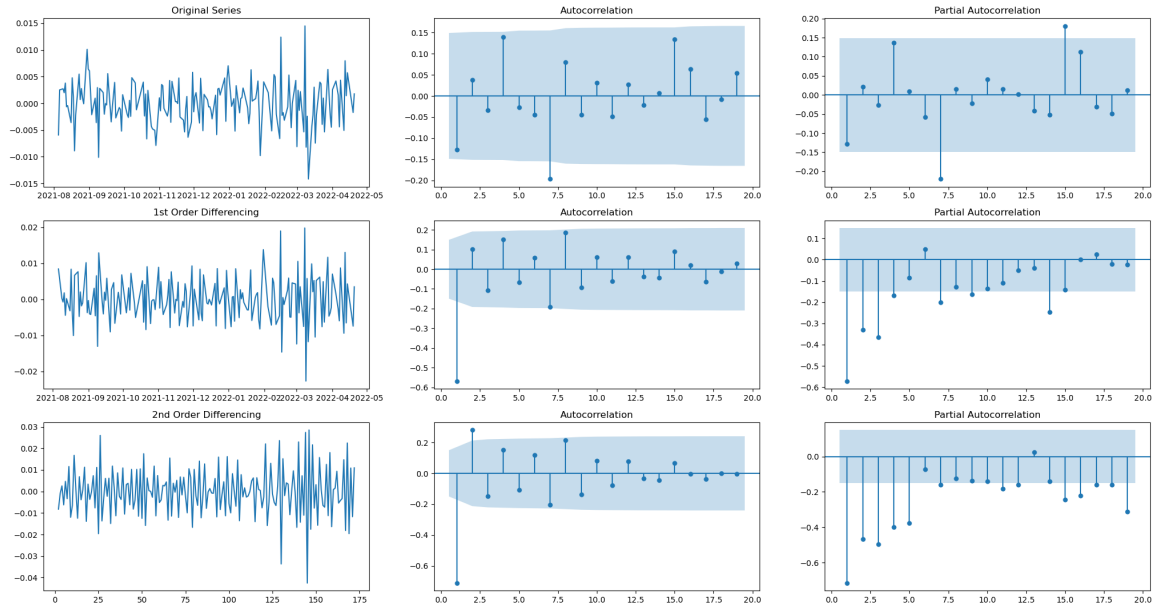


Figure 35: Auto-Arima specified model prediction and real spread-HK.03033 and HK.HTImain-Residual ACF analysis

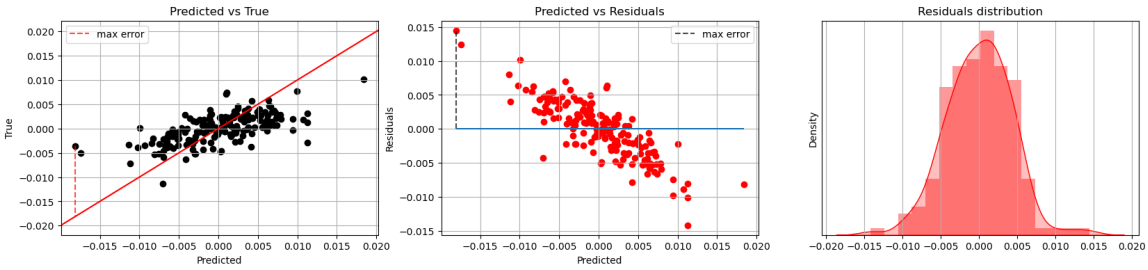


Figure 36: Auto-Arima specified model prediction and real spread-HK.03033 and HK.HTImain-Residual distribution analysis

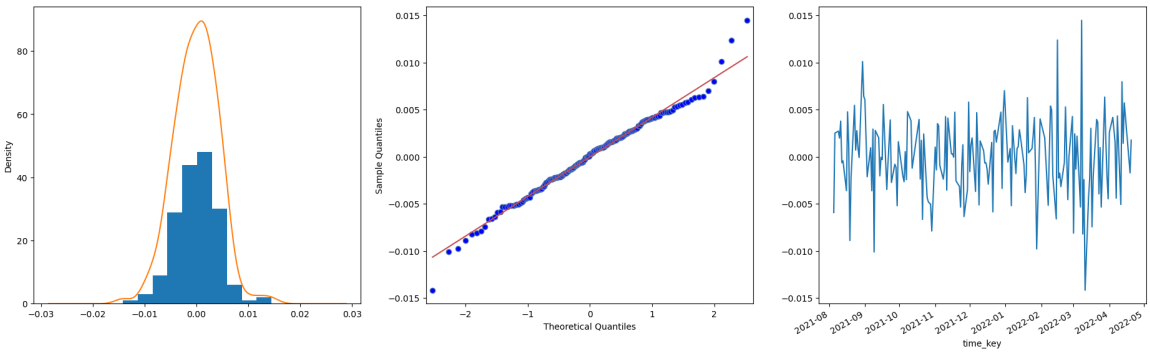


Figure 37: Auto-Arima specified model prediction and real spread-HK.03033 and HK.HTImain-Prediction Value distribution

	Parameter
correlation	0.6457
rmse	0.0042
ljung-box test(1)-p-value	0.0901
Shapiro-Wilk Test p-value	0.2512
R-squared	0.4083
Annualized Return	0.4188
Sharpe Ratio	3.6163
winrate	0.494
Anual number of trade	70.7514
average position	1.4266

Table 10: Overall Performance Metrics of pairstrading between HK.3033 and HK.HTImain

As can be seen, HK.3033 and Hang Seng Technology Index Future forms a good target pair for pairs trading, with promising, steady backtesting results and strong predictive power of future daily log return difference using ARIMA(1,0,1) model. Other pairs are also worth examining and may be taken into consideration when trying to form a pairs-trading strategy, but due to space limit, we will not elaborate more here.

4.1 Risks and Conclusions

Before implementing strategy in the reality, certain risks exists due to both limitations in our research and in pairs trading techniques themselves. Possible risks include:

- Underestimation of transaction cost: Slippage(bid-ask spread), etc. In our model we estimated two side trading costs to be 12 bps, including 7bps of transaction fees, 3bps of slippage and 2 bps of daily leverage costs. However, in times of extreme market conditions, slippage may be enlarged and may lead to deterioration of profits.

- Failure of model: In long-term irrational market conditions, the diverged spread may take a long time to converge, or may not even converge at all. The cointegration relationship may, under extreme conditions, change in the long run.

- Alpha decay: when more people notice possible profitability from statistical arbitrage, spreads would be corrected immediately, leading to lower profit or even losses in strategy implementation.

- Liquidity Risks: Some ETFs may have very low trading volume in certain days, leading to higher slippage and bid-ask spread, eroding profit of the strategy.

In the process of the research, the following findings are revealing and may be of value for further research. First of all, market inefficiency rises as market become more volatile, possibly due to the increase of noise trading and rush-selloff trading. As can be seen from the previous backtesing graphs, the strategy performs especially well during 2022 February and March, when the Russo-Ukrainian War broke out and doubts about the investment grade of Chinese concept stocks shook the market due to the possible delisting of these stocks in the US market. It may be proper to shift more investment focus onto statistical arbitrage strategies during times of market volatility. In addition to that, our research provides a rough framework for examination of pair trading effectiveness and a range of targets with profitability potential in the HK market. Though, as stated, the backtesting period may be more special compared to other times when the market is more calm and efficient, and there're also endogenous liquidity risks and slippage in real life application, leading to possible upward biased profitability backtesting result. Further research can be conducted in examination of real-life practicality of the strategy and optimization of time series model by adding in additional factors such as sentiment factors technical indicators. In general, the HK market is still relatively inefficient compared to the US market in many ways, and

more opportunities may be dug up in the related field with more effort and possibly higher rewards.

STAT4601_GP extended

May 4, 2022

[ ]:
```
from Utils import *
from ml_utils import *
from Futu_Utils import * #Utils, ml_utils and Futu_utils are three libraries I␣
↪wrote myself. Can be shared if required.
from statsmodels.graphics.gofplots import qqplot
from scipy.stats import shapiro
import numpy as np, pandas as pd
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib.pyplot as plt
# from pandas import read_csv
# from pandas import datetime
from matplotlib import pyplot
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
from math import sqrt
from statsmodels.stats.diagnostic import acorr_ljungbox
import yfinance as yf
import statsmodels.api as sm
from statsmodels.tsa.stattools import coint, adfuller
import seaborn as sns; sns.set(style="whitegrid")
```

[ ]:
```
target_list=['^HSI','2800.HK','7500.HK','3067.HK','3033.HK','3032.HK','7226.
↪HK','7200.HK','^HSCE','2828.HK','7552.HK','7226.HK','3088.HK']
target_list=list(set(target_list))
target_list_futu=list(pd.Series(target_list).apply(lambda x:'HK.0'+x[:-3] if␣
↪x[-2:]=='HK' else x))
target_list.sort()
target_list_futu=['HK.800000' if x=='^HSI' else 'HK.02828' if x=='^HSCE' else x␣
↪for x in target_list_futu]
target_list_futu.append('HK.HSImain')
target_list_futu.append('HK.HTImain')
target_list_futu.append('HK.HHImain')
target_list_futu.append('HK.800100')
target_list_futu=list(set(target_list_futu))
```

[ ]:
```
df_futu=get_history(target_list_futu,start='2015-01-01',end='2022-04-20',frequency='day')
df_futu_original=df_futu.set_index(['time_key','code']).unstack(1)
```

```
df_futu=df_futu_original['close'].copy()
```

```
2022-05-02 23:55:51,932 | 12343 | [open_context_base.py]
_send_init_connect_sync:308: InitConnect ok: conn_id=5, host=127.0.0.1,
port=11111, user_id=11959962
2022-05-02 23:55:55,554 | 12343 | [open_context_base.py]
on_disconnect:380: Disconnected: conn_id=5
request successfully
```

0.1 if you want to test run the code yourself you may start from here

[ ]:
```
df1=yf.download(target_list,period='10y',auto_adjust=True,threads=True)['Close']
df2=pd.read_parquet('Data/Daily Data/ETF daily.parquet')
df3=df_futu.copy()
```

```
[**********100%**********]  12 of 12 completed
```

[ ]:
```
#Choose the data source from futu
df=df3.copy()
data=df3.copy()
```

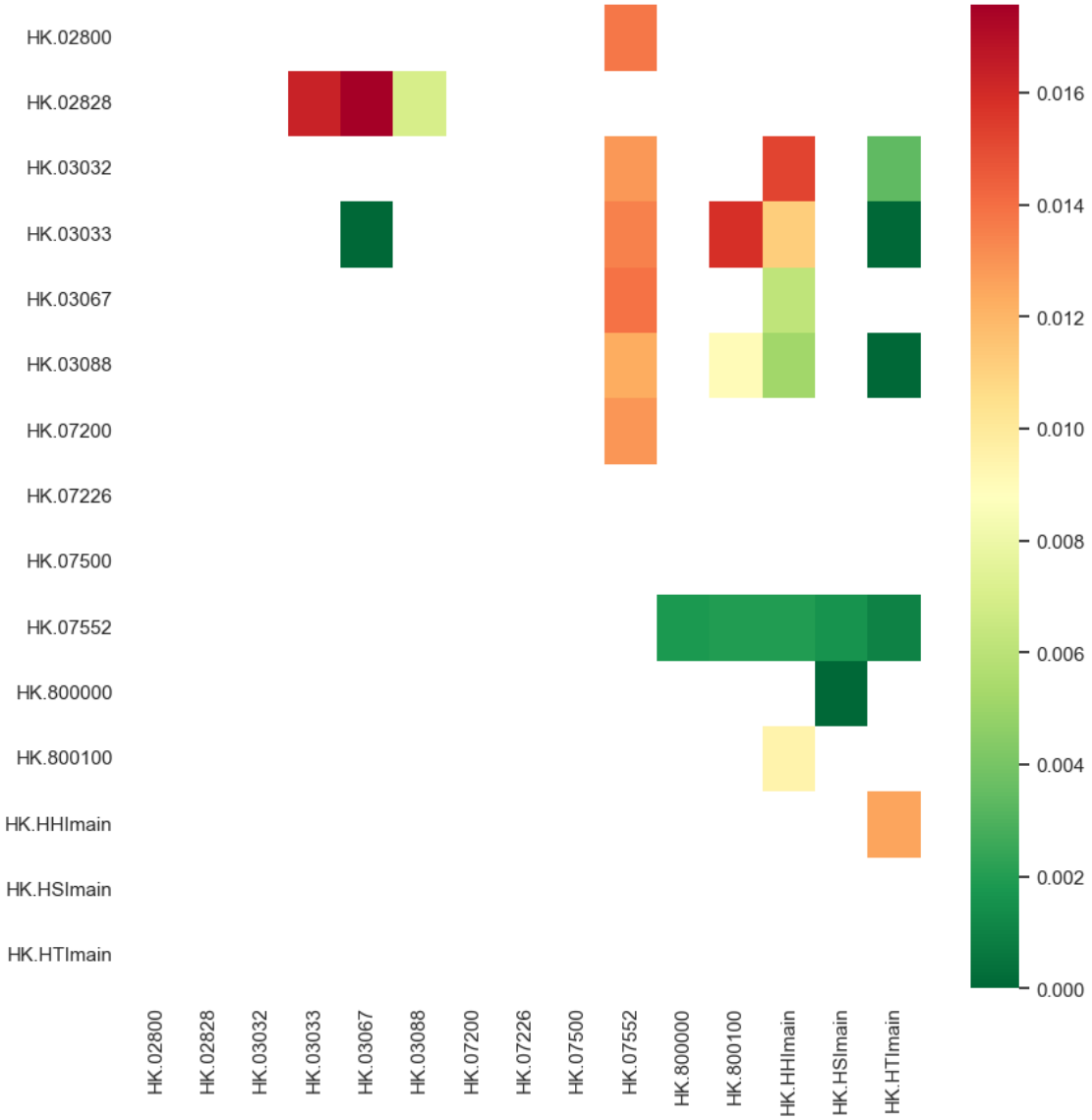
0.2 Correlation and Preliminary Data Screening

[ ]:
```
# df_original.to_csv('10 year Data.csv')
```

[ ]:
```
# Heatmap to show the p-values of the cointegration test between each pair of
# stocks. Only show the value in the upper-diagonal of the heatmap
def find_cointegrated_pairs(data,pvalue_thres):
        n = data.shape[1]
        score_matrix = np.zeros((n, n))
        pvalue_matrix = np.ones((n, n))
        keys = data.keys()
        pairs = []
        for i in range(n):
            for j in range(i+1, n):
                S1 = data[[keys[i],keys[j]]].dropna()[keys[i]]
                S2 = data[[keys[i],keys[j]]].dropna()[keys[j]]
                result = coint(S1, S2)
                score = result[0]
                pvalue = result[1]
                score_matrix[i, j] = score
                pvalue_matrix[i, j] = pvalue
                if pvalue < pvalue_thres:
                    pairs.append((keys[i], keys[j]))
        return score_matrix, pvalue_matrix, pairs
def display_cointegrated_pairs(df,pvalue=0.02):
```

```
    scores, pvalues, pairs = find_cointegrated_pairs(df,pvalue)
    fig, ax = plt.subplots(figsize=(10,10))
    sns.heatmap(pvalues, xticklabels=list(df.columns), yticklabels=list(df.
↪columns), cmap='RdYlGn_r'
                    , mask = (pvalues >= pvalue),ax=ax)
    plt.show()
    return pairs,pd.DataFrame(pvalues,index=df.columns,columns=df.columns)
```

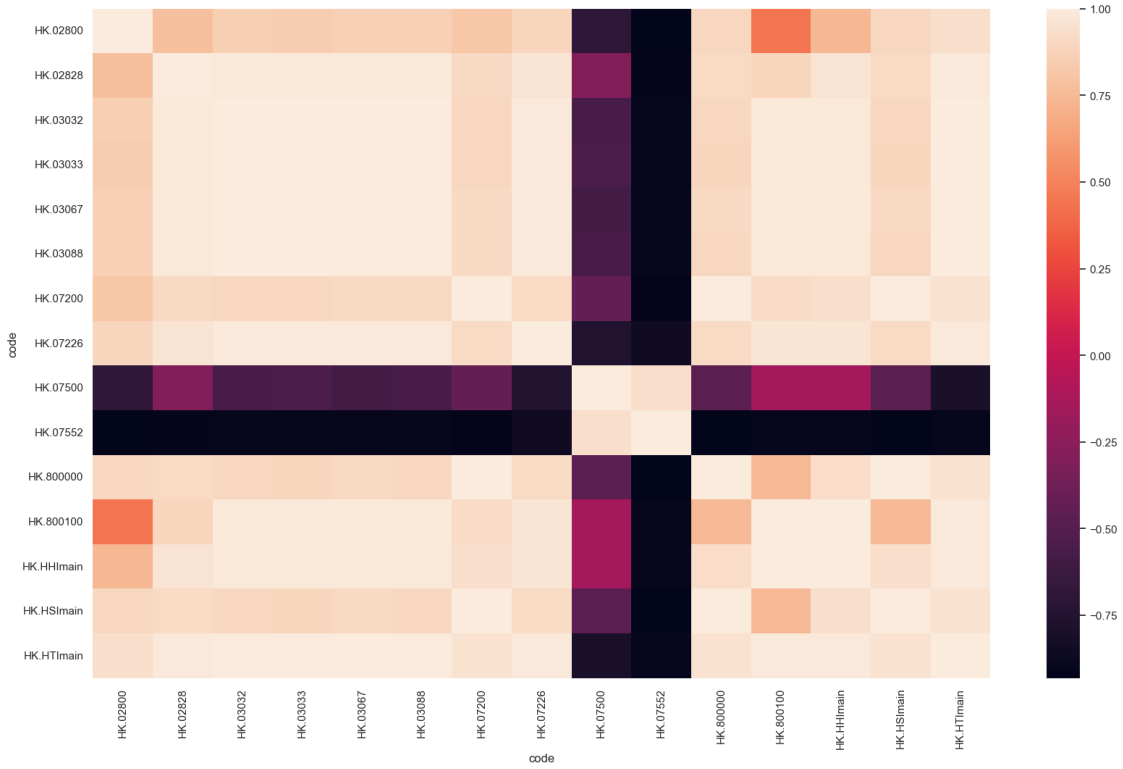
[ ]:
```
pairs,pvalues=display_cointegrated_pairs(df.loc[df.index.year>2017])
```



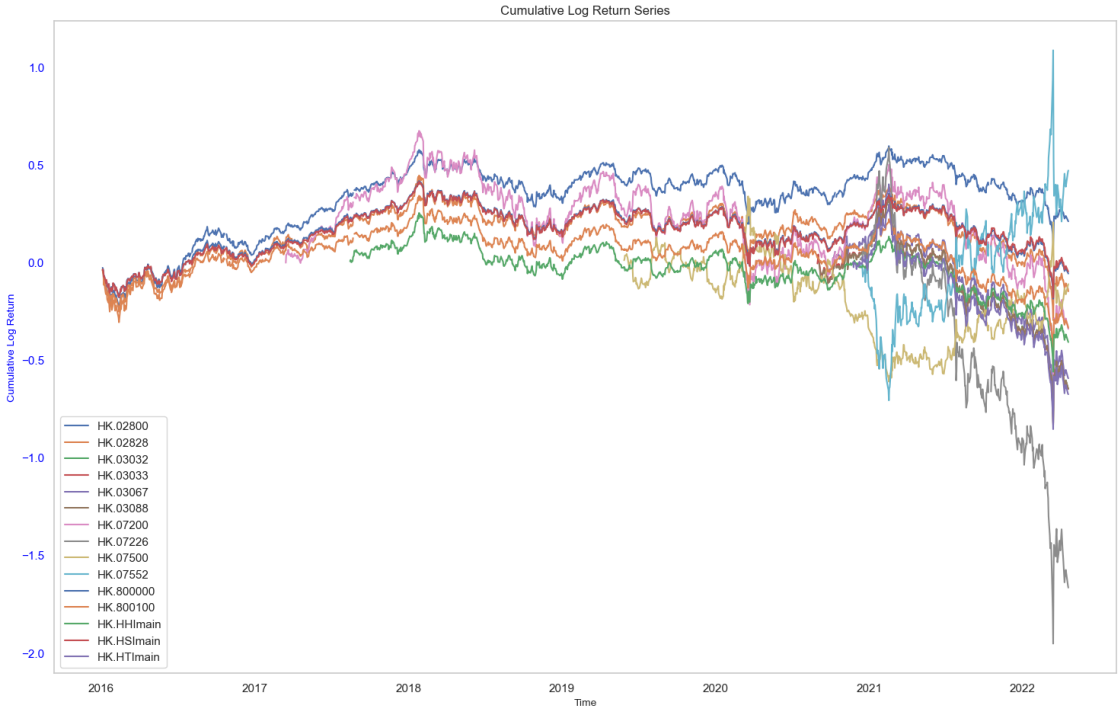
[ ]:
```
pairs_list=pairs.copy()
```

[ ]:
```
#check correlation
corr_df=df.corr()
log_ret=df.apply(lambda x:np.log(x/x.shift(1)))
log_ret=log_ret.loc[log_ret.index.year>2015]
plt.figure(figsize=(20,12))
sns.heatmap(corr_df)
```

[ ]:
```
<AxesSubplot:xlabel='code', ylabel='code'>
```



[ ]:
```
#check Cumulative Log return
plot_line(log_ret.cumsum(),title='Cumulative Log Return␣
↪Series',xlabel='Time',ylabel='Cumulative Log Return',figsize=(20*0.9,20*0.
↪618*0.9))
```



[ ]:
```
p_value_rank=pd.DataFrame()
for name1,name2 in pairs_list:
    p=pvalues.loc[name1][name2]
    p_value_rank=pd.concat([p_value_rank,pd.DataFrame([name1,name2,p])],axis=1)
p_value_rank=p_value_rank.T
p_value_rank.sort_values(2,inplace=True)
p_value_rank.columns=['name1','name2','p']
p_value_rank.head(5)
```

[ ]:

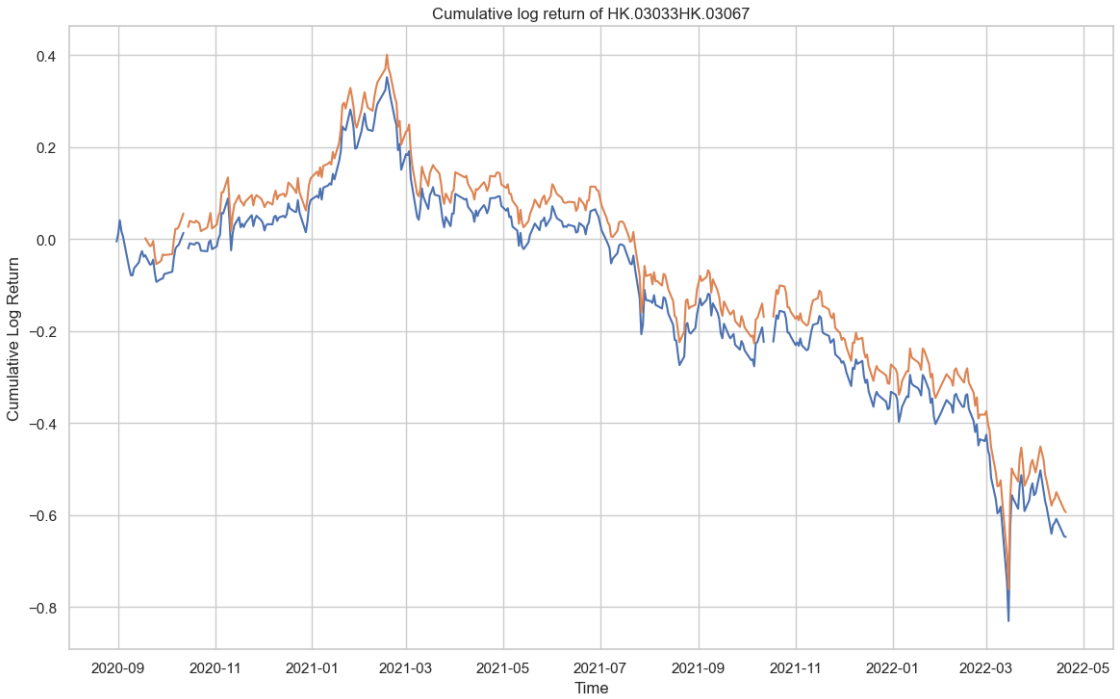
	name1	name2	p
0	HK.03033	HK.HTImain	0.0
0	HK.03033	HK.03067	0.0
0	HK.800000	HK.HSImain	0.0
0	HK.03088	HK.HTImain	0.0
0	HK.07552	HK.HTImain	0.000968

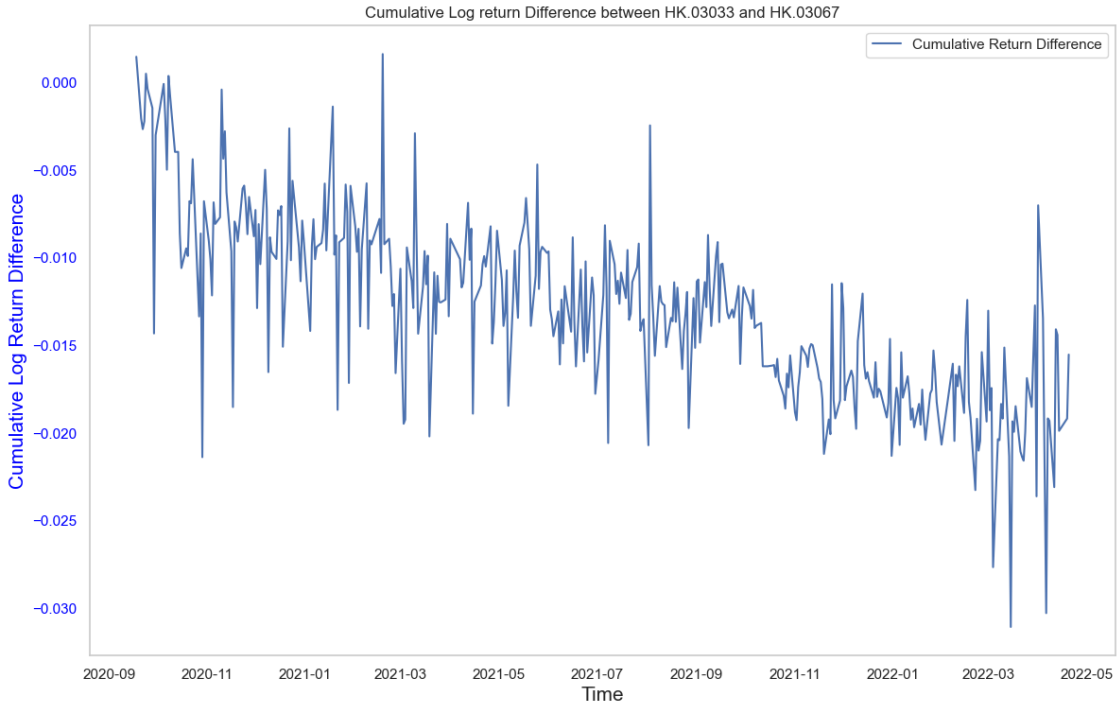
0.3 Here we choose the pair HK.3033 and HK.3067

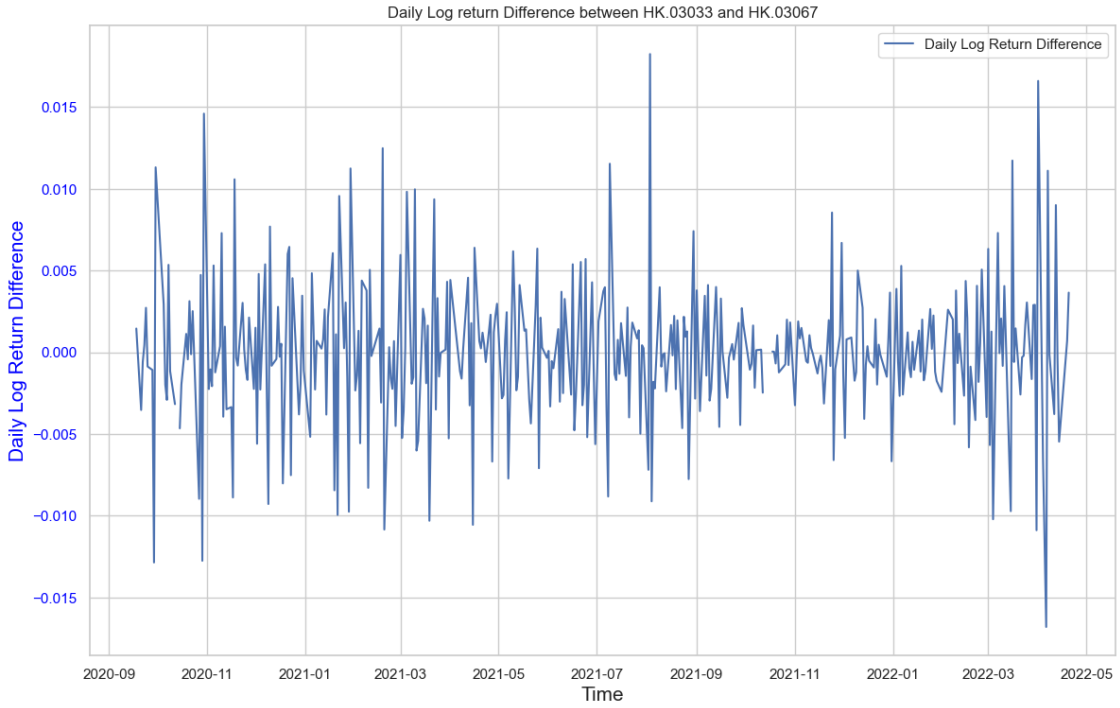
[ ]:
```
name1,name2=('HK.03033', 'HK.03067')
target_series1=log_ret[name1]
target_series2=log_ret[name2]
temp=log_ret[[name1,name2]]
diff=target_series1-target_series2
cum_diff=diff.cumsum()
```

```
transcost=3/10000
```

[ ]:
```
sns.set_style('whitegrid')
plt.rcParams.update({'font.size': 15})
plt.figure(figsize=(20*0.7,12*0.7))

# plt.title('Cumulative Log Return Series of 3033.HK and 3067.HK')
plt.plot(temp.cumsum())
plt.title('Cumulative log return of '+name1+name2)
plt.xlabel('Time')
plt.ylabel('Cumulative Log Return')
# plt.grid()
cum_diff=(temp.iloc[:,0]-temp.iloc[:,1]).cumsum().fillna(method='ffill')
plt.rcParams.update({'font.size': 15})
# plt.figure(figsize=(20*0.7,20*0.618*0.7))
plot_line(cum_diff.rename('Cumulative Return Difference'),title='Cumulative Log␣
 ↪return Difference between '+name1+' and␣
 ↪'+name2,xlabel='Time',ylabel='Cumulative Log Return␣
 ↪Difference',figsize=(20*0.7,20*0.618*0.7),fontsize=15)
log_ret_1=(temp.iloc[:,0]-temp.iloc[:,1])
plt.rcParams.update({'font.size': 15})
# plt.figure(figsize=(20*0.7,20*0.618*0.7))
plot_line(log_ret_1.rename('Daily Log Return Difference'),title='Daily Log␣
 ↪return Difference between '+name1+' and '+name2,xlabel='Time',ylabel='Daily␣
 ↪Log Return Difference',figsize=(20*0.7,20*0.618*0.7),fontsize=15,grid=False)
```



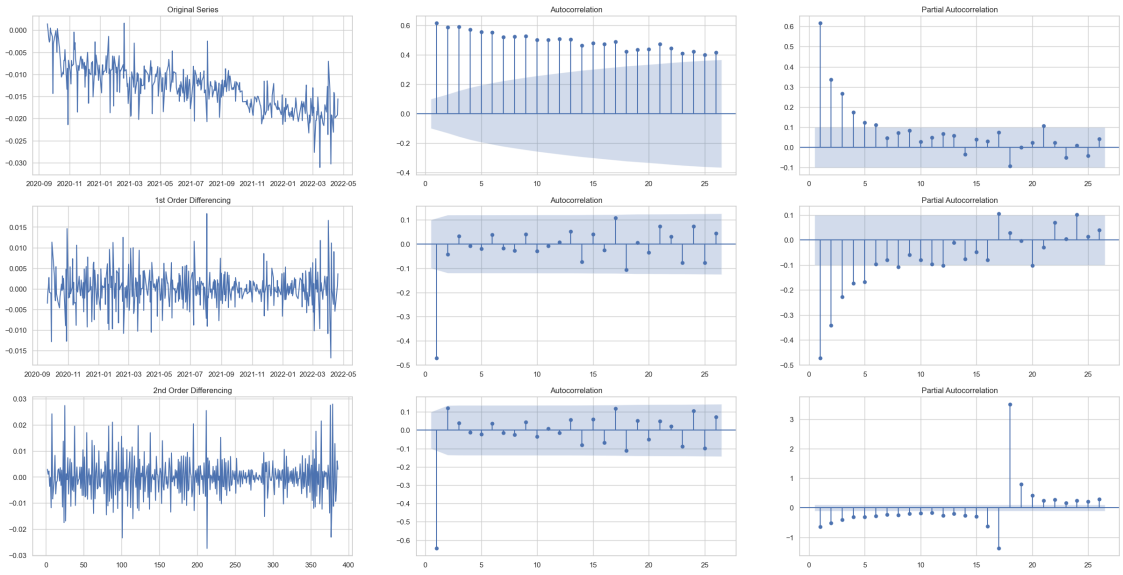

Cumulative Log return Difference between HK.03033 and HK.03067
Cumulative Return Difference
Cumulative Log Return Difference
0.000
−0.005
−0.010
−0.015
−0.020
−0.025
−0.030
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07
2021-09
2021-11
2022-01
2022-03
2022-05
Time


Daily Log return Difference between HK.03033 and HK.03067
Daily Log Return Difference
Daily Log Return Difference
0.015
0.010
0.005
0.000
−0.005
−0.010
−0.015
2020-09
2020-11
2021-01
2021-03
2021-05
2021-07
2021-09
2021-11
2022-01
2022-03
2022-05
Time

[ ]:
```
def resid_analysis(resid,lags=20):
    from scipy.stats import shapiro
    df = resid.dropna()
    fig, axes = plt.subplots(3, 3,figsize=(25,13))
    axes[0, 0].plot(df); axes[0, 0].set_title('Original Series')
    plot_acf(df.dropna().values, ax=axes[0, 1],zero=False,lags=np.
↪arange(1,lags))
    plot_pacf(df.dropna().values, ax=axes[0, 2],zero=False,lags=np.
↪arange(1,lags))

    axes[1, 0].plot(df.diff()); axes[1, 0].set_title('1st Order Differencing')
    plot_acf(df.diff().dropna().values, ax=axes[1, 1],zero=False,lags=np.
↪arange(1,lags))
    plot_pacf(df.diff().dropna(), ax=axes[1, 2],zero=False,lags=np.
↪arange(1,lags))

    # 2nd Differencing
    axes[2, 0].plot(df.diff().diff().values); axes[2, 0].set_title('2nd Order␣
↪Differencing')
    plot_acf(df.diff().diff().dropna().values, ax=axes[2, 1],zero=False,lags=np.
↪arange(1,lags))
    plot_pacf(df.diff().diff().dropna(), ax=axes[2, 2],zero=False,lags=np.
↪arange(1,lags))
    plt.show()
    fig, axes = plt.subplots(1, 3,figsize=(23,7))
    resid.hist(ax=axes[0])
    resid.plot.kde(ax=axes[0])
    qqplot(resid.dropna(),line='s',ax=axes[1])
    resid.plot(ax=axes[2])
    plt.show()
    print('Shapiro-Wilk test result: '+str(shapiro(resid.dropna()).statistic)+'␣
↪/  '+str(shapiro(resid.dropna()).pvalue))
```

[ ]:
```
import numpy as np, pandas as pd
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib.pyplot as plt

plt.rcParams.update({'figure.figsize':(15,9)})
plt.rcParams.update({'font.size':10})
# Import data
df = cum_diff.dropna()
# Original Series
fig, axes = plt.subplots(3, 3,figsize=(30,15))
axes[0, 0].plot(df); axes[0, 0].set_title('Original Series')
plot_acf(df.dropna().values, ax=axes[0, 1],zero=False)
```

```
plot_pacf(df.dropna().values, ax=axes[0, 2],zero=False)
# 1st Differencing
axes[1, 0].plot(df.diff()); axes[1, 0].set_title('1st Order Differencing')
plot_acf(df.diff().dropna().values, ax=axes[1, 1],zero=False)
plot_pacf(df.diff().dropna(), ax=axes[1, 2],zero=False)
# 2nd Differencing
axes[2, 0].plot(df.diff().diff().values); axes[2, 0].set_title('2nd Order␣
 ↪Differencing')
plot_acf(df.diff().diff().dropna().values, ax=axes[2, 1],zero=False)
plot_pacf(df.diff().diff().dropna(), ax=axes[2, 2],zero=False)
plt.show()
```



[ ]:
```
from statsmodels.tsa.stattools import adfuller
print('adfuller value:')
print(adfuller(cum_diff.dropna())[0])
print('adfuller pvalue:')
print(adfuller(cum_diff.dropna())[1])
#1st order diff adfuller
from statsmodels.tsa.stattools import adfuller
print('1st difference adfuller value:')
print(adfuller(cum_diff.diff().dropna())[0])
print('1st difference adfuller pvalue:')
print(adfuller(cum_diff.diff().dropna())[1])
```

```
adfuller value:
-1.791472216562695
adfuller pvalue:
0.3847036451519554
```

```
1st difference adfuller value:
-9.058435184037064
1st difference adfuller pvalue:
4.632975078643397e-15
```

[ ]:
```
#generate latex table for adfuller test
import latextable
from tabulate import tabulate
from texttable import Texttable
display_adf_df=pd.DataFrame([['Cumulative Log Return␣
↪Difference',adfuller(cum_diff.dropna())[0],adfuller(cum_diff.
↪dropna())[1]],['1st order difference',adfuller(cum_diff.diff().
↪dropna())[0],adfuller(cum_diff.diff().dropna())[1]]],columns=['Series␣
↪Name','ADF value','ADF p-value']).set_index('Series Name')
display_adf_df.index.name=None
def gen_latex(display_adf_df):
    table = Texttable()
    table.set_cols_align(["c"] * display_adf_df.shape[1])
    table.set_deco(Texttable.HEADER | Texttable.VLINES)
    table.add_rows(display_adf_df.values)
    print(tabulate(display_adf_df, headers='keys', tablefmt='latex'))
```

[ ]:
```
gen_latex(display_adf_df)
```

```
\begin{tabular}{lrr}
\hline
                                  &   ADF value &   ADF p-value \\
\hline
 Cumulative Log Return Difference &    -1.79147 &   0.384704    \\
 1st order difference             &    -9.05844 &   4.63298e-15 \\
\hline
\end{tabular}
```

0.4 Model Examination

[ ]:
```
name1,name2=('HK.03033', 'HK.03067')
target_series1=log_ret[name1]
target_series2=log_ret[name2]
temp=log_ret[[name1,name2]]
diff=target_series1-target_series2
cum_diff=diff.cumsum()
transcost=3/10000
```

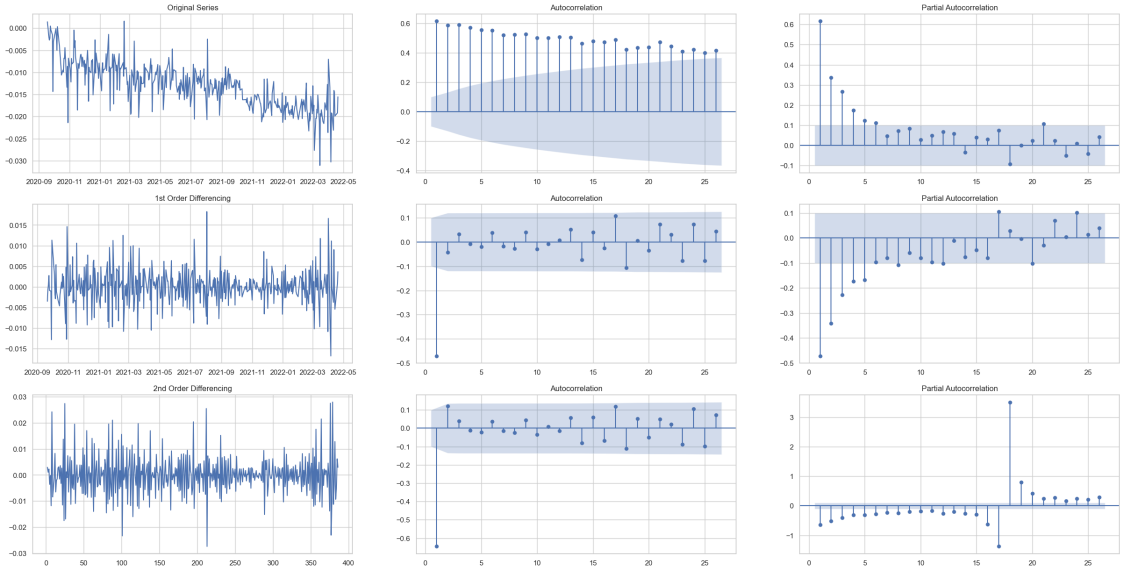
0.5 Examine the selected pair 3033.HK and 3067.HK

0.5.1 calculate all possible model

[ ]:
```
target_result_dic_total={}
(name1,name2)
```

[ ]:
```
('HK.03033', 'HK.03067')
```

[ ]:
```
plt.rcParams.update({'figure.figsize':(15,9)})
plt.rcParams.update({'font.size':10})
# Import data
df = cum_diff.dropna()
# Original Series
fig, axes = plt.subplots(3, 3,figsize=(30,15))
axes[0, 0].plot(df); axes[0, 0].set_title('Original Series')
plot_acf(df.dropna().values, ax=axes[0, 1],zero=False)
plot_pacf(df.dropna().values, ax=axes[0, 2],zero=False)
# 1st Differencing
axes[1, 0].plot(df.diff()); axes[1, 0].set_title('1st Order Differencing')
plot_acf(df.diff().dropna().values, ax=axes[1, 1],zero=False)
plot_pacf(df.diff().dropna(), ax=axes[1, 2],zero=False)
# 2nd Differencing
axes[2, 0].plot(df.diff().diff().values); axes[2, 0].set_title('2nd Order␣
↪Differencing')
plot_acf(df.diff().diff().dropna().values, ax=axes[2, 1],zero=False)
plot_pacf(df.diff().diff().dropna(), ax=axes[2, 2],zero=False)
plt.show()
```



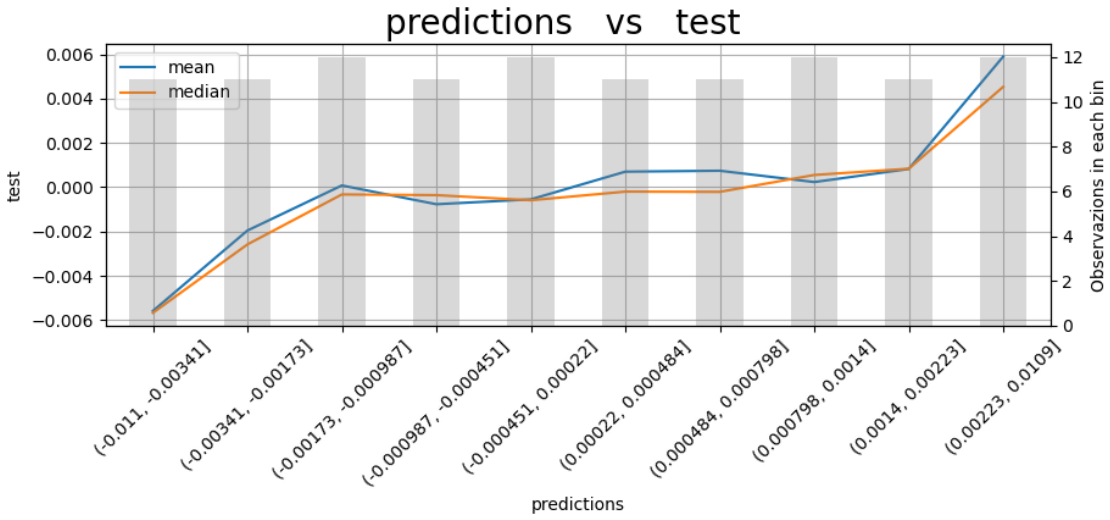
0.6 Model 1: Null Model-Testing ARIMA (0,0,0)

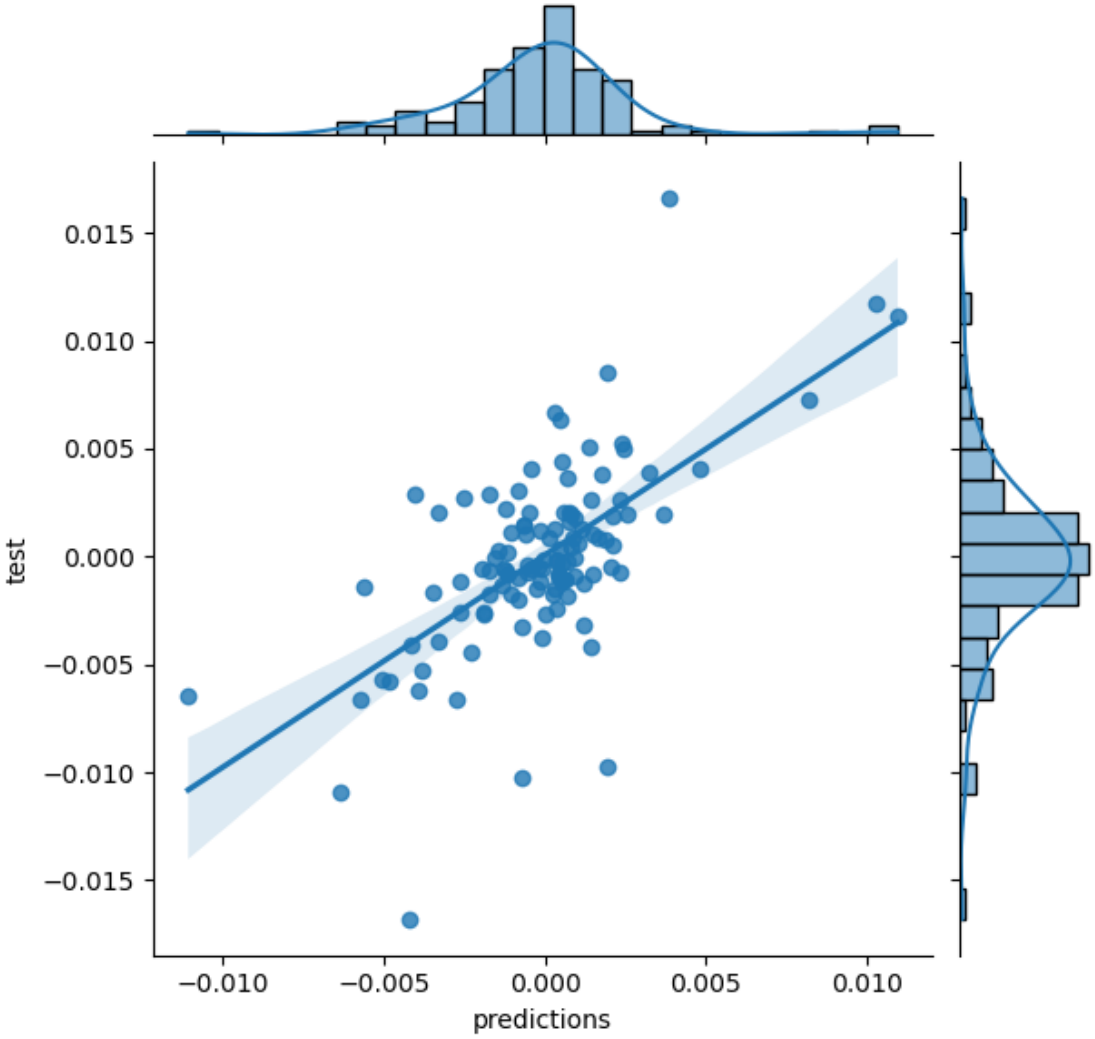
[ ]:
```
params=(1,0,2)
display((name1,name2))
display(params)
period=50
diff=(np.log(data[name1]/data[name1].shift(1))-np.log(data[name2]/data[name2].
 ↪shift(1))).dropna()
diff=diff.loc[(diff.index.year!=2020) | (diff.index.month!=3)]
diff=diff.loc[diff.index.year>2016].iloc[:-5]
series= diff.dropna().reset_index(drop=True)
X = series.values
size = int(len(X) * 0.7)
if size<100:
    size = int(len(X) * 0.5)
train, test = X[0:size], X[size:len(X)]
history = [x for x in train]
predictions = list()
# walk-forward validation
count=0

print('params: '+str(params))
if params not in target_result_dic.keys():
    for t in range(len(test)):
        count+=1
        print("{:2%}".format(count/len(test)))
        model_fit = ARIMA(history,order=params).fit(disp=0)
        yhat = model_fit.forecast()[0]
        predictions.append(yhat[0])
        obs = test[t]
        history.append(obs)
        clear_output(wait=False)
    print((name1,name2))
    print(params)
    result=pd.DataFrame([predictions,test],index=['predictions','test']).T.
 ↪dropna()
    result.index=diff.dropna().iloc[size:len(X)].index
    target_result_dic.update({params:result})
else:
    result=target_result_dic[params]
result=target_result_dic[params]
bivariate_plot(result)
series= diff.dropna()
X = series.values
size = int(len(X) * 0.7)
if size<100:
    size = int(len(X) * 0.5)
```

```
train, test = X[0:size], X[size:len(X)]
# result.index=series.iloc[size:len(X)].index
result['position']=(result['predictions']-result['predictions'].rolling(period).
 ↪mean())/result['predictions'].rolling(period).std()
result['position']=result['position'].apply(lambda x:x if abs(x)>1 else 0)
result['position']=result['position'].apply(lambda x:np.sign(x)*2 if abs(x)>2␣
 ↪else x)
# result['position']/=2
transcost=3/10000
result['profit']=result['position']*result['test']-abs(result['position'])*transcost*4
print('average position: '+str(abs(result['position']).replace(0,np.nan).
 ↪mean()))
stoploss=-1.5*abs(result['profit'].std())
result['profit']=result['profit'].apply(lambda x:stoploss if x<stoploss else x)
result['resid']=result['predictions']-result['test']
plt.rcParams.update({'font.size': 15})
plt.figure(figsize=(20*0.7,20*0.618*0.7))
params1=(params[0],params[1]+1,params[2])
plt.title('Cumulative Return on ARIMA'+str(params1))
plt.xlabel('Time')
plt.ylabel('Cumulative Return')
plt.grid()
plt.plot(result['profit'].cumsum().replace(0,np.nan).dropna())
plt.show()
import matplotlib
matplotlib.rcParams.update(matplotlib.rcParamsDefault)
print('Ljung-Box Test Tatistics')
resid=result['test']-result['predictions']
pd.options.display.max_columns=None
ljungbox_df=acorr_ljungbox(resid,10,return_df=True).T
display(ljungbox_df)
evaluate_regr_model(result.iloc[:,0], result.iloc[:,1], figsize=(25*0.8,5*0.8))
resid_analysis(resid)
from sklearn.metrics import r2_score
corr=result.corr().iloc[0,1]
rmse=sqrt(mean_squared_error(result.iloc[:,0],result.iloc[:,1]))
def r_squared(y, y_hat):
    y_bar = y.mean()
    ss_tot = ((y-y_bar)**2).sum()
    ss_res = ((y-y_hat)**2).sum()
    return 1 - (ss_res/ss_tot)
r2=r_squared(result.iloc[:,1],result.iloc[:,0])
annual_profit=result['profit'].cumsum().iloc[-1]/len(result)*255
sharpe=annual_profit/result['profit'].replace(0,np.nan).dropna().std()/(255**0.
 ↪5)
ljung_1=acorr_ljungbox(resid,1,return_df=True).T.iloc[1,0]
```

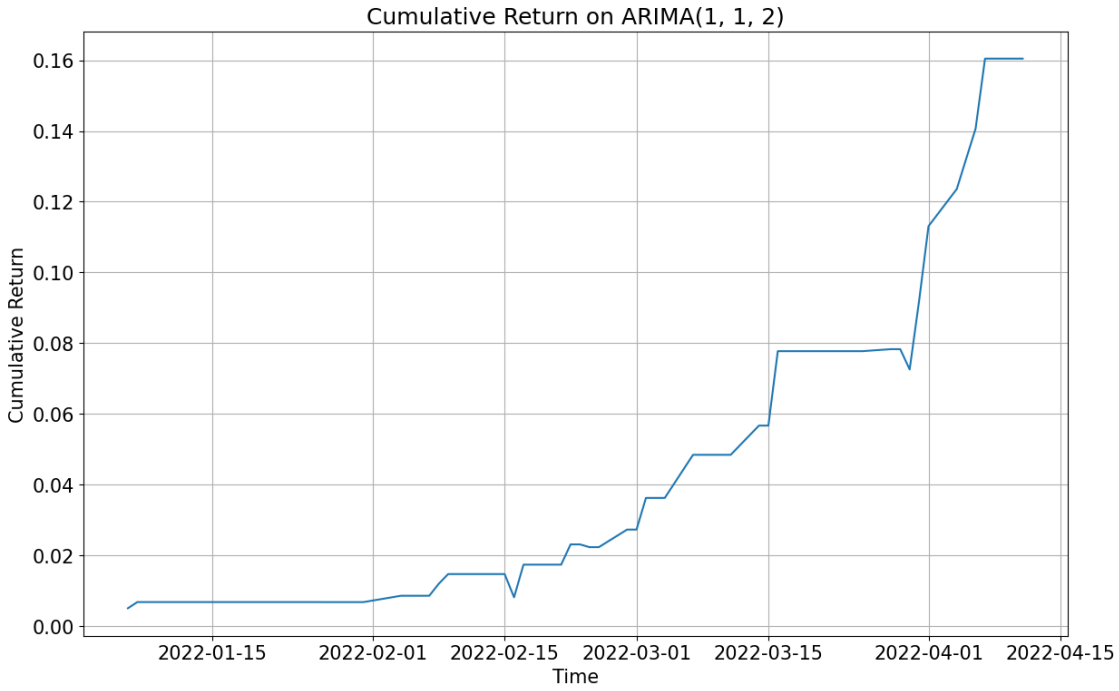
```
winrate=result['profit'].replace(0,np.nan).dropna().apply(lambda x:1 if x>0␣
 ↪else 0).mean()
shapiro_p=shapiro(resid.dropna()).pvalue
performance_df=pd.
 ↪DataFrame([corr,rmse,ljung_1,shapiro_p,r2,annual_profit,sharpe,winrate],index=['correlation
 ↪test(1)-pvalue','Shapiro-Wilk Test pvalue','R-squared','Annualized␣
 ↪Return','Sharpe Ratio','winrate'],columns=['Parameter']).apply(lambda x:
 ↪round(x,4))
display(performance_df)
model_fit = ARIMA(X,order=params).fit(disp=0)
model_fit.summary()
```

```
('HK.03033', 'HK.03067')
(1, 0, 2)
```






```
average position: 1.59049993922608
```

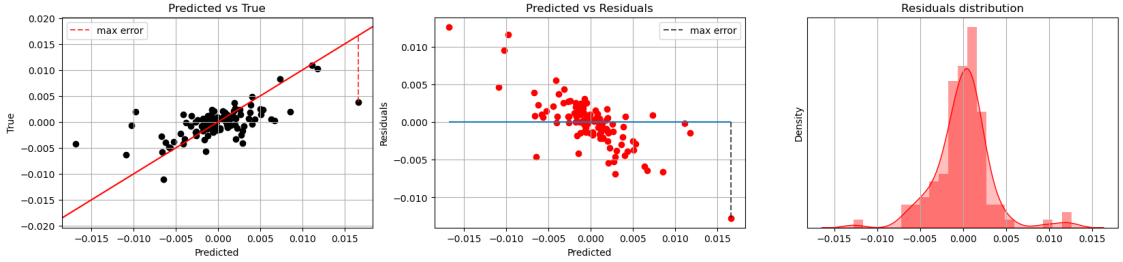


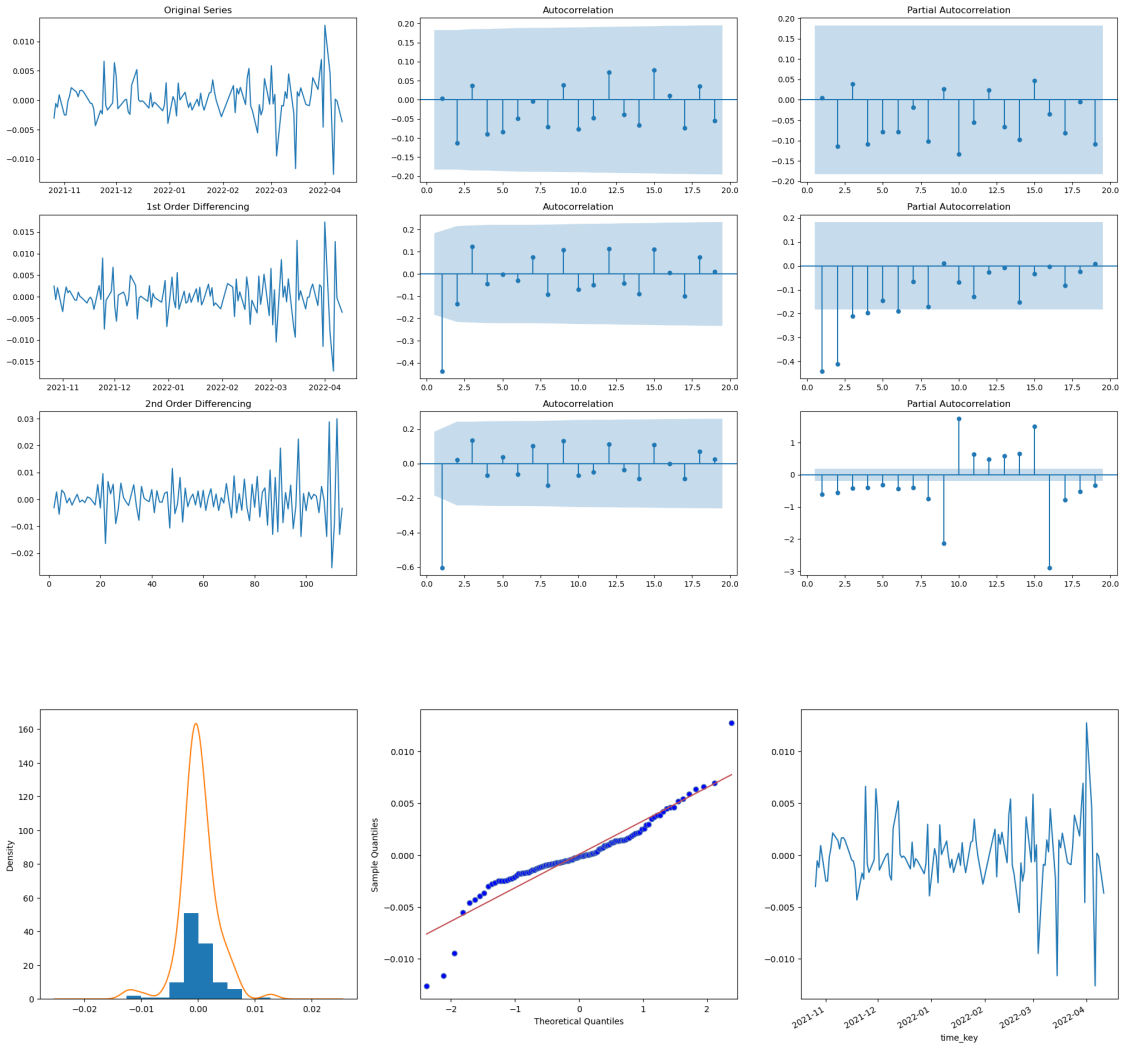
Ljung-Box Test Tatistics

	1	2	3	4	5	6	\
lb_stat	0.001993	1.509829	1.674139	2.647003	3.499096	3.798859	
lb_pvalue	0.964388	0.470051	0.642699	0.618520	0.623524	0.703874	

	7	8	9	10
lb_stat	3.801239	4.442939	4.631384	5.377494
lb_pvalue	0.802365	0.815114	0.865186	0.864578

R2 (explained variance): -0.33
Mean Absolute Perc Error (Σ(|y-pred|/y)/n): 1.5
Mean Absolute Error (Σ|y-pred|/n): 0
Root Mean Squared Error (sqrt(Σ(y-pred)^2/n)): 0
Max Error: -0





Shapiro-Wilk test result: 0.9061086177825928 / 6.498652851405495e-07

	Parameter
correlation	0.6501
rmse	0.0032
ljung-box test(1)-pvalue	0.9644
Shapiro-Wilk Test pvalue	0.0000
R-squared	0.4223
Annualized Return	0.3558
Sharpe Ratio	2.6743
winrate	0.8182

[ ]: <class 'statsmodels.iolib.summary.Summary'>
"""

ARMA Model Results

==========

```
Dep. Variable:                      y   No. Observations:                  382
Model:                     ARMA(1, 2)   Log Likelihood                1630.579
Method:                       css-mle   S.D. of innovations              0.003
Date:                Tue, 03 May 2022   AIC                          -3251.158
Time:                        15:19:40   BIC                          -3231.431
Sample:                             0   HQIC                         -3243.332

==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const      -4.353e-05   1.56e-05     -2.783      0.005   -7.42e-05   -1.29e-05
ar.L1.y       -0.8531      0.342     -2.492      0.013      -1.524      -0.182
ma.L1.y       -0.0360      0.328     -0.110      0.913      -0.678       0.606
ma.L2.y       -0.8027      0.289     -2.773      0.006      -1.370      -0.235
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -1.1722           +0.0000j            1.1722            0.5000
MA.1            1.0939           +0.0000j            1.0939            0.0000
MA.2           -1.1388           +0.0000j            1.1388            0.5000
-----------------------------------------------------------------------------
"""
```

[ ]:
```
ljungbox_df.index=['ljungbox stat','ljungbox p-value']
gen_latex(ljungbox_df.iloc[:,:5])
```

```
\begin{tabular}{lrrrrr}
\hline
                  &        1 &        2 &        3 &       4 &        5 \\
\hline
 ljungbox stat    & 0.22654  & 1.73613  & 2.37615  & 3.1181  & 3.67378  \\
 ljungbox p-value & 0.634101 & 0.419763 & 0.498089 & 0.53826 & 0.597271 \\
\hline
\end{tabular}
```

[ ]:
```
gen_latex(performance_df)
```

```
\begin{tabular}{lr}
\hline
                          &   Parameter \\
\hline
 correlation              &      0.654  \\
 rmse                     &      0.0032 \\
 ljung-box test(1)-pvalue &      0.6341 \\
 Shapiro-Wilk Test pvalue &      0      \\
 R-squared                &      0.4263 \\
 Annualized Return        &      0.3542 \\
```

```
 Sharpe Ratio             &      2.6556 \\
 winrate                  &      0.8182 \\
\hline
\end{tabular}
```

0.7 Auto ARIMA model

[ ]:
```
diff=(np.log(data[name1]/data[name1].shift(1))-np.log(data[name2]/data[name2].
 ↪shift(1))).dropna()
diff=diff.loc[(diff.index.year>2017)&(diff.index.date<pd.
 ↪to_datetime('2023-07-01',format="%Y-%m-%d"))]
diff=diff.loc[(diff.index.year!=2020) | (diff.index.month!=3)]
df = diff.dropna()
model_1 = pm.auto_arima(train, start_p=0, start_q=0,
                      test='adf',       # use adftest to find optimal 'd'
                      max_p=600, max_q=600, # maximum p and q
                      m=1,              # frequency of series
                      d=None,           # let model determine 'd'
                      seasonal=False,   # No Seasonality
                      trace=True,
                      information_criterion='aic',
                      error_action='ignore',
                      suppress_warnings=True,
                      stepwise=False)
model_1.get_params()['order']
model_1.summary()
```

```
ARIMA(0,0,0)(0,0,0)[0]             : AIC=-2121.906, Time=0.06 sec
ARIMA(0,0,1)(0,0,0)[0]             : AIC=-2251.591, Time=0.12 sec
ARIMA(0,0,2)(0,0,0)[0]             : AIC=-2255.389, Time=0.07 sec
ARIMA(0,0,3)(0,0,0)[0]             : AIC=-2254.699, Time=0.15 sec
ARIMA(0,0,4)(0,0,0)[0]             : AIC=-2252.946, Time=0.17 sec
ARIMA(0,0,5)(0,0,0)[0]             : AIC=-2252.266, Time=0.19 sec
ARIMA(1,0,0)(0,0,0)[0]             : AIC=-2193.396, Time=0.10 sec
ARIMA(1,0,1)(0,0,0)[0]             : AIC=-2250.119, Time=0.08 sec
ARIMA(1,0,2)(0,0,0)[0]             : AIC=-2253.268, Time=0.15 sec
ARIMA(1,0,3)(0,0,0)[0]             : AIC=-2254.185, Time=0.43 sec
ARIMA(1,0,4)(0,0,0)[0]             : AIC=-2250.495, Time=0.19 sec
ARIMA(2,0,0)(0,0,0)[0]             : AIC=-2217.443, Time=0.16 sec
ARIMA(2,0,1)(0,0,0)[0]             : AIC=-2255.573, Time=0.22 sec
ARIMA(2,0,2)(0,0,0)[0]             : AIC=-2251.827, Time=0.20 sec
ARIMA(2,0,3)(0,0,0)[0]             : AIC=-2252.843, Time=0.44 sec
ARIMA(3,0,0)(0,0,0)[0]             : AIC=-2232.953, Time=0.08 sec
ARIMA(3,0,1)(0,0,0)[0]             : AIC=-2240.532, Time=0.15 sec
ARIMA(3,0,2)(0,0,0)[0]             : AIC=-2248.542, Time=0.10 sec
ARIMA(4,0,0)(0,0,0)[0]             : AIC=-2238.881, Time=0.13 sec
ARIMA(4,0,1)(0,0,0)[0]             : AIC=-2240.290, Time=0.17 sec
ARIMA(5,0,0)(0,0,0)[0]             : AIC=-2240.844, Time=0.15 sec
```

```
Best model:  ARIMA(2,0,1)(0,0,0)[0]
Total fit time: 3.551 seconds
```

[ ]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==========================================================================
Dep. Variable:                      y   No. Observations:                  267
Model:               SARIMAX(2, 0, 1)   Log Likelihood                1131.787
Date:                Tue, 03 May 2022   AIC                           -2255.573
Time:                        15:00:09   BIC                           -2241.224
Sample:                             0   HQIC                          -2249.809
                                - 267
Covariance Type:                  opg
==========================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------
ar.L1          0.0154      0.083      0.186      0.853      -0.147       0.178
ar.L2         -0.0117      0.080     -0.146      0.884      -0.169       0.146
ma.L1         -0.8416      0.046    -18.416      0.000      -0.931      -0.752
sigma2      1.209e-05   7.02e-07     17.218      0.000    1.07e-05    1.35e-05
==========================================================================
===
Ljung-Box (L1) (Q):                   0.10   Jarque-Bera (JB):
84.57
Prob(Q):                              0.75   Prob(JB):
0.00
Heteroskedasticity (H):               0.47   Skew:
-0.53
Prob(H) (two-sided):                  0.00   Kurtosis:
5.54
==========================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
"""
```

0.8 Predictions Given a pair

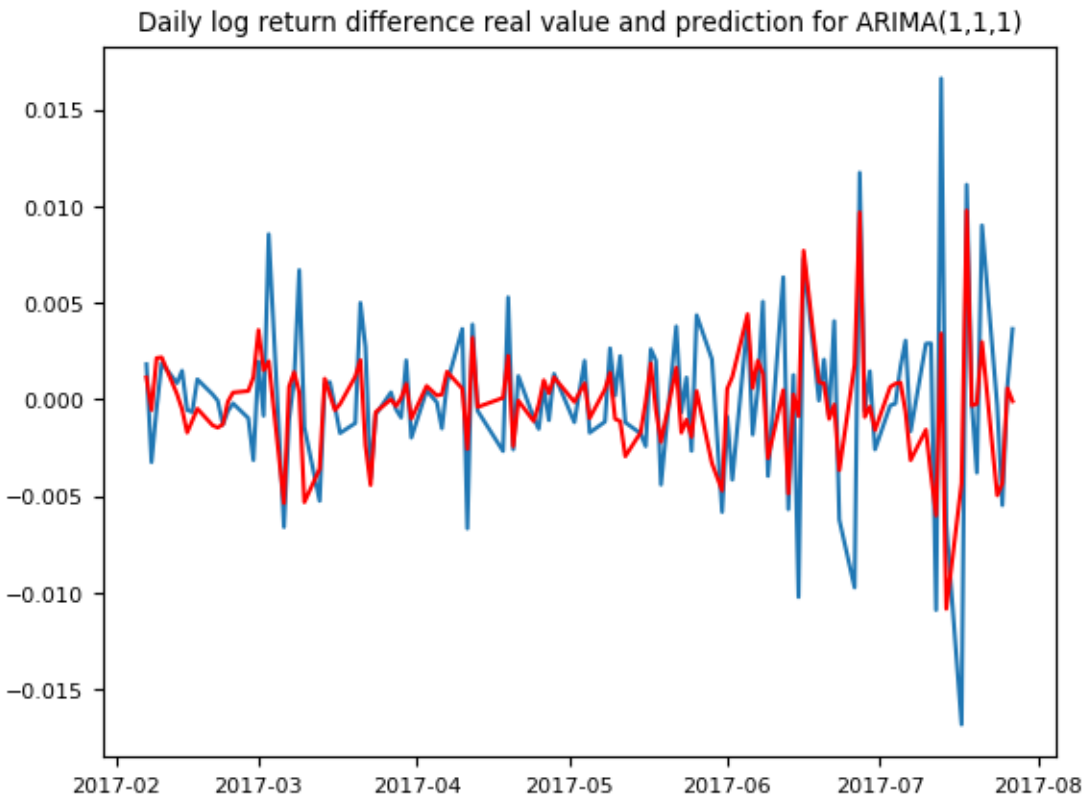
[ ]:
```
fail_dic={}
```

[ ]:
```
fail_dic={}
result_dic={}
param_dic={}
```

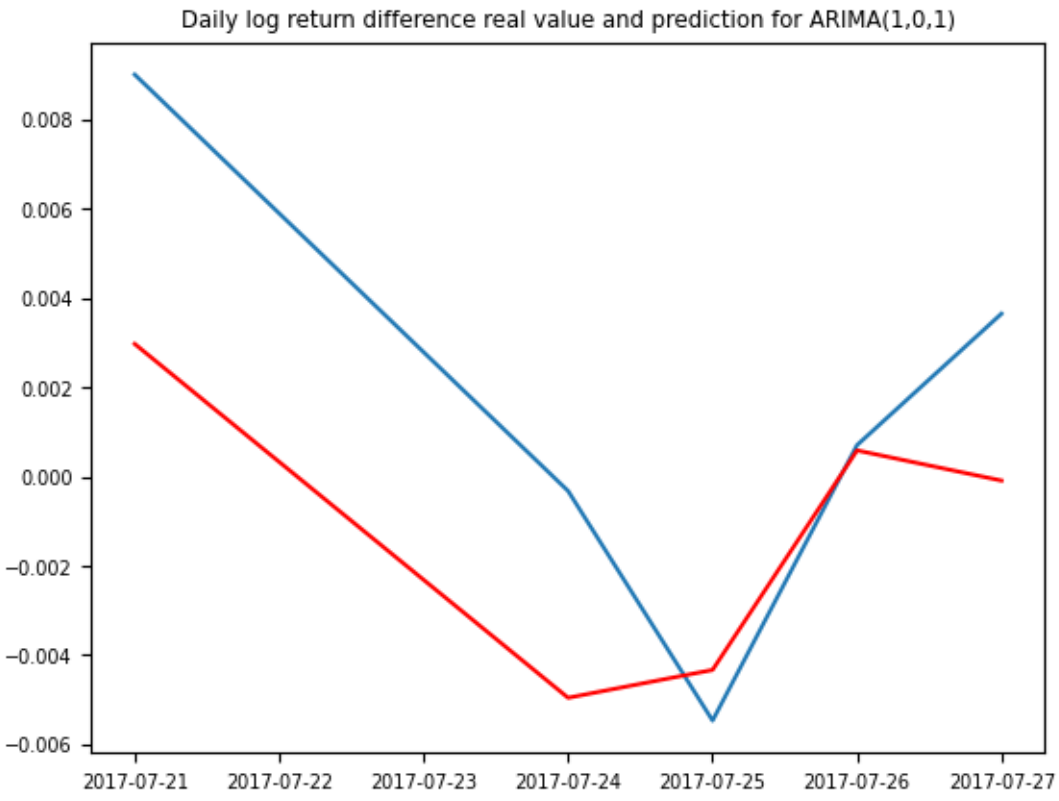
```
[ ]: #Give prediction to a pair given the pair and parameter for ARIMA
name1,name2=('HK.03033', 'HK.03067')
target_series1=log_ret[name1]
target_series2=log_ret[name2]
temp=log_ret[[name1,name2]]
diff=target_series1-target_series2
cum_diff=diff.cumsum()
transcost=3/10000
params=(1,0,1)
try:
    if (name1,name2) not in result_dic.keys():
        diff=(np.log(data[name1]/data[name1].shift(1))-np.log(data[name2]/
 ↪data[name2].shift(1))).dropna()
        diff=diff.loc[(diff.index.year!=2020) | (diff.index.month!=3)]
        diff=diff.loc[diff.index.year>2017]
        df = diff.dropna()
        # print(model.summary())

        series= diff.dropna().reset_index(drop=True)
        # split into train and test sets
        X = series.values
        size = int(len(X) * 0.7)
        if size<100:
            size = int(len(X) * 0.5)
        train, test = X[0:size], X[size:len(X)]
        history = [x for x in train]
        predictions = list()
        # walk-forward validation
        count=0
        for t in range(len(test)):
            count+=1
            print("{:2%}".format(count/len(test)))
            model_fit = ARIMA(history,order=params).fit(disp=0)
            yhat = model_fit.forecast()
            predictions.append(yhat[0][0])
            # upperbound.append(yhat[2][0][0])
            # lowerbound.append(yhat[2][0][1])
            obs = test[t]
            history.append(obs)
            clear_output(wait=False)
        # evaluate forecasts
        # rmse = sqrt(mean_squared_error(test, predictions))
        # print('Test RMSE: %.3f' % rmse)
        # plot forecasts against actual outcomes
        print((name1,name2))
        pyplot.plot(test)
        pyplot.plot(predictions, color='red')
```

```
        pyplot.show()
        result=pd.
 ↪DataFrame([predictions,test],index=['predictions','test'],columns=diff.
 ↪iloc[size:len(X)].index).T.dropna()
        bivariate_plot(result)
        regression(result,mute=True,summary=True)
        param_dic.update({(name1,name2):params})
        result_dic.update({(name1,name2):result})
except Exception as e:
    fail_dic.update({pairs:str(e)})
test=pd.Series(list(test),index=diff.iloc[size:len(X)].index).T
predictions=pd.Series(list(predictions),index=diff.iloc[size:len(X)].index).T
# display real value and regressed values in the test set
print((name1,name2))
plt.rcParams.update({'font.size': 8})
pyplot.title('Daily log return difference real value and prediction for␣
 ↪ARIMA(1,1,1)')
pyplot.plot(test)
pyplot.plot(predictions, color='red')
pyplot.show()

# display real value and regressed values of the last five values
print((name1,name2))
plt.rcParams.update({'font.size': 7})
pyplot.title('Daily log return difference real value and prediction for␣
 ↪ARIMA(1,0,1)')
pyplot.plot(test.iloc[-5:])
pyplot.plot(predictions.iloc[-5:], color='red')
pyplot.show()
```

```
('HK.03033', 'HK.03067')
```




```
('HK.03033', 'HK.03067')
```



0.9 Evaluation of multiple models

Significant pairs are identified earlier

[ ]:
```
pairs
```

[ ]:
```
[('HK.02800', 'HK.07552'),
 ('HK.02828', 'HK.03033'),
 ('HK.02828', 'HK.03067'),
 ('HK.02828', 'HK.03088'),
 ('HK.03032', 'HK.07552'),
 ('HK.03032', 'HK.HHImain'),
 ('HK.03032', 'HK.HTImain'),
 ('HK.03033', 'HK.03067'),
 ('HK.03033', 'HK.07552'),
 ('HK.03033', 'HK.800100'),
 ('HK.03033', 'HK.HHImain'),
 ('HK.03033', 'HK.HTImain'),
 ('HK.03067', 'HK.07552'),
 ('HK.03067', 'HK.HHImain'),
 ('HK.03088', 'HK.07552'),
 ('HK.03088', 'HK.800100'),
```

```
 ('HK.03088', 'HK.HHImain'),
 ('HK.03088', 'HK.HTImain'),
 ('HK.07200', 'HK.07552'),
 ('HK.07552', 'HK.800000'),
 ('HK.07552', 'HK.800100'),
 ('HK.07552', 'HK.HHImain'),
 ('HK.07552', 'HK.HSImain'),
 ('HK.07552', 'HK.HTImain'),
 ('HK.800000', 'HK.HSImain'),
 ('HK.800100', 'HK.HHImain'),
 ('HK.HHImain', 'HK.HTImain')]
```

[ ]:
```
info_df=data.apply(lambda x:[list(x.dropna().head(1).index)[0],list(x.dropna().
 ↪tail(1).index)[0]]).T.rename(columns={0:'start-date',1:'end-date'})
info_df['start-date']=info_df['start-date'].apply(lambda x:str(x)[:10])
info_df['end-date']=info_df['end-date'].apply(lambda x:str(x)[:10])
# info_df['Chinese Name']=quote_ctx.get_market_state(list(info_df.index))[1].
 ↪set_index('code')['stock_name']
info_df['English Name']=['Tracker Fund of Hong Kong','Hang Seng China␣
 ↪Enterprises Index ETF','Hang Seng TECH Index ETF','CSOP Hang Seng TECH Index␣
 ↪ETF ','iShares Hang Seng TECH ETF','ChinaAMC Hang Seng TECH Index ETF','CSOP␣
 ↪Hang Seng Index Daily (2x)','CSOP Hang Seng TECH Index Daily (2x)','CSOP␣
 ↪Hang Seng Index Daily (-2x)','CSOP Hang Seng TECH Index Daily (-2x)','Hang␣
 ↪Seng Index','Hang Seng Chinese Enterprise Index','Hang Seng Chinese␣
 ↪Enterprise Index Future Main','Hang Seng Index Future Main','Hang Seng␣
 ↪Technology Index Future Main']
info_df
```

[ ]:
```
            start-date    end-date  \
code
HK.02800    2015-01-02  2022-04-20
HK.02828    2015-01-02  2022-04-20
HK.03032    2020-09-04  2022-04-20
HK.03033    2020-08-28  2022-04-20
HK.03067    2020-09-17  2022-04-20
HK.03088    2020-09-03  2022-04-20
HK.07200    2017-03-14  2022-04-20
HK.07226    2020-12-10  2022-04-20
HK.07500    2019-05-28  2022-04-20
HK.07552    2020-12-10  2022-04-20
HK.800000   2015-01-02  2022-04-20
HK.800100   2015-01-02  2022-04-20
HK.HHImain  2017-08-14  2022-04-20
HK.HSImain  2015-01-02  2022-04-20
HK.HTImain  2020-11-23  2022-04-20

                                                English Name
```

```
code
HK.02800                          Tracker Fund of Hong Kong
HK.02828              Hang Seng China Enterprises Index ETF
HK.03032                           Hang Seng TECH Index ETF
HK.03033                    CSOP Hang Seng TECH Index ETF
HK.03067                         iShares Hang Seng TECH ETF
HK.03088                  ChinaAMC Hang Seng TECH Index ETF
HK.07200                    CSOP Hang Seng Index Daily (2x)
HK.07226                CSOP Hang Seng TECH Index Daily (2x)
HK.07500                   CSOP Hang Seng Index Daily (-2x)
HK.07552              CSOP Hang Seng TECH Index Daily (-2x)
HK.800000                                      Hang Seng Index
HK.800100                 Hang Seng Chinese Enterprise Index
HK.HHImain  Hang Seng Chinese Enterprise Index Future Main
HK.HSImain                       Hang Seng Index Future Main
HK.HTImain           Hang Seng Technology Index Future Main
```

[ ]:
```
gen_latex(info_df)
```

```
\begin{tabular}{llll}
\hline
 code       & start-date   & end-date   & English Name
\\
\hline
 HK.02800   & 2015-01-02   & 2022-04-20 & Tracker Fund of Hong Kong
\\
 HK.02828   & 2015-01-02   & 2022-04-20 & Hang Seng China Enterprises Index ETF
\\
 HK.03032   & 2020-09-04   & 2022-04-20 & Hang Seng TECH Index ETF
\\
 HK.03033   & 2020-08-28   & 2022-04-20 & CSOP Hang Seng TECH Index ETF
\\
 HK.03067   & 2020-09-17   & 2022-04-20 & iShares Hang Seng TECH ETF
\\
 HK.03088   & 2020-09-03   & 2022-04-20 & ChinaAMC Hang Seng TECH Index ETF
\\
 HK.07200   & 2017-03-14   & 2022-04-20 & CSOP Hang Seng Index Daily (2x)
\\
 HK.07226   & 2020-12-10   & 2022-04-20 & CSOP Hang Seng TECH Index Daily (2x)
\\
 HK.07500   & 2019-05-28   & 2022-04-20 & CSOP Hang Seng Index Daily (-2x)
\\
 HK.07552   & 2020-12-10   & 2022-04-20 & CSOP Hang Seng TECH Index Daily (-2x)
\\
 HK.800000  & 2015-01-02   & 2022-04-20 & Hang Seng Index
\\
 HK.800100  & 2015-01-02   & 2022-04-20 & Hang Seng Chinese Enterprise Index
\\
```

```
 HK.HHImain & 2017-08-14   & 2022-04-20 & Hang Seng Chinese Enterprise Index
Future Main \\
 HK.HSImain & 2015-01-02   & 2022-04-20 & Hang Seng Index Future Main
\\
 HK.HTImain & 2020-11-23   & 2022-04-20 & Hang Seng Technology Index Future Main
\\
\hline
\end{tabular}
```

[ ]:
```
result_dic={}
performance_dic={}
fail_dic={}
```

[ ]:
```
def analyze_result(result,transcost=6/
↪10000,analysis=True,display_analysis=False,period=30,threshold=1,return_ljung=False):
↪
    result['position']=(result['predictions']-result['predictions'].
↪rolling(period).mean())/result['predictions'].rolling(period).std()
    result['position']=result['position'].apply(lambda x:x if abs(x)>threshold␣
↪else 0)
    result['position']=result['position'].apply(lambda x:np.sign(x)*2 if␣
↪abs(x)>2 else x)
    # result['position']/=2
    result['profit']=result['position']*result['test']
    result['transcost']=abs(result['position']-result['position'].
↪shift(1))*transcost
    stoploss=-1.5*abs(result['profit'].std())
    result['profit']=result['profit'].apply(lambda x:stoploss if x<stoploss␣
↪else x)
    result['profit_aftercost']=result['profit']-result['transcost']
    result['resid']=result['predictions']-result['test']
    resid=result['resid']
    ljungbox_df=acorr_ljungbox(resid,5,return_df=True).T
    performance_df=pd.DataFrame()
    if analysis:
        corr=result.corr().iloc[0,1]
        rmse=sqrt(mean_squared_error(result.iloc[:,0],result.iloc[:,1]))
        def r_squared(y, y_hat):
            y_bar = y.mean()
            ss_tot = ((y-y_bar)**2).sum()
            ss_res = ((y-y_hat)**2).sum()
            return 1 - (ss_res/ss_tot)
        r2=r_squared(result.iloc[:,1],result.iloc[:,0])
        annual_profit=result['profit_aftercost'].cumsum().iloc[-1]/
↪len(result)*255
        trade_number_annual=result['position'].apply(lambda x:np.nan if x==0␣
↪else x).dropna().count()/len(result)*255
```

```
        average_position=abs(result['position'].apply(lambda x:np.nan if x==0␣
↪else x).dropna()).mean()
        sharpe=annual_profit/result['profit_aftercost'].replace(0,np.nan).
↪dropna().std()/(255**0.5)
        ljung_1=acorr_ljungbox(resid,1,return_df=True).T.iloc[1,0]
        winrate=result['profit_aftercost'].replace(0,np.nan).dropna().
↪apply(lambda x:1 if x>0 else 0).mean()
        shapiro_p=shapiro(resid.dropna()).pvalue
        performance_df=pd.
↪DataFrame([corr,rmse,ljung_1,shapiro_p,r2,annual_profit,sharpe,winrate,trade_number_annual,
↪test(1)-pvalue','Shapiro-Wilk Test pvalue','R-squared','Annualized␣
↪Return','Sharpe Ratio','winrate','Anual number of trade','average␣
↪position'],columns=['Parameter']).apply(lambda x:round(x,4))
    if display_analysis:
        bivariate_plot(result)
        display(ljungbox_df)
        plot_line(result['profit_aftercost'].replace(0,np.nan).dropna().
↪cumsum(),title='Cumulative profit after cost',figsize=(20,12))
        evaluate_regr_model(result.iloc[:,0], result.iloc[:,1], figsize=(25*0.
↪8,5*0.8))
        resid_analysis(resid)
        display(performance_df)
    if return_ljung:
        return result,performance_df,ljungbox_df
    return result,performance_df
```

[ ]:
```
for pair in log_progress(pairs_list):
    name1,name2=pair
    try:
        print(pair)
        if (name1,name2) not in result_dic.keys():
            period=30
            diff=(np.log(data[name1]/data[name1].shift(1))-np.log(data[name2]/
↪data[name2].shift(1))).dropna()
            diff=diff.loc[(diff.index.year>2016)&(diff.index.date<pd.
↪to_datetime('2023-07-01',format="%Y-%m-%d"))]
            diff=diff.loc[(diff.index.year!=2020) | (diff.index.month!=3)]
            series= diff.dropna().reset_index(drop=True)
            X = series.values
            size = int(len(X) * 0.5)
            if size<100:
                size = int(len(X) * 0.5)
            train, test = X[0:size], X[size:len(X)];history = [x for x in␣
↪train];predictions = list()
            model_1 = pm.auto_arima(train, start_p=0, start_q=0,
```

```
                                test='adf',       # use adftest to find optimal␣
↪'d'
                                max_p=600, max_q=600, # maximum p and q
                                m=1,              # frequency of series
                                d=None,           # let model determine 'd'
                                seasonal=False,   # No Seasonality
                                trace=False,
                                information_criterion='aic',
                                error_action='ignore',
                                suppress_warnings=True,
                                stepwise=False)
            params=model_1.get_params()['order']
            param_dic.update({(name1,name2):params})
            # walk-forward validation
            display((name1,name2))
            display(params)
            count=0
            if (name1,name2) not in result_dic.keys():
                for t in log_progress(range(len(test))):
                    model_fit = ARIMA(history,order=params).fit(disp=0)
                    yhat = model_fit.forecast()[0]
                    predictions.append(yhat[0])
                    obs = test[t]
                    history.append(obs)
                print((name1,name2))
                print(params)
                result=pd.
↪DataFrame([predictions,test],index=['predictions','test']).T.dropna()
                result.index=diff.dropna().iloc[size:len(X)].index
                result_dic.update({(name1,name2):result})
            else:
                result=result_dic[(name1,name2)]
            result,performance_df=analyze_result(result)
            result_dic.update({(name1,name2):result})
            performance_dic.update({(name1,name2):performance_df})
    except Exception as e:
        print((name1,name2))
        print(str(e))
        fail_dic.update({(name1,name2):str(e)})
```

```
VBox(children=(HTML(value=''), IntProgress(value=0, max=27)))
```

```
('HK.02800', 'HK.07552')
```

```
('HK.02800', 'HK.07552')
```

```
(1, 0, 3)
```

```
VBox(children=(HTML(value=''), IntProgress(value=0, max=166)))
```

```
('HK.02800', 'HK.07552')
(1, 0, 3)
('HK.02828', 'HK.03033')

('HK.02828', 'HK.03033')

(0, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=201)))

('HK.02828', 'HK.03033')
(0, 0, 3)
('HK.02828', 'HK.03067')

('HK.02828', 'HK.03067')

(0, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=194)))

('HK.02828', 'HK.03067')
(0, 0, 3)
('HK.02828', 'HK.03088')

('HK.02828', 'HK.03088')

(0, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=199)))

('HK.02828', 'HK.03088')
(0, 0, 3)
('HK.03032', 'HK.07552')

('HK.03032', 'HK.07552')

(1, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=166)))

('HK.03032', 'HK.07552')
(1, 0, 3)
('HK.03032', 'HK.HHImain')

('HK.03032', 'HK.HHImain')

(0, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=198)))

('HK.03032', 'HK.HHImain')
(0, 0, 3)
('HK.03032', 'HK.HTImain')

('HK.03032', 'HK.HTImain')

(1, 0, 1)

VBox(children=(HTML(value=''), IntProgress(value=0, max=173)))
```

```
('HK.03032', 'HK.HTImain')
(1, 0, 1)
('HK.03033', 'HK.03067')

('HK.03033', 'HK.03067')

(0, 0, 2)

VBox(children=(HTML(value=''), IntProgress(value=0, max=194)))

('HK.03033', 'HK.03067')
(0, 0, 2)
('HK.03033', 'HK.07552')

('HK.03033', 'HK.07552')

(1, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=166)))

('HK.03033', 'HK.07552')
(1, 0, 3)
('HK.03033', 'HK.800100')

('HK.03033', 'HK.800100')

(0, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=201)))

('HK.03033', 'HK.800100')
(0, 0, 3)
('HK.03033', 'HK.HHImain')

('HK.03033', 'HK.HHImain')

(0, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=201)))

('HK.03033', 'HK.HHImain')
(0, 0, 3)
('HK.03033', 'HK.HTImain')

('HK.03033', 'HK.HTImain')

(2, 0, 1)

VBox(children=(HTML(value=''), IntProgress(value=0, max=173)))

('HK.03033', 'HK.HTImain')
(2, 0, 1)
('HK.03067', 'HK.07552')

('HK.03067', 'HK.07552')

(1, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=166)))
```

```
('HK.03067', 'HK.07552')
(1, 0, 3)
('HK.03067', 'HK.HHImain')

('HK.03067', 'HK.HHImain')

(0, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=194)))

('HK.03067', 'HK.HHImain')
(0, 0, 3)
('HK.03088', 'HK.07552')

('HK.03088', 'HK.07552')

(1, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=166)))

('HK.03088', 'HK.07552')
(1, 0, 3)
('HK.03088', 'HK.800100')

('HK.03088', 'HK.800100')

(0, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=199)))

('HK.03088', 'HK.800100')
(0, 0, 3)
('HK.03088', 'HK.HHImain')

('HK.03088', 'HK.HHImain')

(0, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=199)))

('HK.03088', 'HK.HHImain')
(0, 0, 3)
('HK.03088', 'HK.HTImain')

('HK.03088', 'HK.HTImain')

(0, 0, 2)

VBox(children=(HTML(value=''), IntProgress(value=0, max=173)))

('HK.03088', 'HK.HTImain')
(0, 0, 2)
('HK.07200', 'HK.07552')

('HK.07200', 'HK.07552')

(1, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=166)))
```

```
('HK.07200', 'HK.07552')
(1, 0, 3)
('HK.07552', 'HK.800000')

('HK.07552', 'HK.800000')

(1, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=166)))

('HK.07552', 'HK.800000')
(1, 0, 3)
('HK.07552', 'HK.800100')

('HK.07552', 'HK.800100')

(1, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=166)))

('HK.07552', 'HK.800100')
(1, 0, 3)
('HK.07552', 'HK.HHImain')

('HK.07552', 'HK.HHImain')

(1, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=166)))

('HK.07552', 'HK.HHImain')
(1, 0, 3)
('HK.07552', 'HK.HSImain')

('HK.07552', 'HK.HSImain')

(1, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=166)))

('HK.07552', 'HK.HSImain')
(1, 0, 3)
('HK.07552', 'HK.HTImain')

('HK.07552', 'HK.HTImain')

(1, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=166)))

('HK.07552', 'HK.HTImain')
(1, 0, 3)
('HK.800000', 'HK.HSImain')

('HK.800000', 'HK.HSImain')

(0, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=640)))
```

```
('HK.800000', 'HK.HSImain')
(0, 0, 3)
('HK.800100', 'HK.HHImain')

('HK.800100', 'HK.HHImain')

(2, 0, 1)

VBox(children=(HTML(value=''), IntProgress(value=0, max=564)))

('HK.800100', 'HK.HHImain')
(2, 0, 1)
('HK.HHImain', 'HK.HTImain')

('HK.HHImain', 'HK.HTImain')

(0, 0, 3)

VBox(children=(HTML(value=''), IntProgress(value=0, max=174)))

('HK.HHImain', 'HK.HTImain')
(0, 0, 3)
```

[ ]:
```
result_dic[list(performance_dic.keys())[2]]
```

[ ]:
```
            predictions      test  position  profit  transcost  \
time_key
2021-11-16    -0.000982 -0.002022       0.0    -0.0        NaN
2021-11-17     0.002418  0.003724       0.0     0.0        0.0
2021-11-18     0.002306  0.011109       0.0     0.0        0.0
2021-11-19     0.001244 -0.000025       0.0    -0.0        0.0
2021-11-22    -0.000164  0.001767       0.0     0.0        0.0
…                   …       …        …      …         …
2022-04-12     0.000319 -0.015060       0.0    -0.0        0.0
2022-04-13    -0.004089  0.004729       0.0     0.0        0.0
2022-04-14    -0.002139 -0.004122       0.0    -0.0        0.0
2022-04-19     0.002906  0.012274       0.0     0.0        0.0
2022-04-20    -0.000361 -0.004189       0.0    -0.0        0.0

            profit_aftercost     resid
time_key
2021-11-16               NaN  0.001040
2021-11-17               0.0 -0.001306
2021-11-18               0.0 -0.008803
2021-11-19              -0.0  0.001269
2021-11-22               0.0 -0.001932
…                        …       …
2022-04-12              -0.0  0.015379
2022-04-13               0.0 -0.008818
2022-04-14              -0.0  0.001984
2022-04-19               0.0 -0.009369
```

```
2022-04-20              -0.0  0.003828

[105 rows x 7 columns]
```

[ ]:
```
result_all=pd.DataFrame()
for key in performance_dic.keys():
    sub=performance_dic[key]
    sub.columns=[str(key)]
    result_all=pd.concat([result_all,sub],axis=1)
```

[ ]:
```
result_all=result_all.T.sort_values('Annualized Return',ascending=False)
```

[ ]:
```
result_filtered=result_all.reset_index().loc[pd.Series(result_all.index).
↪apply(lambda x:False if '7' in x else True)].reset_index(drop=True).
↪rename(columns={'index':'pairs'})
result_filtered
```

[ ]:
```
                         pairs  correlation    rmse  \
0     ('HK.03033', 'HK.HTImain')       0.6915  0.0042
1     ('HK.03088', 'HK.HHImain')       0.2379  0.0125
2       ('HK.02828', 'HK.03088')       0.2202  0.0126
3     ('HK.03032', 'HK.HTImain')       0.6284  0.0043
4   ('HK.HHImain', 'HK.HTImain')       0.1743  0.0133
5     ('HK.03088', 'HK.HTImain')       0.6457  0.0042
6     ('HK.03033', 'HK.HHImain')       0.2116  0.0124
7     ('HK.03032', 'HK.HHImain')       0.2430  0.0124
8       ('HK.02828', 'HK.03033')       0.2145  0.0124
9      ('HK.03088', 'HK.800100')       0.2022  0.0122
10     ('HK.03033', 'HK.800100')       0.1864  0.0121
11   ('HK.800100', 'HK.HHImain')       0.6203  0.0025
12   ('HK.800000', 'HK.HSImain')       0.6109  0.0023

    ljung-box test(1)-pvalue  Shapiro-Wilk Test pvalue  R-squared  \
0                     0.0466                    0.2061     0.4616
1                     0.0239                    0.0000     0.0464
2                     0.0349                    0.0000     0.0367
3                     0.3612                    0.3380     0.3899
4                     0.1104                    0.0000     0.0163
5                     0.0901                    0.2512     0.4083
6                     0.0346                    0.0000     0.0338
7                     0.0310                    0.0000     0.0502
8                     0.0332                    0.0000     0.0329
9                     0.0357                    0.0000     0.0257
10                    0.0382                    0.0000     0.0166
11                    0.1848                    0.0000     0.3829
12                    0.0369                    0.0000     0.3685
```

```
    Annualized Return  Sharpe Ratio  winrate  Anual number of trade  \
0              0.5652        4.6987   0.5172                78.1214
1              0.5206        1.4094   0.3704                64.0704
2              0.4984        1.6745   0.3488                66.6332
3              0.4604        3.6977   0.5128                67.8035
4              0.4208        1.1548   0.3521                63.0172
5              0.4188        3.6163   0.4940                70.7514
6              0.4094        1.1727   0.3418                60.8955
7              0.4018        1.1085   0.3077                61.8182
8              0.4017        1.5165   0.3444                69.7761
9              0.3961        1.3955   0.3488                67.9146
10             0.3451        1.3821   0.3444                67.2388
11             0.2209        2.7736   0.4882                67.8191
12             0.1943        2.7421   0.4593                73.3125

    average position
0             1.4070
1             1.4706
2             1.4451
3             1.4727
4             1.4442
5             1.4266
6             1.4911
7             1.4657
8             1.4050
9             1.4222
10            1.4005
11            1.4967
12            1.4873
```

[ ]:
```
gen_latex(result_filtered.set_index('pairs').apply(lambda x:x.apply(lambda y:
 ↪round(float(y),3)))[[i for i in result_filtered.columns if i !='ljung-box␣
 ↪test(1)-pvalue' and i!='pairs']].iloc[:,:-3])
```

```
\begin{tabular}{lrrrrrr}
\hline
 pairs                         &   correlation &   rmse &   Shapiro-Wilk Test
pvalue &   R-squared &   Annualized Return &   Sharpe Ratio \\
\hline
 ('HK.03033', 'HK.HTImain')    &         0.692 &  0.004 &
0.206 &       0.462 &                0.565 &          4.699 \\
 ('HK.03088', 'HK.HHImain')    &         0.238 &  0.013 &                       0
&       0.046 &                0.521 &          1.409 \\
 ('HK.02828', 'HK.03088')      &         0.22  &  0.013 &                       0
&       0.037 &                0.498 &          1.675 \\
 ('HK.03032', 'HK.HTImain')    &         0.628 &  0.004 &
0.338 &       0.39  &                0.46  &          3.698 \\
 ('HK.HHImain', 'HK.HTImain') &         0.174 &  0.013 &                       0
```

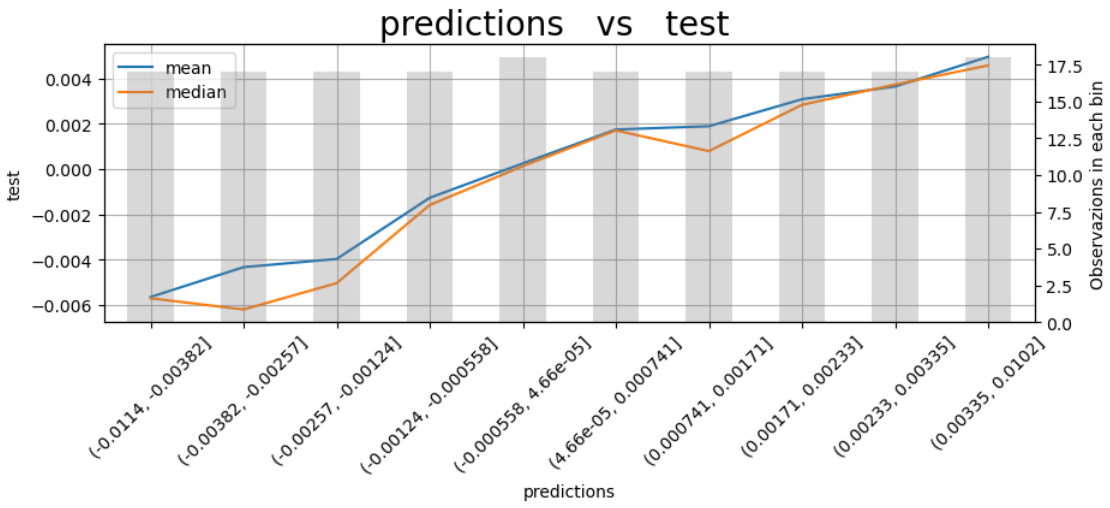
```
&       0.016 &                0.421 &          1.155 \\
 ('HK.03088', 'HK.HTImain')   &         0.646 &  0.004 &
0.251 &       0.408 &                0.419 &          3.616 \\
 ('HK.03033', 'HK.HHImain')   &         0.212 &  0.012 &                      0
&       0.034 &                0.409 &          1.173 \\
 ('HK.03032', 'HK.HHImain')   &         0.243 &  0.012 &                      0
&       0.05  &                0.402 &          1.109 \\
 ('HK.02828', 'HK.03033')     &         0.214 &  0.012 &                      0
&       0.033 &                0.402 &          1.516 \\
 ('HK.03088', 'HK.800100')    &         0.202 &  0.012 &                      0
&       0.026 &                0.396 &          1.395 \\
 ('HK.03033', 'HK.800100')    &         0.186 &  0.012 &                      0
&       0.017 &                0.345 &          1.382 \\
 ('HK.800100', 'HK.HHImain')  &         0.62  &  0.003 &                      0
&       0.383 &                0.221 &          2.774 \\
 ('HK.800000', 'HK.HSImain')  &         0.611 &  0.002 &                      0
&       0.368 &                0.194 &          2.742 \\
\hline
\end{tabular}
```

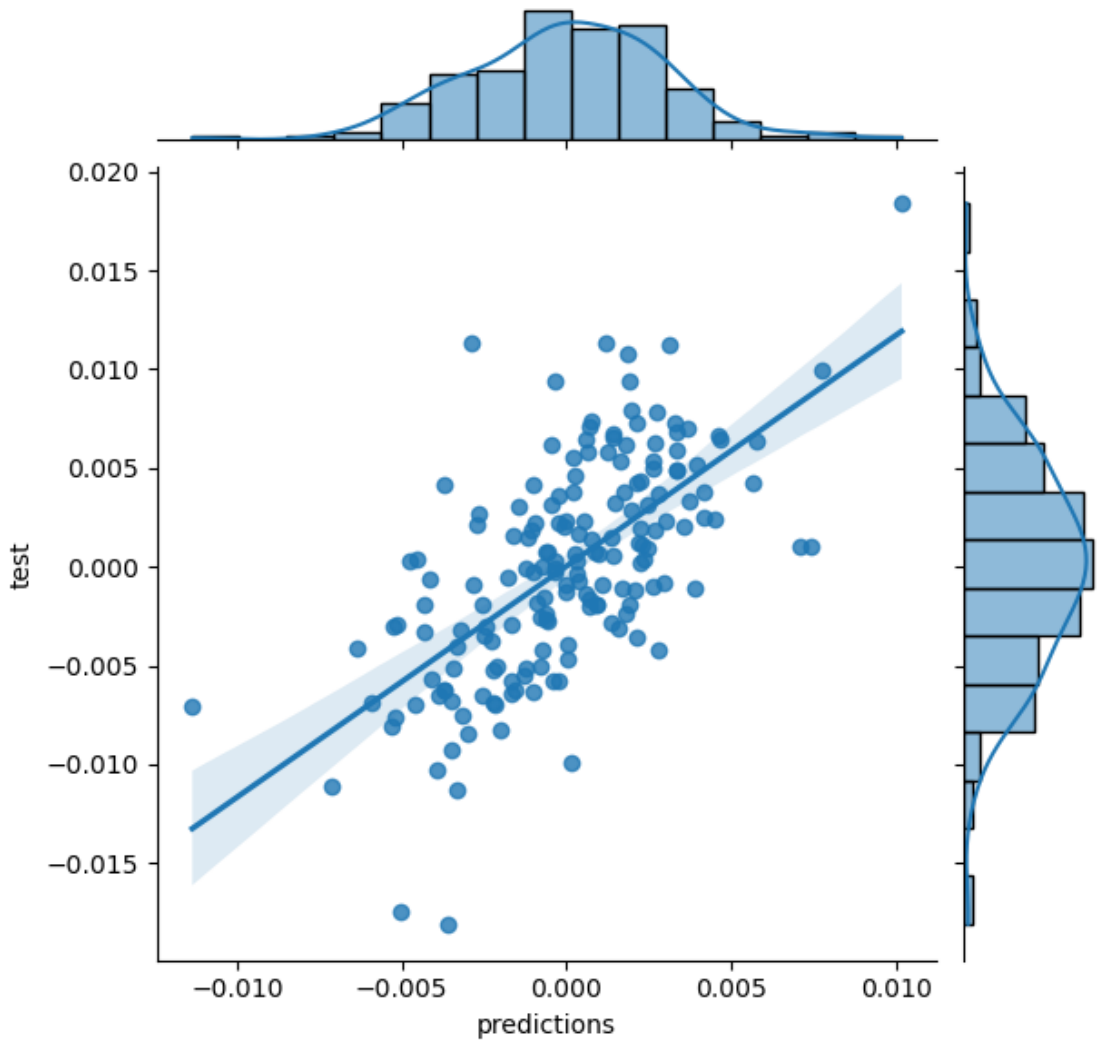
[ ]:
```
target_pair=('HK.03088', 'HK.HTImain')
```

[ ]:
```
param_dic[target_pair]
```

[ ]:
```
(0, 0, 2)
```

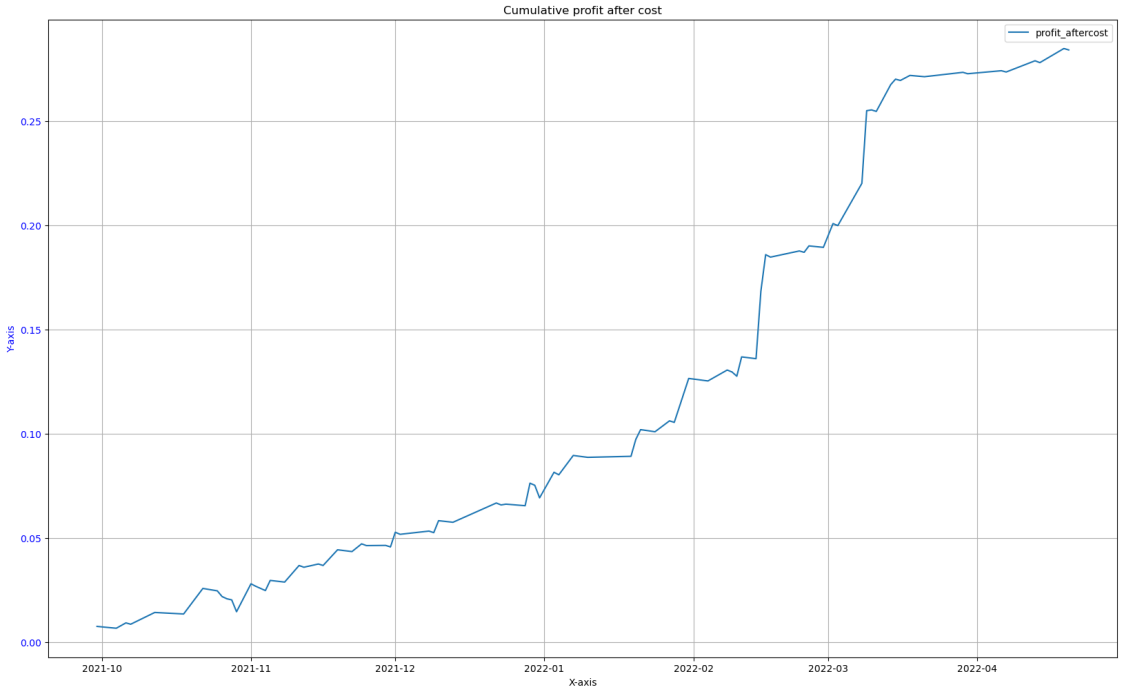
[ ]:
```
result,performance_df,ljung_df=analyze_result(result_dic[target_pair],display_analysis=True,re
```

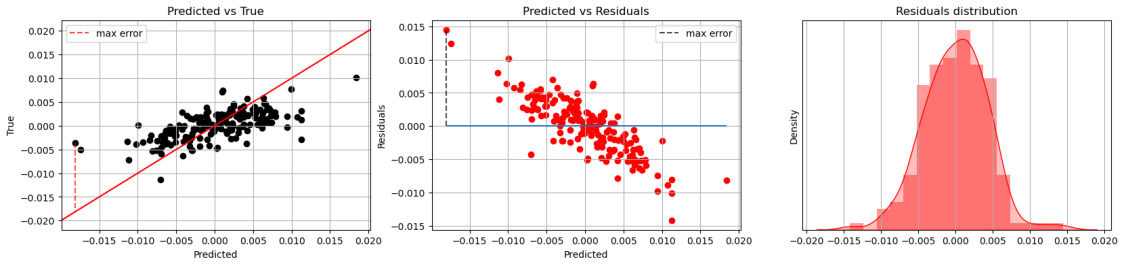


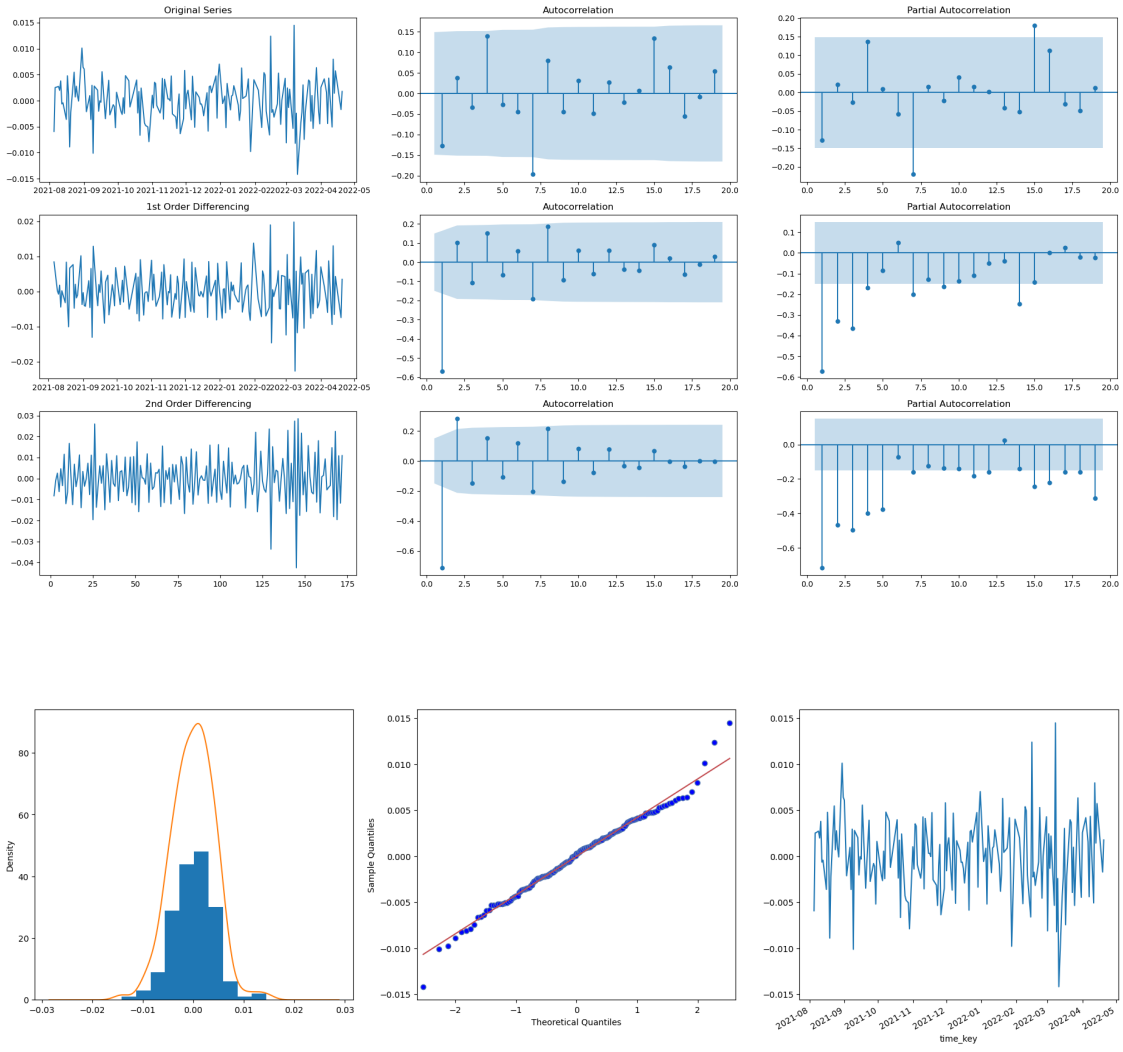


	1	2	3	4	5
lb_stat	2.871858	3.11787	3.317293	6.794192	6.922135
lb_pvalue	0.090141	0.21036	0.345243	0.147172	0.226496

```
R2 (explained variance): -0.94
Mean Absolute Perc Error (Σ(|y-pred|/y)/n): 1.49
Mean Absolute Error (Σ|y-pred|/n): 0
Root Mean Squared Error (sqrt(Σ(y-pred)^2/n)): 0
Max Error: 0
```


Cumulative profit after cost
profit_aftercost
Y-axis
X-axis
0.00
0.05
0.10
0.15
0.20
0.25
2021-10
2021-11
2021-12
2022-01
2022-02
2022-03
2022-04


Predicted vs True
max error
True
Predicted
Predicted vs Residuals
max error
Residuals
Predicted
Residuals distribution
Density




```
Shapiro-Wilk test result: 0.9898039698600769 /  0.25121697783470154
```

	Parameter
correlation	0.6457
rmse	0.0042
ljung-box test(1)-pvalue	0.0901
Shapiro-Wilk Test pvalue	0.2512
R-squared	0.4083
Annualized Return	0.4188
Sharpe Ratio	3.6163
winrate	0.4940
Anual number of trade	70.7514
average position	1.4266

[ ]:
```
gen_latex(performance_df)
```

```
\begin{tabular}{lr}
\hline
                          &   Parameter \\
\hline
 correlation              &      0.6457 \\
 rmse                     &      0.0042 \\
 ljung-box test(1)-pvalue &      0.0901 \\
 Shapiro-Wilk Test pvalue &      0.2512 \\
 R-squared                &      0.4083 \\
 Annualized Return        &      0.4188 \\
 Sharpe Ratio             &      3.6163 \\
 winrate                  &      0.494  \\
 Anual number of trade    &     70.7514 \\
 average position         &      1.4266 \\
\hline
\end{tabular}
```

[ ]:
```
gen_latex(ljung_df)
```

```
\begin{tabular}{lrrrrr}
\hline
           &         1 &       2 &        3 &        4 &        5 \\
\hline
 lb\_stat   & 2.87186   & 3.11787 & 3.31729  & 6.79419  & 6.92214  \\
 lb\_pvalue & 0.0901408 & 0.21036 & 0.345243 & 0.147172 & 0.226496 \\
\hline
\end{tabular}
```

[ ]:
```
data.to_csv('futu.csv')
```

[ ]:
```

```