Evaluation and potential improvements of a deep reinforcement learning model for automated stock trading

A directed study by Rainer Jager (rj63@students.waikato.ac.nz), Student of Computing & Mathematical Sciences, University of Waikato.

February, 2021

Abstract

The basis of this analysis is a model presented at the ACM International Conference in New York on AI in Finance in October 2020. [1]

The authors claim that the introduced deep reinforcement learning ensemble model outperforms the Dow Jones Industrial Average Index, and the three individual algorithms that form the ensemble in terms of the risk-adjusted returns measured by the Sharpe ratio. Furthermore, it is claimed that the ensemble is more robust and reliable than the individual agents. [1]

We evaluate these claims for statistical significance. As some weaknesses of the model become evident, we suggest a work-around and show the results with the suggested alteration. Finally, we combine all the findings and present an alternative model.

Acknowledgements

This work was done under the supervision of Prof. Bernhard Pfahringer (bernhard@waikato.ac.nz). His teachings in Data Mining, Machine Learning and Deep Learning helped me build a solid foundation to further explore on. I would also like to thank him for his valuable feedback as part of this directed study.

Table of contents

1. Introduction

The fund management industry may be organized into active and passive. Passive fund management tries to replicate the performance of an index and has become very popular since the Global Financial Crisis in the form of Exchange Traded Funds. [2]

Active fund management aims to outperform the index via discretionary or automated strategies. With the ever increasing computational power, advances in the field of Artificial Intelligence[1], and the extraordinary success of pioneers like Jim Simons [4], active, automated, quantitative approaches in the form of computer programs - commonly known as “algos” - now make up the vast majority of trading volume on exchanges in the US[2].

The model which forms the basis of this analysis is one example of such a computer program. [1]

The remainder of this paper introduces related work. It then gives a bird’s-eye-view of how the algorithm works. In section 4 we look at the results of the deep learning ensemble, discuss shortfalls of the model and suggest a work-around. The summary of our findings is the basis of a new model. We then compare the performance of the new model to the original model. [1] It concludes with ideas for future work. Appendix A contains the details to the statistical tests and Appendix B contains hitherto not mentioned experiments.

[1] “Recently, self-learning systems have achieved remarkable success in several challenging problems for artificial intelligence, by combining reinforcement learning with deep neural networks. In this talk, I describe the ideas and algorithms that led to AlphaGo: the first program to defeat a human champion in the game of Go; AlphaZero: which learned, from scratch, to also defeat the world computer champions in chess and shogi; and AlphaStar: the first program to defeat a human champion in the real-time strategy game of StarCraft.” [3]

[2] “In the U.S. stock market and many other developed financial markets, about 70-80 percent of overall trading volume is generated through algorithmic trading.” [5]

2. Related work

The rise of the use of algos in finance as seen in the share of trading volume has mainly been driven by advances in the underlying computer programs. The development from machine learning to the use of artificial intelligence in the form of neural networks in general, and deep reinforcement learning algorithms in particular has its origins in the success of these algos in other domains, specifically in gaming. [3]

Deep Reinforcement learning combines the idea of object formulation and object optimization in the form of reinforcement learning and deep learning. [3] These types of algorithms are typically categorized into either of the following three types: a) actor-only, b) critic-only, and c) actor-critic. [6]

The idea of the actor-only type is to learn from the observation state directly. The critic-only algorithms on the other hand are choosing their actions based on the value-network's prediction. Finally, the actor-critic algorithm's idea is to exploit the advantages of both types. It employs two agents: an actor deciding actions based on the state of the environment and a critic computing the rewards of those actions. The idea is that the actor's network is gradually adjusted so to maximize the rewards predicted by the critic. [6]

The deep reinforcement learning ensemble model at the heart of this study is a pure actor-critic ensemble. All its three algos - Advantage Actor Critic (A2C), Deep Deterministic Policy Gradient (DDPG), and Proximal Policy Optimization (PPO) fall into this category. [1]

3. Algorithm

Firstly, the stock market data of the Dow Jones Industrial Index for its 30 members from 2009 to mid-2020 is preprocessed. This process results in consistent, that is adjusted for stock splits and dividends, end of day prices, price-based technical indicators[3], and a turbulence[4] index.

The individual agents then get trained from January 2009 to October 2015 based on the preprocessed data. The agent that achieved the highest risk-adjusted return, commonly known

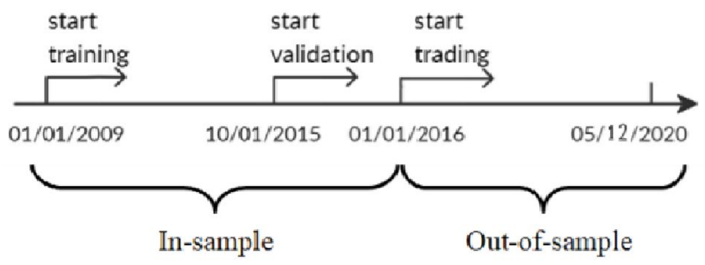
[3] Technical indicators try to take advantage of market inefficiencies and are used in short-term trading. They are either trend- or momentum-based. The model introduces four such indicators to the observation space. [1]

[4] $\text{turbulence}_t = (y_t - \mu)\, \Sigma^{-1} (y_t - \mu)' \in R$, where $y_t \in R^{D}$ denotes the stock returns for current period t, $\mu \in R^{D}$ denotes the average of historical returns, and $\Sigma \in R^{D\times D}$ denotes the covariance of historical returns. [1]

as Sharpe ratio[5], in the three months prior to the start of the trading period - January 2016 - is picked by the ensemble.

This three month selection process is called the validation period. The selected algorithm then exclusively trades for the next three months.

Prior to every day of trading the level of the current turbulence index is compared with a constant threshold and all positions are squared if the current index exceeds that threshold. Trading resumes when the turbulence index falls below the threshold.

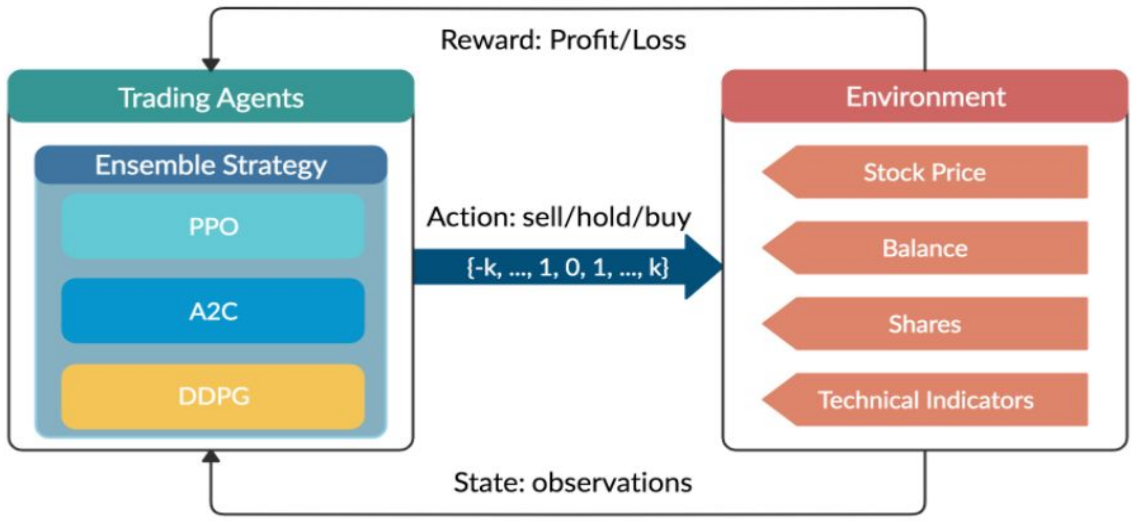


This training-validation-trading cycle gets extended by three months and repeated until the end.

The stock market is modelled as a Markov Decision Process in a standard reinforcement learning environment. [7]

Graph 1: Stock data splitting. Adjusted from [1]

Stock prices, a money balance, the current portfolio, and four technical indicators are part of the observation space. Buy, sell, or holding of securities are part of the action space. Finally, a reward function to maximise the money balance is formulated.



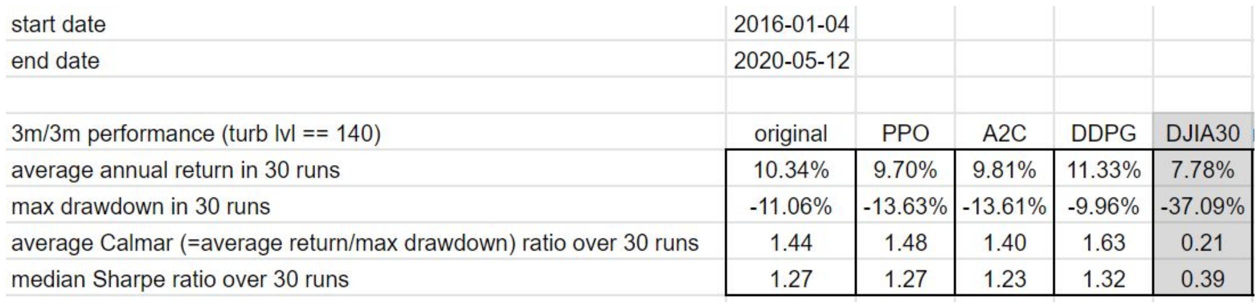
Graph 2: Modelling of a stock-market environment from [1]

[5] We define the Sharpe ratio as: (annual return) / (annualized standard deviation of daily returns)

For a more detailed discussion see [1] [7] and for the python code see [8].

4. Experiments

This report focuses on an analysis of the performance of the deep reinforcement learning ensemble model introduced in the paper and suggestions for improving its performance. The results of section 6.2 of the paper are based on only seven runs[6]. [1] This is not sufficient to draw statistically supported conclusions. [9] Based on 30 runs each, the trading performance of the ensemble, its agents individually, and the Dow Jones Index is as follows:



start date	2016-01-04				
end date	2020-05-12				
3m/3m performance (turb lvl == 140)	original	PPO	A2C	DDPG	DJIA30
average annual return in 30 runs	10.34%	9.70%	9.81%	11.33%	7.78%
max drawdown in 30 runs	-11.06%	-13.63%	-13.61%	-9.96%	-37.09%
average Calmar (=average return/max drawdown) ratio over 30 runs	1.44	1.48	1.40	1.63	0.21
median Sharpe ratio over 30 runs	1.27	1.27	1.23	1.32	0.39

Table 1: Performance overview

The benchmark, DJIA30, is greyed out as it is the result of one backtest based on the data compared to the 30 runs each of the ensemble and its individual agents that were produced from stochastic learning processes.

We can now test the paper's claim of risk-adjusted outperformance of the ensemble compared to a buy and hold strategy of the benchmark index for statistical significance.

A test[7] of normal distribution for the ensemble's Sharpe ratios produces a high p-value and hence the hypothesis of normally distributed Sharpe ratios can not be dismissed. The z-value of the buy & hold Sharpe ratio is outside a 95%-confidence interval. Consequently, we agree with the paper's claim of the ensemble's risk-adjusted outperformance based on statistical significance.

[6] see backtesting.ipynb, 24th January 2021 on webpage: https://github.com/AI4Finance-LLC/Deep-Reinforcement-Learning-for-Automated-Stock-Trading-Ensemble-Strategy-ICAIF-2020

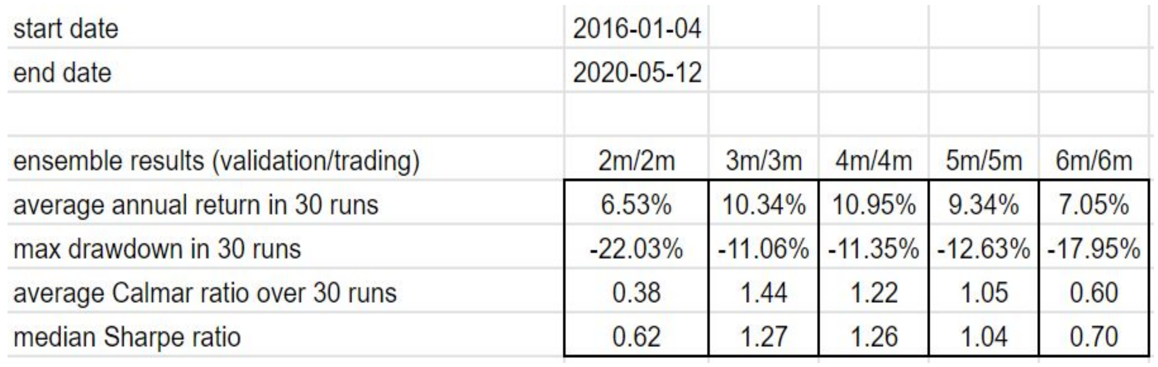
[7] See detailed calculations under Appendix A.1, and Appendix A.1.1

4.1 Analysis of potential shortfalls of the ensemble model

4.1.1 Validation/trading period

The ensemble model introduced uses a three month validation period at the end of which the agent with the highest Sharpe ratio during that period is picked to exclusively trade for the next three months. [1] This logic is rolled forward until the end of the trading period. Additionally, it uses a constant turbulence index threshold level of 140 to avoid trading during unusual volatile periods.

Is it possible to achieve better results with a different combination of validation/trading periods? Simulation of 30 runs each produced the following results:



start date	2016-01-04				
end date	2020-05-12				
ensemble results (validation/trading)	2m/2m	3m/3m	4m/4m	5m/5m	6m/6m
average annual return in 30 runs	6.53%	10.34%	10.95%	9.34%	7.05%
max drawdown in 30 runs	-22.03%	-11.06%	-11.35%	-12.63%	-17.95%
average Calmar ratio over 30 runs	0.38	1.44	1.22	1.05	0.60
median Sharpe ratio	0.62	1.27	1.26	1.04	0.70

Table 2: Performance overview for different validation/trading period combinations

It is evident that a 4m/4m combination of validation/trading period produced higher returns. However, the 3m/3m combination produced the highest risk-adjusted returns. We therefore agree that a 3m/3m validation/trading period combination chosen by the authors [1] produces the best risk-adjusted returns.

4.1.2 Validation rule

Intuitively, it looks like a good idea to pick a trader that is currently in form to do the trading for the immediate future. This is essentially what the ensemble is doing by using a three months validation period and deciding the agent that is going to be trading for the next three months based on the agents' risk-adjusted performance during the validation period.

However, if we simulate 30 runs of an ensemble with the same validation/trading combination and a turbulence index of 140 that picks its agent to do the trading randomly, the results are as follows:

start date	2016-01-04	
end date	2020-05-12	
3m/3m performance (turb lvl == 140)	original-ensemble	random-choice
average annual return in 30 runs	10.34%	10.42%
max drawdown in 30 runs	-11.06%	-10.19%
average Calmar ratio over 30 runs	1.44	1.47
median Sharpe ratio over 30 runs	1.27	1.29

Table 3: Performance of original-ensemble versus a random-choice ensemble

The results produced by 30 simulations of the random-choice ensemble are better on all four performance criteria, notably it also produced better risk-adjusted returns.

As the difference between the two models is very small, we conclude that the result of a 'random-choice' ensemble produces as good risk-adjusted returns as the original ensemble.[8]

4.1.3 Turbulence Index

Firstly, we want to find the best threshold level possible: Introducing various threshold levels for the buy and hold strategy produces the following results for the period between 4th January 2016 to 12th May 2020:

turbulence index threshold	60	80	100	120	140	160	180	200	220
annual return	3.53%	4.34%	8.03%	12.82%	12.82%	10.30%	7.03%	7.64%	7.62%
cumulative returns	16.30%	20.30%	39.97%	69.05%	69.05%	53.23%	34.44%	37.80%	37.68%
annual volatility	3.67%	6.31%	8.65%	11.23%	11.23%	13.96%	15.01%	15.13%	15.55%
Sharpe ratio (r=0)	0.96	0.69	0.93	1.14	1.14	0.74	0.47	0.51	0.49
max drawdown	-4.84%	-8.11%	-8.47%	-14.62%	-14.62%	-22.57%	-21.71%	-20.27%	-24.94%
RoMaD ("Calmar")	0.73	0.53	0.95	0.88	0.88	0.46	0.32	0.38	0.31

Table 4: Dow Jones Industrial performance with different turbulence index levels

A t-test between the produced Sharpe ratios of a turbulence index threshold level of 120 and 140 for each quarter during the period 2010 to 2016 was not able to dismiss the

[8] See detailed calculations under Appendix A.2 and Appendix A.2.1

hypothesis that both produce equally good risk-adjusted returns.[9] We conclude that the threshold of 140 for the turbulence index as used in the paper [1] is the right level if a turbulence index is used.

It becomes clear that the introduction of the turbulence index for the benchmark vastly improves its risk-adjusted performance from a Sharpe ratio of 0.39 as per table 1 to 1.14 as per table 4.

Even though the performance of the benchmark improves with the help of the turbulence index, the ensemble model still produces significantly better risk-adjusted returns.[10]

However, the introduction of a turbulence index threshold is problematic as it requires to set a constant variable in a dynamic environment.

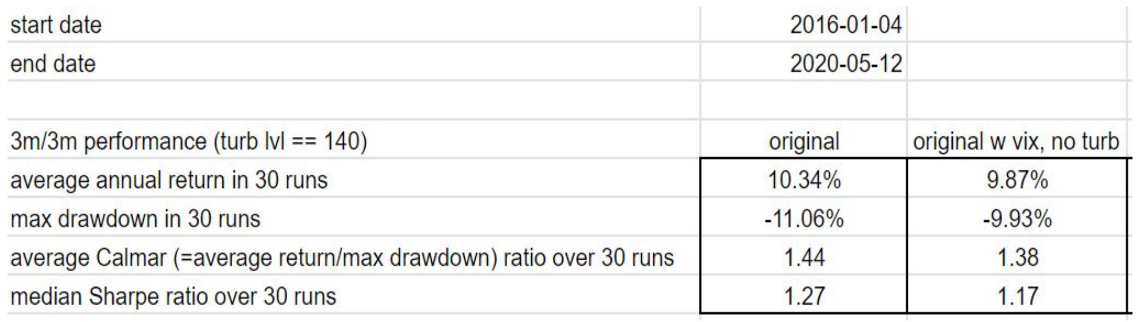
4.1.3.1 An alternative to the turbulence index

One way to tweak the ensemble and circumvent the problem of setting a constant in a dynamic environment and the critique of having its performance to compare to an equivalent benchmark is to replace the turbulence index with a new, observable variable like the VIX.[11]

This gives the ensemble the opportunity to learn adjusting its positions when there is turbulence ahead. Turbulence is then based on its own judgement.

Here is the result produced by 30 samples of the ensemble without a turbulence index but VIX data instead:

[9] See detailed calculations under Appendix A.3 and Appendix A.3.1
[10] See detailed calculations under Appendix A.3.2
[11]VIX data is freely available for download. “The VIX Index is a financial benchmark designed to be an up-to-the-minute market estimate of expected volatility of the S&P 500 Index, and is calculated by using the midpoint of real-time S&P 500® Index (SPX) option bid/ask quotes. More specifically, the VIX Index is intended to provide an instantaneous measure of how much the market thinks the S&P 500 Index will fluctuate in the 30 days from the time of each tick of the VIX Index.”, 26th January 2021 from webpage: https://www.cboe.com/tradable_products/vix/faqs/



start date	2016-01-04	
end date	2020-05-12	
3m/3m performance (turb lvl == 140)	original	original w vix, no turb
average annual return in 30 runs	10.34%	9.87%
max drawdown in 30 runs	-11.06%	-9.93%
average Calmar (=average return/max drawdown) ratio over 30 runs	1.44	1.38
median Sharpe ratio over 30 runs	1.27	1.17

Table 5: Performance of the original ensemble versus an ensemble without a turbulence index but VIX data

We can conclude that the newly introduced ensemble's risk-adjusted returns are equivalent to the original ensemble's.[12] Additionally, the problem of finding the optimal turbulence level and then setting this level as a constant has been avoided.

4.1.4 Ensemble versus agents

As indicated in Table 1, the performance of the ensemble is not better than its agents. In fact, it is worthwhile analyzing if the DDPG agent outperforms the ensemble significantly. Despite having outperformed the ensemble in all four performance criteria, the result of a t-Test[13] on the produced Sharpe ratios shows that the risk-adjusted outperformance of the DDPG agent is not significant.

4.1.4.1 Robustness of DDPG results

The authors [1] claim to make the trading strategy more robust and reliable by deploying the ensemble. "Annualized volatility and max drawdown measure the robustness of a model." [1]

The claim of increasing the strategy's robustness and reliability can not be maintained. Even though the annualized volatility of the returns is significantly lower for the ensemble it is not the case for the drawdowns.[14]

[12] See detailed calculations under Appendix A.4 and Appendix A.4.1
[13] See detailed calculations under Appendix A.5 and Appendix A.5.1
[14] See detailed calculations under Appendix A.6, Appendix A.6.1, Appendix A.7 and Appendix A.7.1

4.2 Summary of findings

The risk-adjusted performance of a single DDPG agent produces as high risk-adjusted returns as the original ensemble. Its results are as robust and reliable as the ensemble's.

Additionally, every three months the ensemble uses a validation period to pick the best-performing agent to do the trading for the next quarter. Given that the results of an ensemble that picks its trader randomly produces equally good results, we see no value in the validation process.

Finally, we regard the use of a constant turbulence index level as problematic.

We conclude that an improved model should be built around a DDPG agent only. This is computationally less expensive, and it eliminates the decision after the validation period in the original model. To further make use of our findings, we are looking at the performance of the DDPG model without a turbulence index but VIX data instead in our next section.

5. A new model

Having shown the weaknesses of the ensemble, a new, improved model may be built as a sole 3m3m DDPG agent with VIX data as part of the observation space instead of using a turbulence index level.

The simulation of 30 runs arrives at the following results:

start date	2016-01-04	
end date	2020-05-12	
3m/3m performance (turb lvl == 140)	original-ensemble	new model
average annual return in 30 runs	10.34%	11.36%
max drawdown in 30 runs	-11.06%	-9.41%
average Calmar ratio over 30 runs	1.44	1.56
median Sharpe ratio over 30 runs	1.27	1.39

Table 6: Performance of the original ensemble versus the new model

The results look very promising as the new model beats the original ensemble model in all categories. Although better, the risk-adjusted outperformance of the new model is not significant.[15]Nevertheless, we think the newly introduced model is better compared to the original ensemble [1]:

It produces equally good risk-adjusted returns.
It Is equally robust[16].
It does not require a constant turbulence index threshold.
It does not require a validation period criteria.
It is less computationally expensive.
It is simpler.

6. Conclusions and ideas for future work

It is possible to improve the original ensemble model by focusing on its best performing agent and introducing VIX data instead of a turbulence index.

There are plenty of areas for future work to improve the performance of new models that are either based on the original ensemble or the newly introduced model.

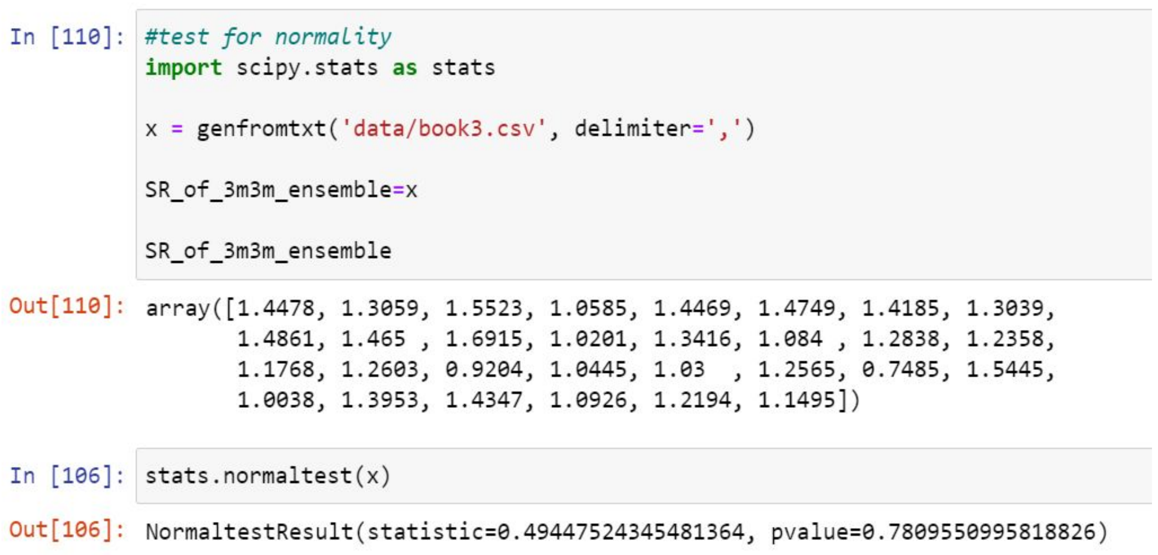
We see opportunities in a revised ensemble algorithm that may not use its agents exclusively during the trading period but weights its actions as per the validation period performance. It is also worthwhile to consider additional agents or different agents as part of the ensemble. In addition to the authors [1] suggested areas of future research, we also see potential in expanding the action space for short selling of stocks, and introducing money management rules. [12]

As the ensemble is computationally expensive, there may be more scope for additional variables in the newly introduced model. Particularly trading volume is often used in context with technical indicators. It may be worth amending the model itself by introducing parameter noise. [13] Probably the most challenging but also most promising area of research may be the introduction of new, deeper neural networks.

[15] See detailed calculations under Appendix A.8 and Appendix A.8.1
[16] See detailed calculations under Appendix A.9, Appendix A.9.1, Appendix A.10 and Appendix A.10.1

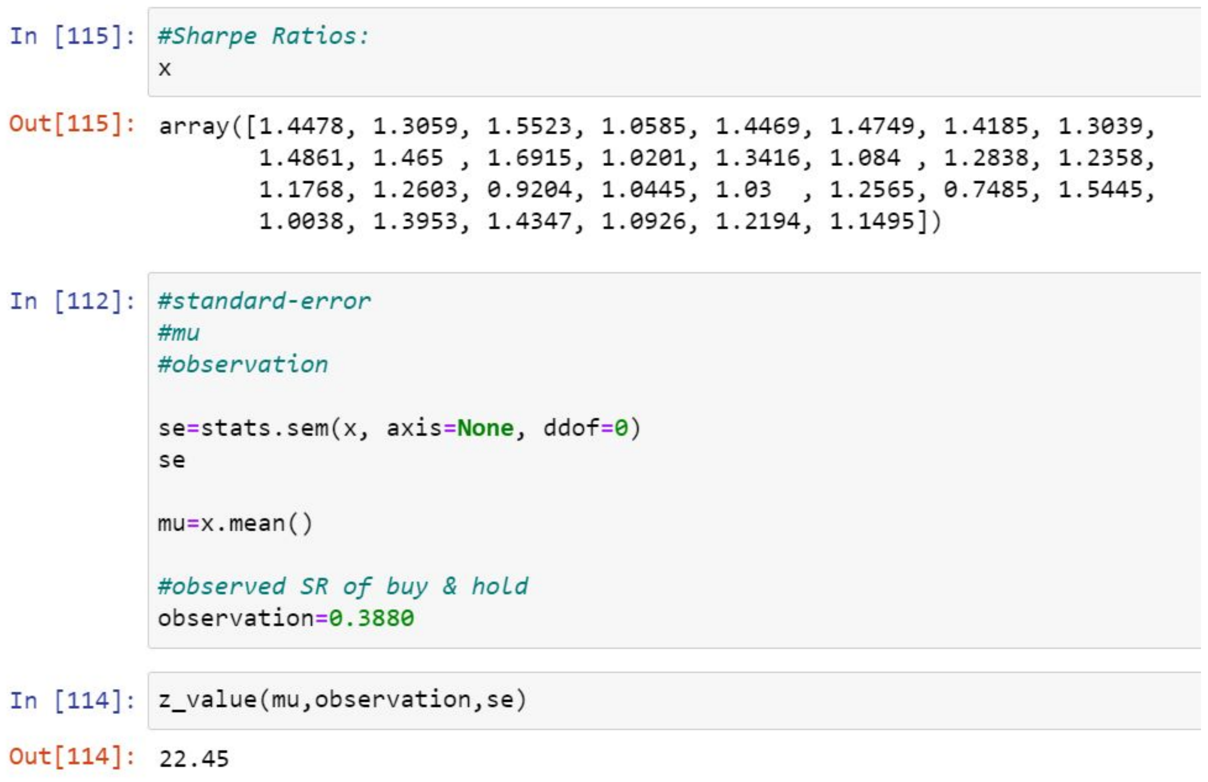
Appendix A: Statistical Tests

A.1 D'Agostino and Pearson test for normal distribution of the Sharpe ratios produced by the ensemble



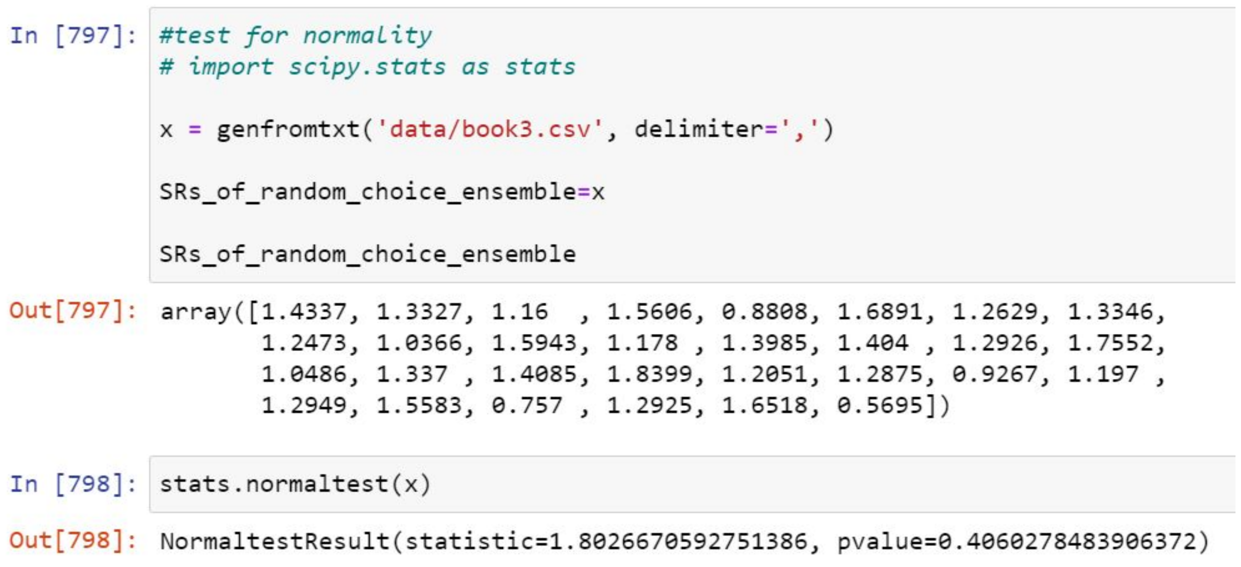
```
In [110]: #test for normality
          import scipy.stats as stats

          x = genfromtxt('data/book3.csv', delimiter=',')

          SR_of_3m3m_ensemble=x

          SR_of_3m3m_ensemble
Out[110]: array([1.4478, 1.3059, 1.5523, 1.0585, 1.4469, 1.4749, 1.4185, 1.3039,
                 1.4861, 1.465 , 1.6915, 1.0201, 1.3416, 1.084 , 1.2838, 1.2358,
                 1.1768, 1.2603, 0.9204, 1.0445, 1.03  , 1.2565, 0.7485, 1.5445,
                 1.0038, 1.3953, 1.4347, 1.0926, 1.2194, 1.1495])

In [106]: stats.normaltest(x)
Out[106]: NormaltestResult(statistic=0.49447524345481364, pvalue=0.7809550995818826)
```

A.1.1 z-Value of buy and hold strategy



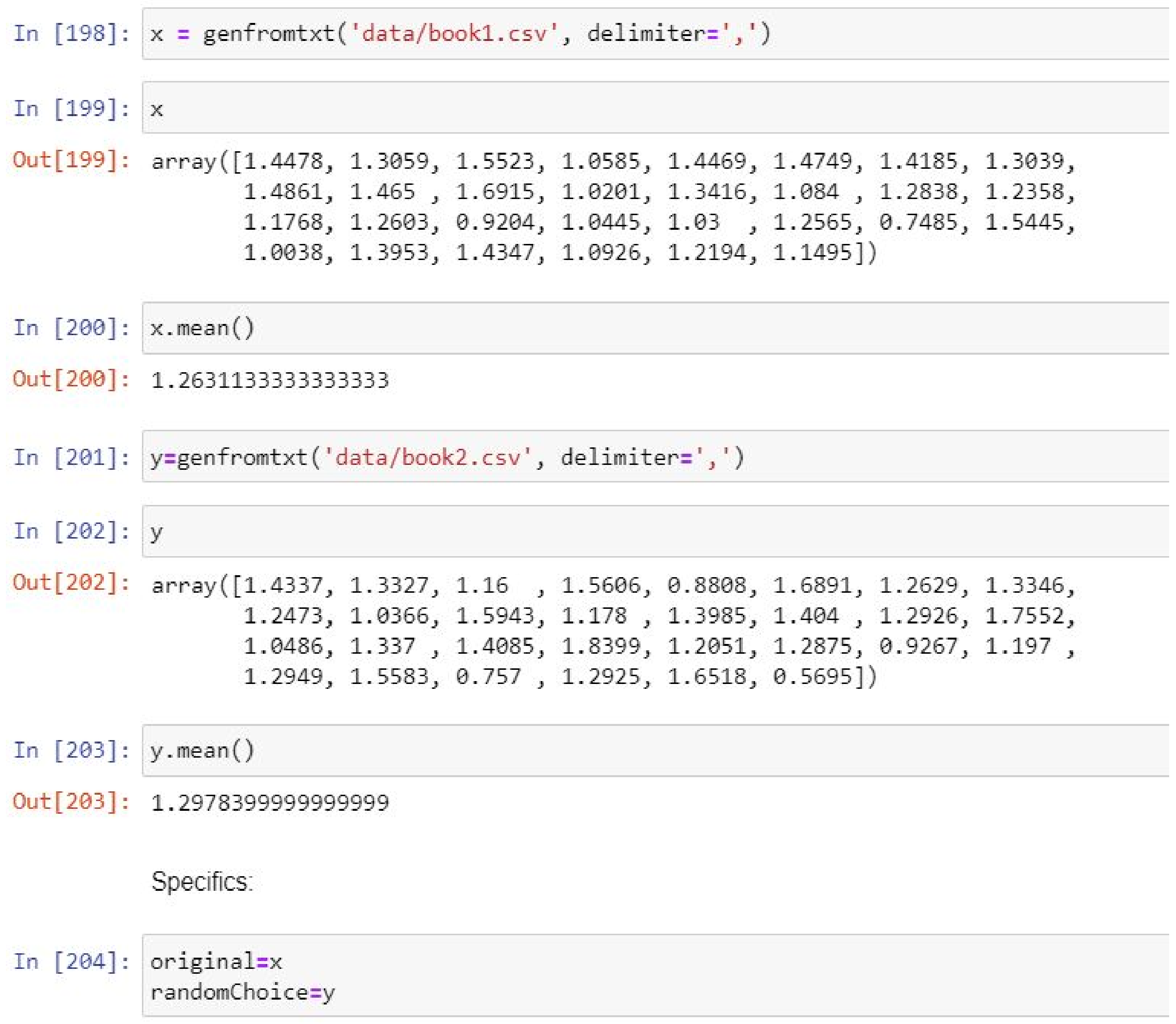
```
In [115]: #Sharpe Ratios:
          x

Out[115]: array([1.4478, 1.3059, 1.5523, 1.0585, 1.4469, 1.4749, 1.4185, 1.3039,
                 1.4861, 1.465 , 1.6915, 1.0201, 1.3416, 1.084 , 1.2838, 1.2358,
                 1.1768, 1.2603, 0.9204, 1.0445, 1.03  , 1.2565, 0.7485, 1.5445,
                 1.0038, 1.3953, 1.4347, 1.0926, 1.2194, 1.1495])

In [112]: #standard-error
          #mu
          #observation

          se=stats.sem(x, axis=None, ddof=0)
          se

          mu=x.mean()

          #observed SR of buy & hold
          observation=0.3880

In [114]: z_value(mu,observation,se)

Out[114]: 22.45
```

A.2 D'Agostino and Pearson test for normal distribution of the Sharpe ratios produced by the random-choice-ensemble



```
In [797]: #test for normality
          # import scipy.stats as stats

          x = genfromtxt('data/book3.csv', delimiter=',')

          SRs_of_random_choice_ensemble=x

          SRs_of_random_choice_ensemble

Out[797]: array([1.4337, 1.3327, 1.16  , 1.5606, 0.8808, 1.6891, 1.2629, 1.3346,
                 1.2473, 1.0366, 1.5943, 1.178 , 1.3985, 1.404 , 1.2926, 1.7552,
                 1.0486, 1.337 , 1.4085, 1.8399, 1.2051, 1.2875, 0.9267, 1.197 ,
                 1.2949, 1.5583, 0.757 , 1.2925, 1.6518, 0.5695])

In [798]: stats.normaltest(x)

Out[798]: NormaltestResult(statistic=1.8026670592751386, pvalue=0.4060278483906372)
```

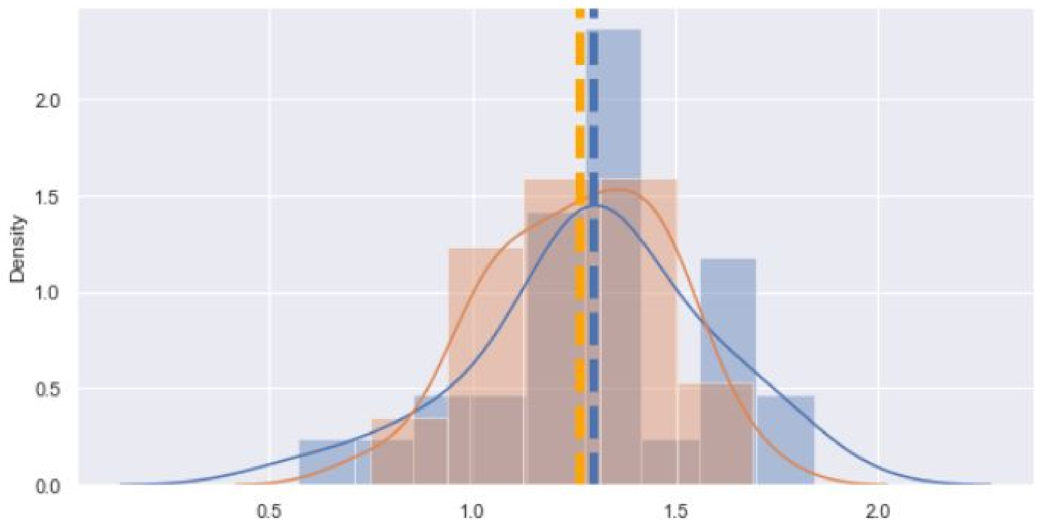
A.2.1 t-Test of Sharpe ratios of original ensemble versus a random-choice ensemble



```
In [198]: x = genfromtxt('data/book1.csv', delimiter=',')

In [199]: x

Out[199]: array([1.4478, 1.3059, 1.5523, 1.0585, 1.4469, 1.4749, 1.4185, 1.3039,
                 1.4861, 1.465 , 1.6915, 1.0201, 1.3416, 1.084 , 1.2838, 1.2358,
                 1.1768, 1.2603, 0.9204, 1.0445, 1.03  , 1.2565, 0.7485, 1.5445,
                 1.0038, 1.3953, 1.4347, 1.0926, 1.2194, 1.1495])

In [200]: x.mean()

Out[200]: 1.2631133333333333

In [201]: y=genfromtxt('data/book2.csv', delimiter=',')

In [202]: y

Out[202]: array([1.4337, 1.3327, 1.16  , 1.5606, 0.8808, 1.6891, 1.2629, 1.3346,
                 1.2473, 1.0366, 1.5943, 1.178 , 1.3985, 1.404 , 1.2926, 1.7552,
                 1.0486, 1.337 , 1.4085, 1.8399, 1.2051, 1.2875, 0.9267, 1.197 ,
                 1.2949, 1.5583, 0.757 , 1.2925, 1.6518, 0.5695])

In [203]: y.mean()

Out[203]: 1.2978399999999999
```

Specifics:

```
In [204]: original=x
          randomChoice=y
```



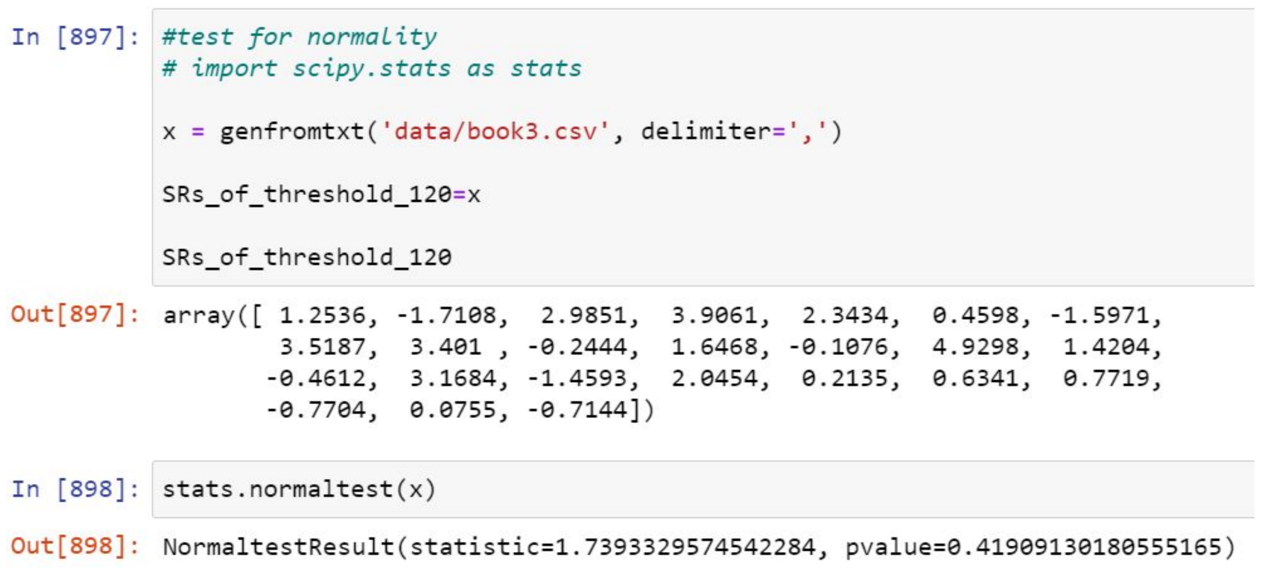
```
In [210]: def compare_2_groups(arr_1, arr_2, alpha, sample_size):
              stat, p=ttest_ind(arr_1, arr_2)
              print("statistics=%.3f,p=%.3f" %(stat,p))
              if p>alpha:
                  print("same distributiion (fail to reject H0)")
              else:
                  print("different distribution (reject H0)")
```

```
In [211]: sample_size=30
          alpha=0.05
          compare_2_groups(x,y, alpha, sample_size)
```

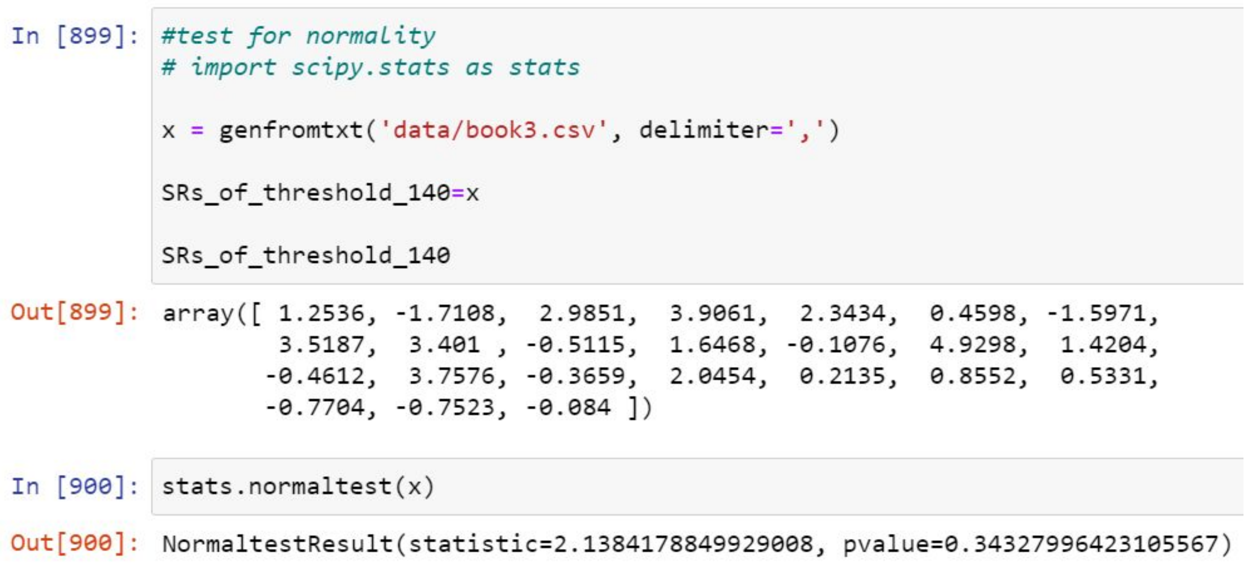
```
statistics=-0.529,p=0.599
same distributiion (fail to reject H0)
```

A.3 D'Agostino and Pearson test for normal distribution of the Sharpe ratios produced by the threshold levels 120 and 140

By splitting the period from 5th January 2010 to 6th January 2016, and running performance backtests for each quarter we get the following Sharpe ratios:

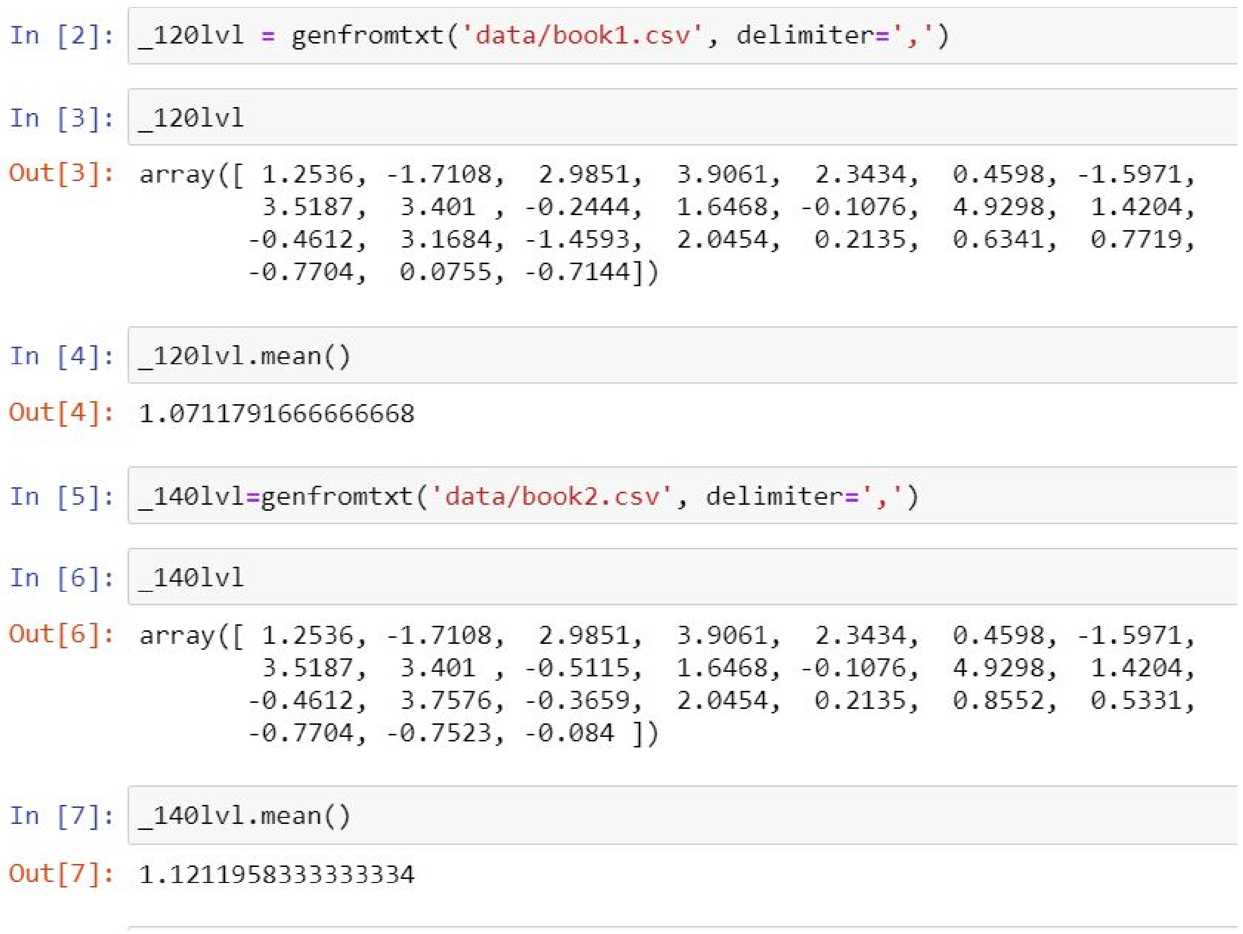


```
In [897]: #test for normality
          # import scipy.stats as stats

          x = genfromtxt('data/book3.csv', delimiter=',')

          SRs_of_threshold_120=x

          SRs_of_threshold_120

Out[897]: array([ 1.2536, -1.7108,  2.9851,  3.9061,  2.3434,  0.4598, -1.5971,
                  3.5187,  3.401 , -0.2444,  1.6468, -0.1076,  4.9298,  1.4204,
                 -0.4612,  3.1684, -1.4593,  2.0454,  0.2135,  0.6341,  0.7719,
                 -0.7704,  0.0755, -0.7144])

In [898]: stats.normaltest(x)

Out[898]: NormaltestResult(statistic=1.7393329574542284, pvalue=0.41909130180555165)
```

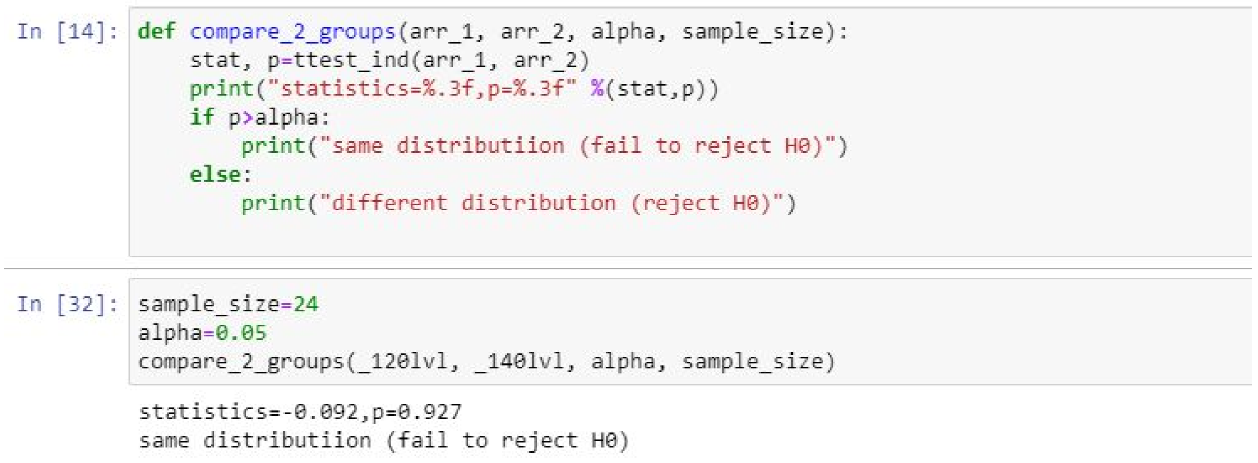


```
In [899]: #test for normality
          # import scipy.stats as stats

          x = genfromtxt('data/book3.csv', delimiter=',')

          SRs_of_threshold_140=x

          SRs_of_threshold_140

Out[899]: array([ 1.2536, -1.7108,  2.9851,  3.9061,  2.3434,  0.4598, -1.5971,
                  3.5187,  3.401 , -0.5115,  1.6468, -0.1076,  4.9298,  1.4204,
                 -0.4612,  3.7576, -0.3659,  2.0454,  0.2135,  0.8552,  0.5331,
                 -0.7704, -0.7523, -0.084 ])

In [900]: stats.normaltest(x)

Out[900]: NormaltestResult(statistic=2.1384178849929008, pvalue=0.34327996423105567)
```

A.3.1 t-Test of Sharpe ratios with turbulence threshold at 120 and 140 to determine the best turbulence threshold level

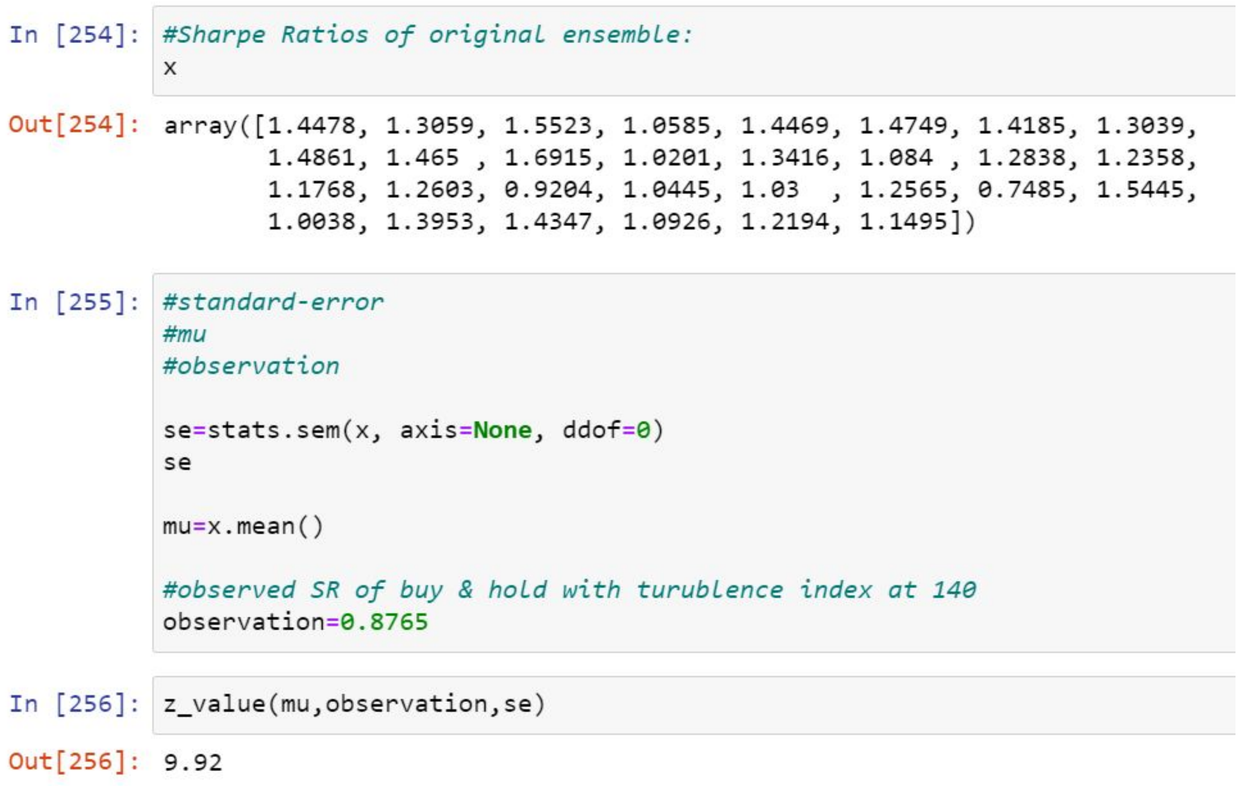


```
In [2]: _120lvl = genfromtxt('data/book1.csv', delimiter=',')

In [3]: _120lvl

Out[3]: array([ 1.2536, -1.7108,  2.9851,  3.9061,  2.3434,  0.4598, -1.5971,
                3.5187,  3.401 , -0.2444,  1.6468, -0.1076,  4.9298,  1.4204,
               -0.4612,  3.1684, -1.4593,  2.0454,  0.2135,  0.6341,  0.7719,
               -0.7704,  0.0755, -0.7144])

In [4]: _120lvl.mean()

Out[4]: 1.0711791666666668

In [5]: _140lvl=genfromtxt('data/book2.csv', delimiter=',')

In [6]: _140lvl

Out[6]: array([ 1.2536, -1.7108,  2.9851,  3.9061,  2.3434,  0.4598, -1.5971,
                3.5187,  3.401 , -0.5115,  1.6468, -0.1076,  4.9298,  1.4204,
               -0.4612,  3.7576, -0.3659,  2.0454,  0.2135,  0.8552,  0.5331,
               -0.7704, -0.7523, -0.084 ])

In [7]: _140lvl.mean()

Out[7]: 1.1211958333333334
```



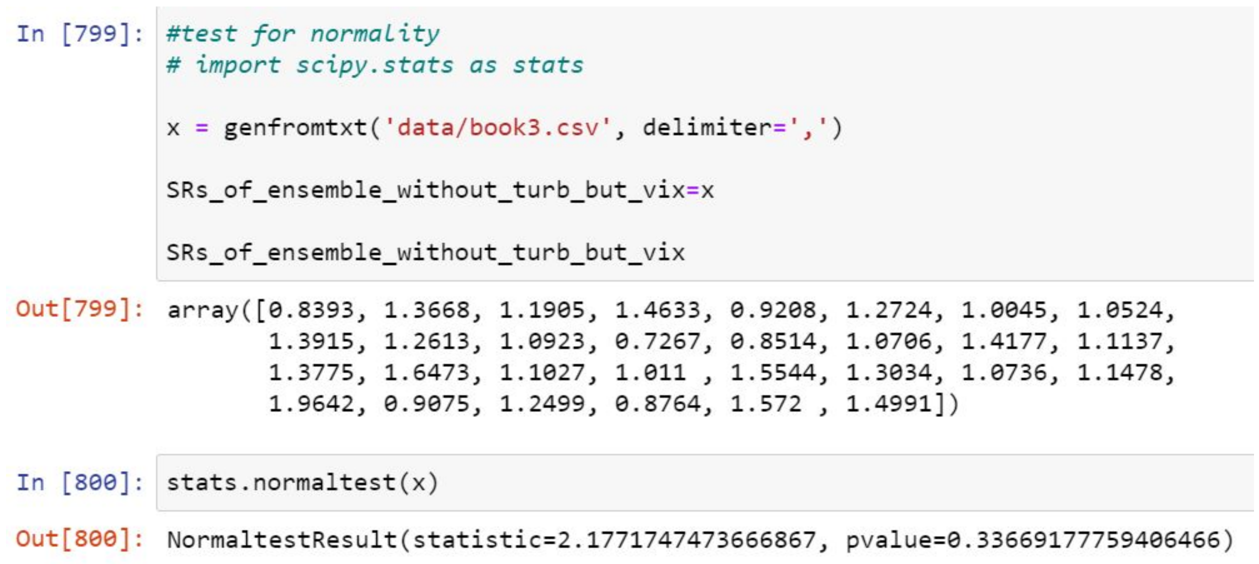
```
In [14]: def compare_2_groups(arr_1, arr_2, alpha, sample_size):
             stat, p=ttest_ind(arr_1, arr_2)
             print("statistics=%.3f,p=%.3f" %(stat,p))
             if p>alpha:
                 print("same distributiion (fail to reject H0)")
             else:
                 print("different distribution (reject H0)")

In [32]: sample_size=24
         alpha=0.05
         compare_2_groups(_120lvl, _140lvl, alpha, sample_size)

         statistics=-0.092,p=0.927
         same distributiion (fail to reject H0)
```

A.3.2 z-Value of buy and hold strategy with turbulence index



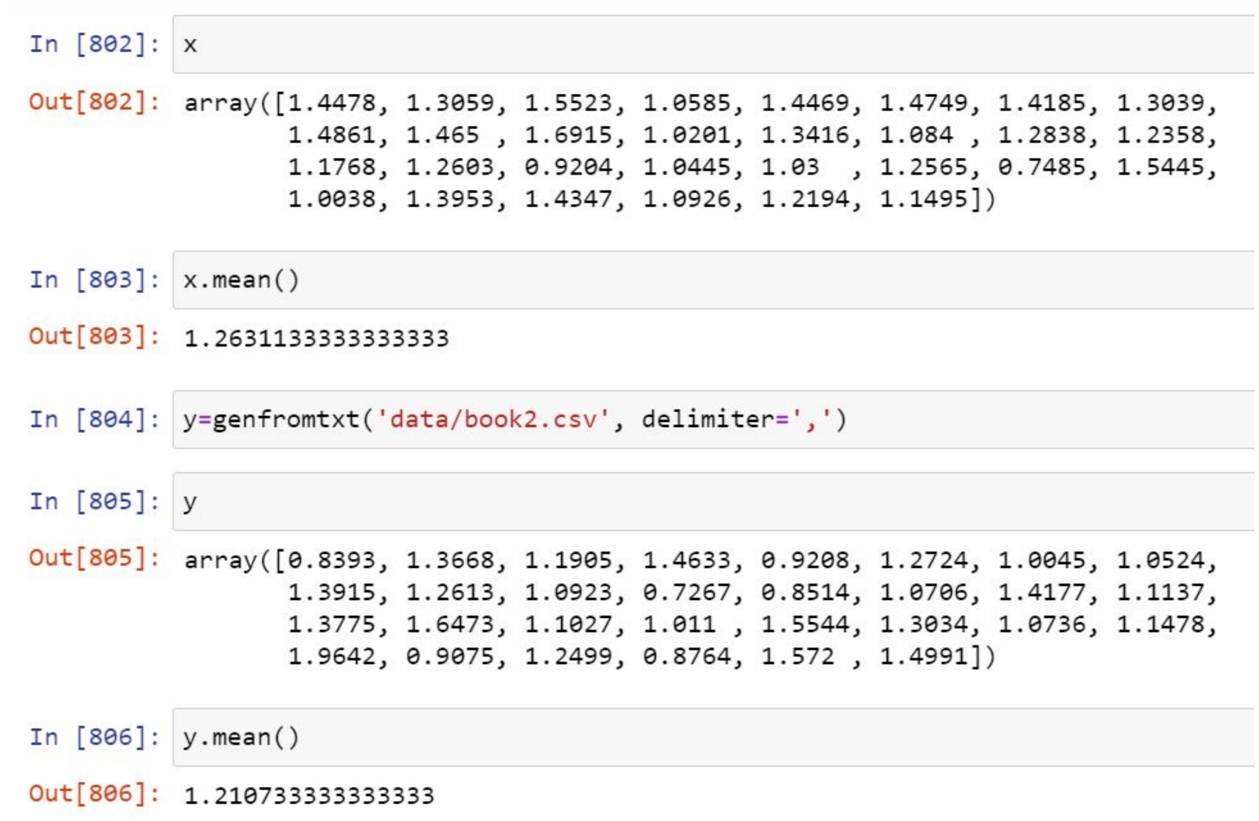
```
In [254]: #Sharpe Ratios of original ensemble:
          x

Out[254]: array([1.4478, 1.3059, 1.5523, 1.0585, 1.4469, 1.4749, 1.4185, 1.3039,
                 1.4861, 1.465 , 1.6915, 1.0201, 1.3416, 1.084 , 1.2838, 1.2358,
                 1.1768, 1.2603, 0.9204, 1.0445, 1.03  , 1.2565, 0.7485, 1.5445,
                 1.0038, 1.3953, 1.4347, 1.0926, 1.2194, 1.1495])

In [255]: #standard-error
          #mu
          #observation

          se=stats.sem(x, axis=None, ddof=0)
          se

          mu=x.mean()

          #observed SR of buy & hold with turublence index at 140
          observation=0.8765

In [256]: z_value(mu,observation,se)

Out[256]: 9.92
```

A.4 D'Agostino and Pearson test for normal distribution of the Sharpe ratios produced by the ensemble with VIX but no turbulence index

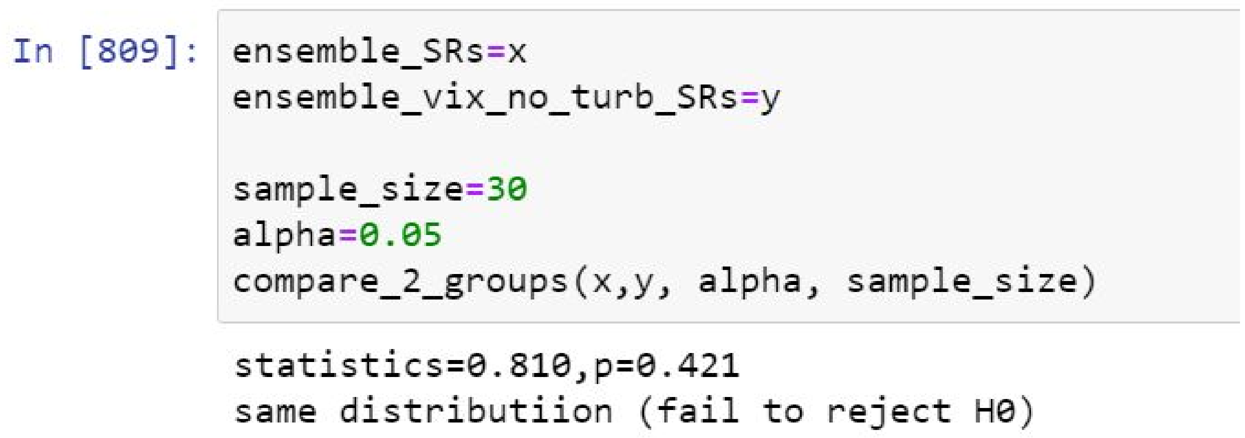


```
In [799]: #test for normality
          # import scipy.stats as stats

          x = genfromtxt('data/book3.csv', delimiter=',')

          SRs_of_ensemble_without_turb_but_vix=x

          SRs_of_ensemble_without_turb_but_vix

Out[799]: array([0.8393, 1.3668, 1.1905, 1.4633, 0.9208, 1.2724, 1.0045, 1.0524,
                 1.3915, 1.2613, 1.0923, 0.7267, 0.8514, 1.0706, 1.4177, 1.1137,
                 1.3775, 1.6473, 1.1027, 1.011 , 1.5544, 1.3034, 1.0736, 1.1478,
                 1.9642, 0.9075, 1.2499, 0.8764, 1.572 , 1.4991])

In [800]: stats.normaltest(x)

Out[800]: NormaltestResult(statistic=2.177174747366867, pvalue=0.33669177759406466)
```

A.4.1 t-Test of Sharpe ratios of original ensemble versus an ensemble without the turbulence index but VIX data

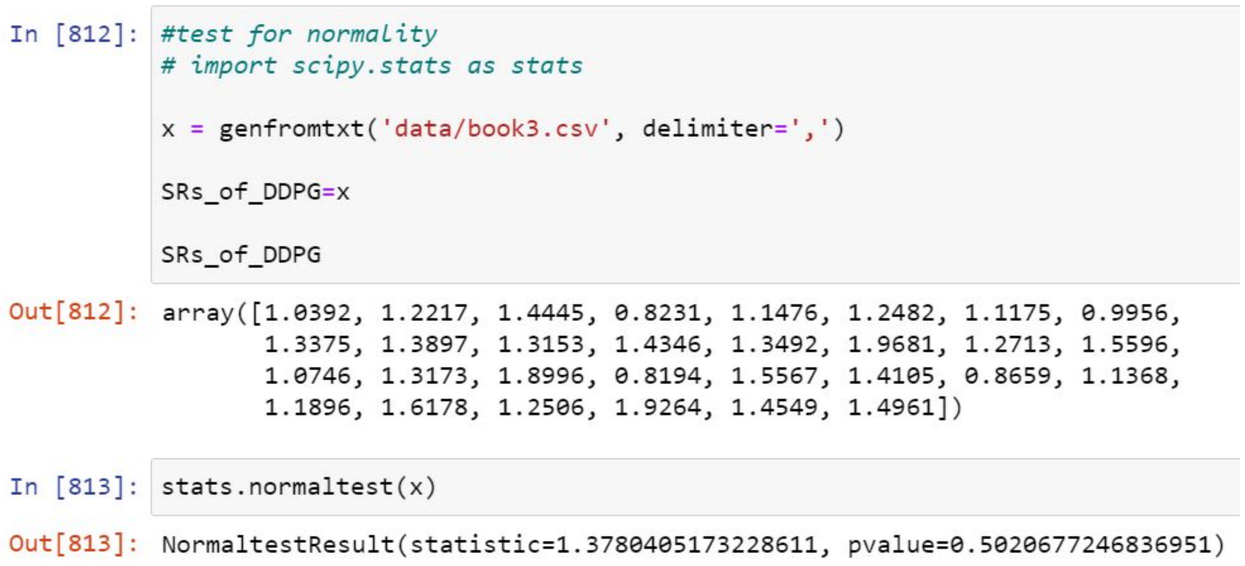


```
In [802]: x

Out[802]: array([1.4478, 1.3059, 1.5523, 1.0585, 1.4469, 1.4749, 1.4185, 1.3039,
                 1.4861, 1.465 , 1.6915, 1.0201, 1.3416, 1.084 , 1.2838, 1.2358,
                 1.1768, 1.2603, 0.9204, 1.0445, 1.03  , 1.2565, 0.7485, 1.5445,
                 1.0038, 1.3953, 1.4347, 1.0926, 1.2194, 1.1495])

In [803]: x.mean()

Out[803]: 1.2631133333333333

In [804]: y=genfromtxt('data/book2.csv', delimiter=',')

In [805]: y

Out[805]: array([0.8393, 1.3668, 1.1905, 1.4633, 0.9208, 1.2724, 1.0045, 1.0524,
                 1.3915, 1.2613, 1.0923, 0.7267, 0.8514, 1.0706, 1.4177, 1.1137,
                 1.3775, 1.6473, 1.1027, 1.011 , 1.5544, 1.3034, 1.0736, 1.1478,
                 1.9642, 0.9075, 1.2499, 0.8764, 1.572 , 1.4991])

In [806]: y.mean()

Out[806]: 1.210733333333333
```



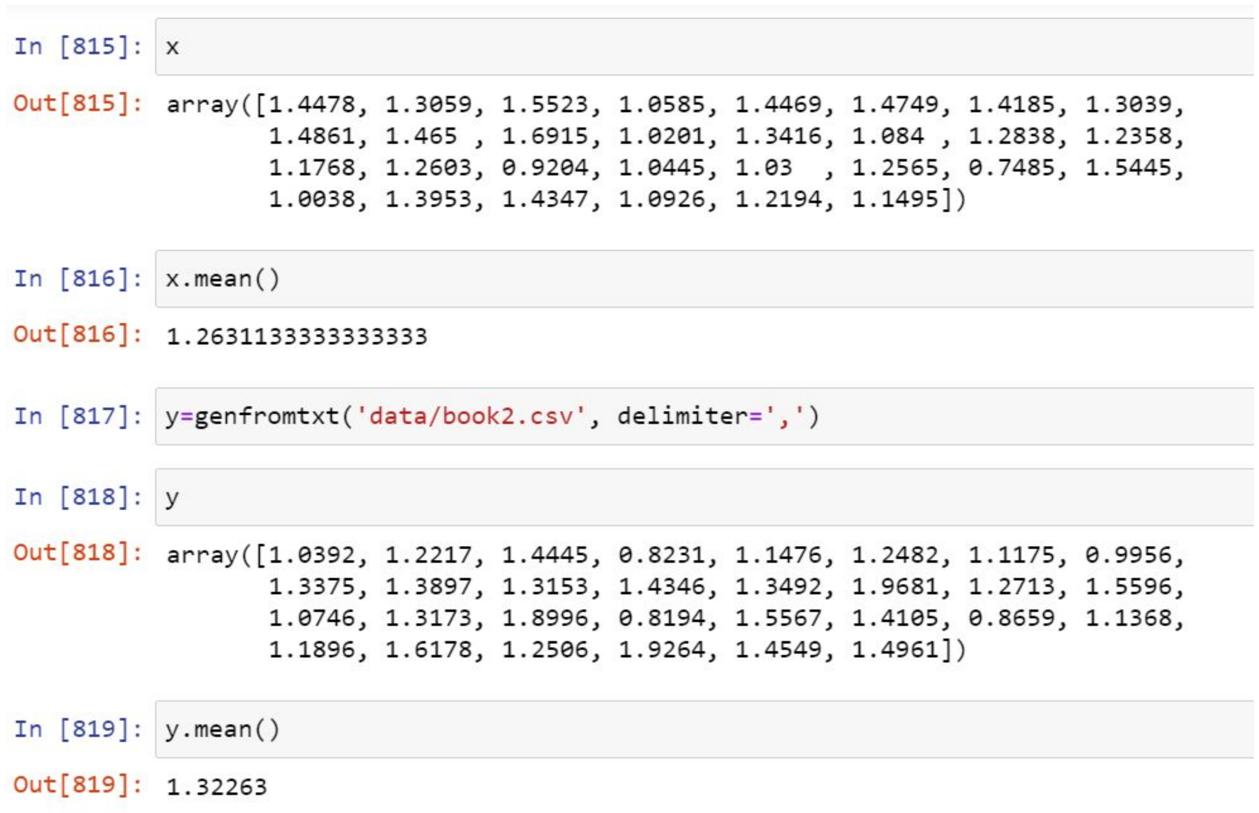
```
In [809]: ensemble_SRs=x
          ensemble_vix_no_turb_SRs=y

          sample_size=30
          alpha=0.05
          compare_2_groups(x,y, alpha, sample_size)

          statistics=0.810,p=0.421
          same distributiion (fail to reject H0)
```

A.5 D'Agostino and Pearson test for normal distribution of the Sharpe ratios produced by the DDPG agent

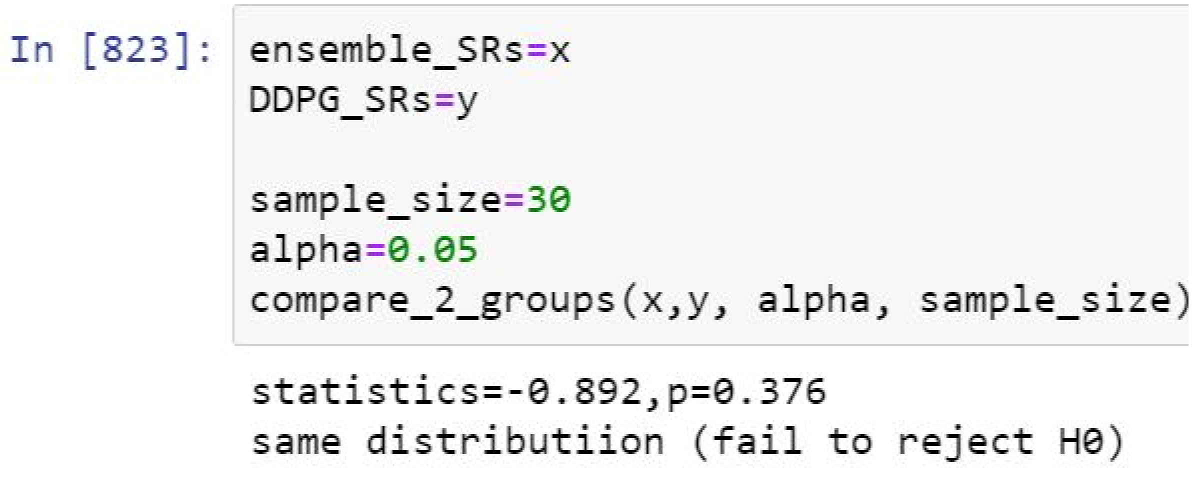


```
In [812]: #test for normality
          # import scipy.stats as stats

          x = genfromtxt('data/book3.csv', delimiter=',')

          SRs_of_DDPG=x

          SRs_of_DDPG

Out[812]: array([1.0392, 1.2217, 1.4445, 0.8231, 1.1476, 1.2482, 1.1175, 0.9956,
                 1.3375, 1.3897, 1.3153, 1.4346, 1.3492, 1.9681, 1.2713, 1.5596,
                 1.0746, 1.3173, 1.8996, 0.8194, 1.5567, 1.4105, 0.8659, 1.1368,
                 1.1896, 1.6178, 1.2506, 1.9264, 1.4549, 1.4961])

In [813]: stats.normaltest(x)

Out[813]: NormaltestResult(statistic=1.3780405173228611, pvalue=0.5020677246836951)
```

A.5.1 t-Test of original ensemble’s Sharpe ratios versus DDPG’s Sharpe ratios

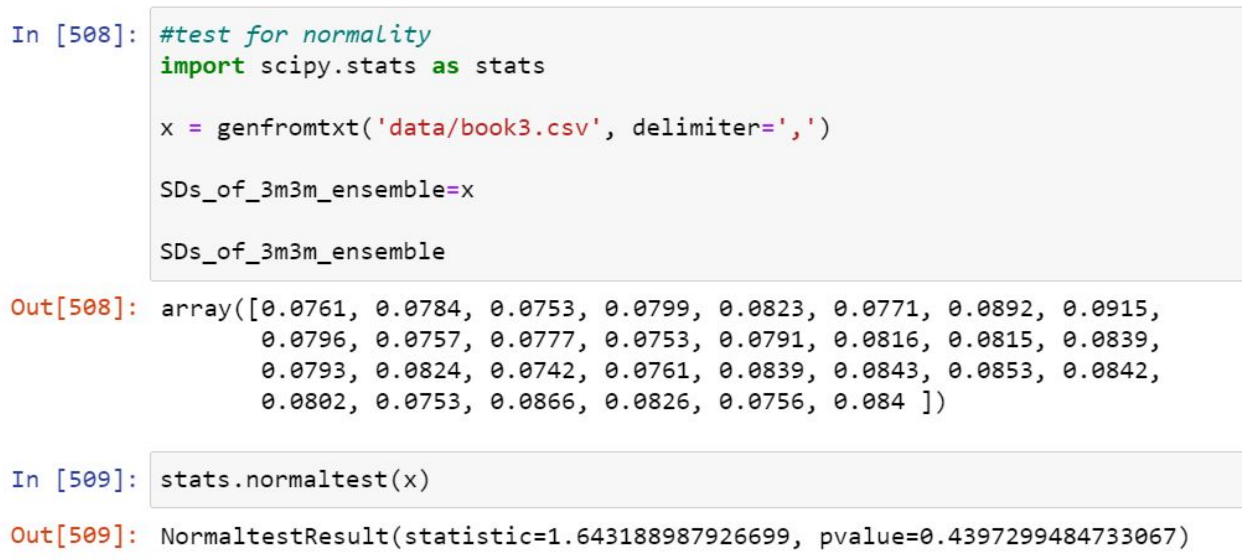


```
In [815]: x

Out[815]: array([1.4478, 1.3059, 1.5523, 1.0585, 1.4469, 1.4749, 1.4185, 1.3039,
                 1.4861, 1.465 , 1.6915, 1.0201, 1.3416, 1.084 , 1.2838, 1.2358,
                 1.1768, 1.2603, 0.9204, 1.0445, 1.03  , 1.2565, 0.7485, 1.5445,
                 1.0038, 1.3953, 1.4347, 1.0926, 1.2194, 1.1495])

In [816]: x.mean()

Out[816]: 1.2631133333333333

In [817]: y=genfromtxt('data/book2.csv', delimiter=',')

In [818]: y

Out[818]: array([1.0392, 1.2217, 1.4445, 0.8231, 1.1476, 1.2482, 1.1175, 0.9956,
                 1.3375, 1.3897, 1.3153, 1.4346, 1.3492, 1.9681, 1.2713, 1.5596,
                 1.0746, 1.3173, 1.8996, 0.8194, 1.5567, 1.4105, 0.8659, 1.1368,
                 1.1896, 1.6178, 1.2506, 1.9264, 1.4549, 1.4961])

In [819]: y.mean()

Out[819]: 1.32263
```

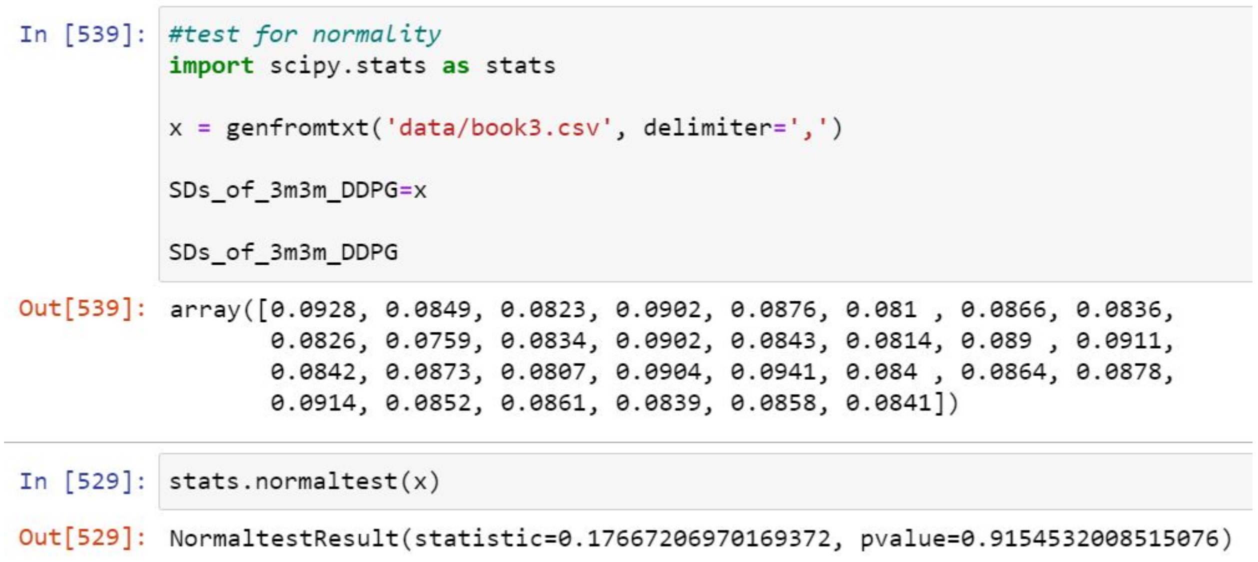


```
In [823]: ensemble_SRs=x
          DDPG_SRs=y

          sample_size=30
          alpha=0.05
          compare_2_groups(x,y, alpha, sample_size)

          statistics=-0.892,p=0.376
          same distributiion (fail to reject H0)
```

A.6 D'Agostino and Pearson test for normal distribution of standard-deviations of returns of ensemble and DDPG agent

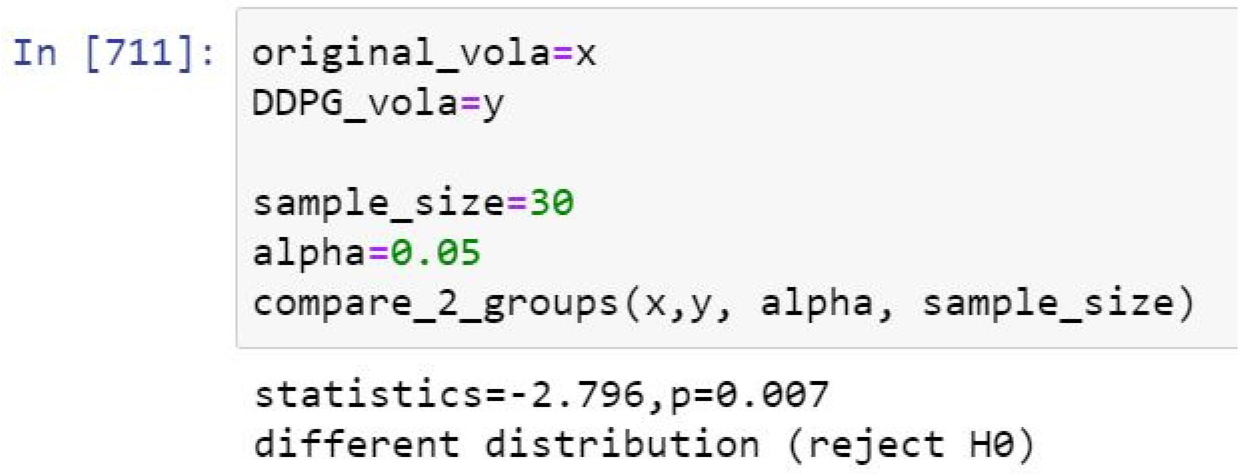


```
In [508]: #test for normality
          import scipy.stats as stats

          x = genfromtxt('data/book3.csv', delimiter=',')

          SDs_of_3m3m_ensemble=x

          SDs_of_3m3m_ensemble

Out[508]: array([0.0761, 0.0784, 0.0753, 0.0799, 0.0823, 0.0771, 0.0892, 0.0915,
                 0.0796, 0.0757, 0.0777, 0.0753, 0.0791, 0.0816, 0.0815, 0.0839,
                 0.0793, 0.0824, 0.0742, 0.0761, 0.0839, 0.0843, 0.0853, 0.0842,
                 0.0802, 0.0753, 0.0866, 0.0826, 0.0756, 0.084 ])

In [509]: stats.normaltest(x)

Out[509]: NormaltestResult(statistic=1.643188987926699, pvalue=0.4397299484733067)
```



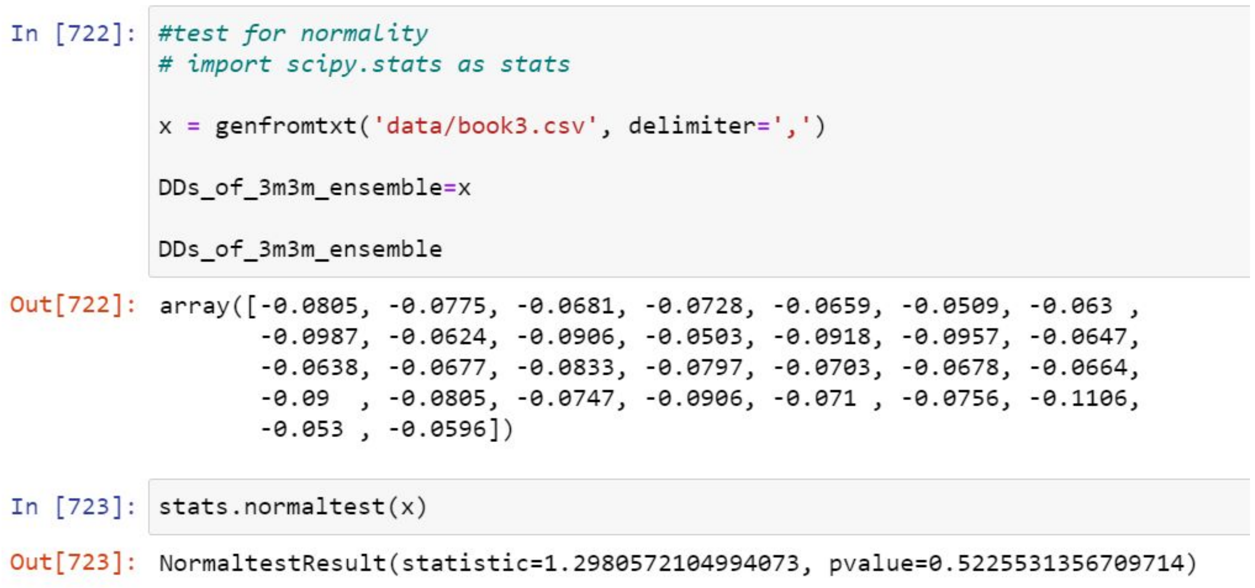
```
In [539]: #test for normality
          import scipy.stats as stats

          x = genfromtxt('data/book3.csv', delimiter=',')

          SDs_of_3m3m_DDPG=x

          SDs_of_3m3m_DDPG

Out[539]: array([0.0928, 0.0849, 0.0823, 0.0902, 0.0876, 0.081 , 0.0866, 0.0836,
                 0.0826, 0.0759, 0.0834, 0.0902, 0.0843, 0.0814, 0.089 , 0.0911,
                 0.0842, 0.0873, 0.0807, 0.0904, 0.0941, 0.084 , 0.0864, 0.0878,
                 0.0914, 0.0852, 0.0861, 0.0839, 0.0858, 0.0841])

In [529]: stats.normaltest(x)

Out[529]: NormaltestResult(statistic=0.17667206970169372, pvalue=0.9154532008515076)
```

A.6.1 t-Test of original ensemble’s volatility (=standard deviation) of returns versus DDPG’s volatility of returns

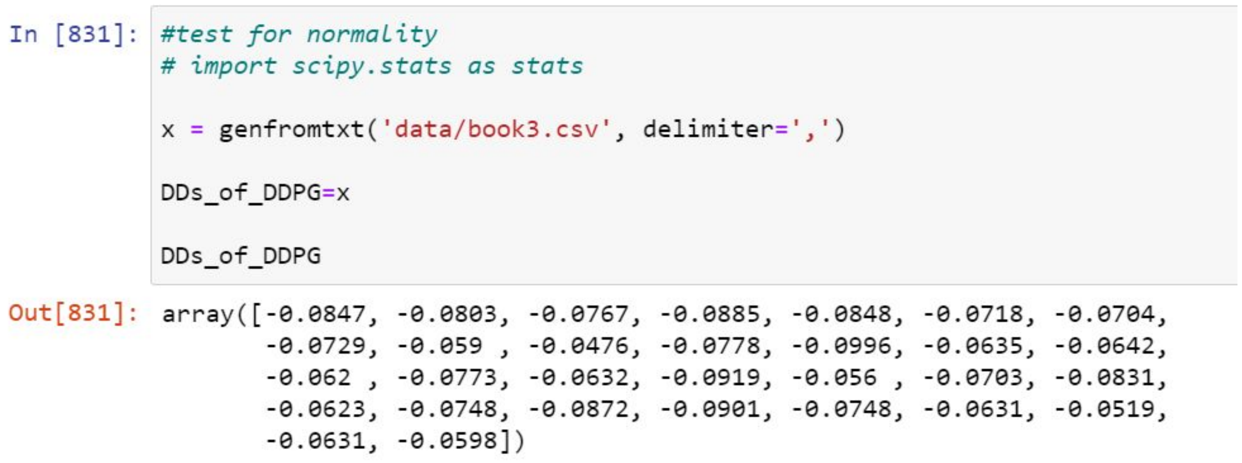


```
In [711]: original_vola=x
          DDPG_vola=y

          sample_size=30
          alpha=0.05
          compare_2_groups(x,y, alpha, sample_size)

          statistics=-2.796,p=0.007
          different distribution (reject H0)
```

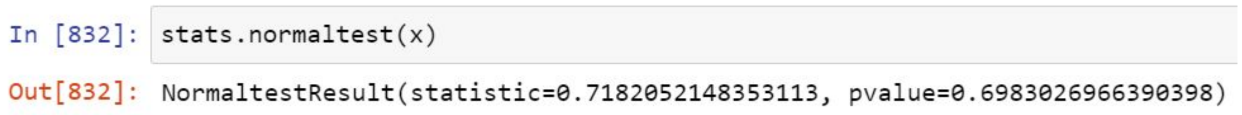
A.7 D’Agostino and Person test for normal distribution of max drawdowns of ensemble and DDPG agent



```
In [722]: #test for normality
          # import scipy.stats as stats

          x = genfromtxt('data/book3.csv', delimiter=',')

          DDs_of_3m3m_ensemble=x

          DDs_of_3m3m_ensemble

Out[722]: array([-0.0805, -0.0775, -0.0681, -0.0728, -0.0659, -0.0509, -0.063 ,
                 -0.0987, -0.0624, -0.0906, -0.0503, -0.0918, -0.0957, -0.0647,
                 -0.0638, -0.0677, -0.0833, -0.0797, -0.0703, -0.0678, -0.0664,
                 -0.09  , -0.0805, -0.0747, -0.0906, -0.071 , -0.0756, -0.1106,
                 -0.053 , -0.0596])

In [723]: stats.normaltest(x)

Out[723]: NormaltestResult(statistic=1.2980572104994073, pvalue=0.5225531356709714)
```

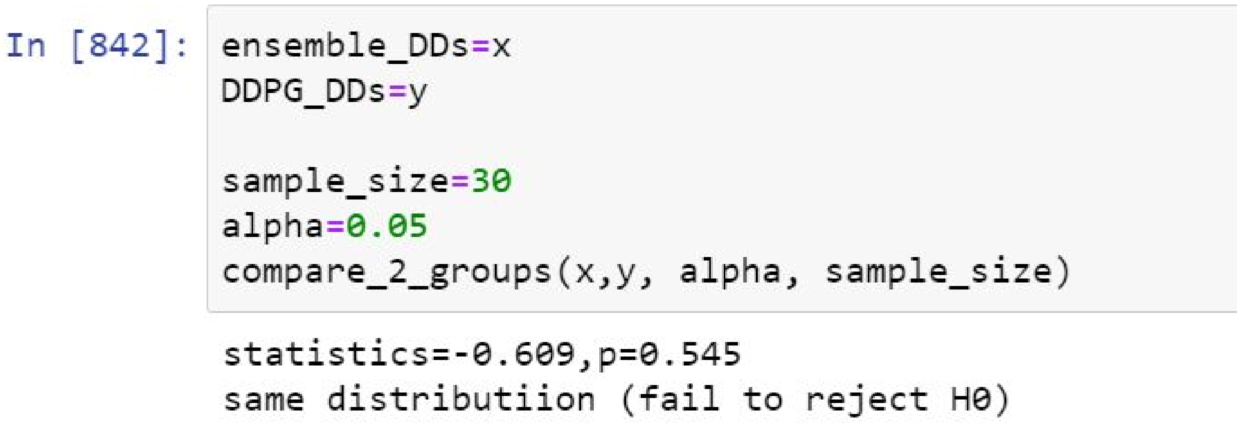


```
In [831]: #test for normality
          # import scipy.stats as stats

          x = genfromtxt('data/book3.csv', delimiter=',')

          DDs_of_DDPG=x

          DDs_of_DDPG

Out[831]: array([-0.0847, -0.0803, -0.0767, -0.0885, -0.0848, -0.0718, -0.0704,
                 -0.0729, -0.059 , -0.0476, -0.0778, -0.0996, -0.0635, -0.0642,
                 -0.062 , -0.0773, -0.0632, -0.0919, -0.056 , -0.0703, -0.0831,
                 -0.0623, -0.0748, -0.0872, -0.0901, -0.0748, -0.0631, -0.0519,
                 -0.0631, -0.0598])
```

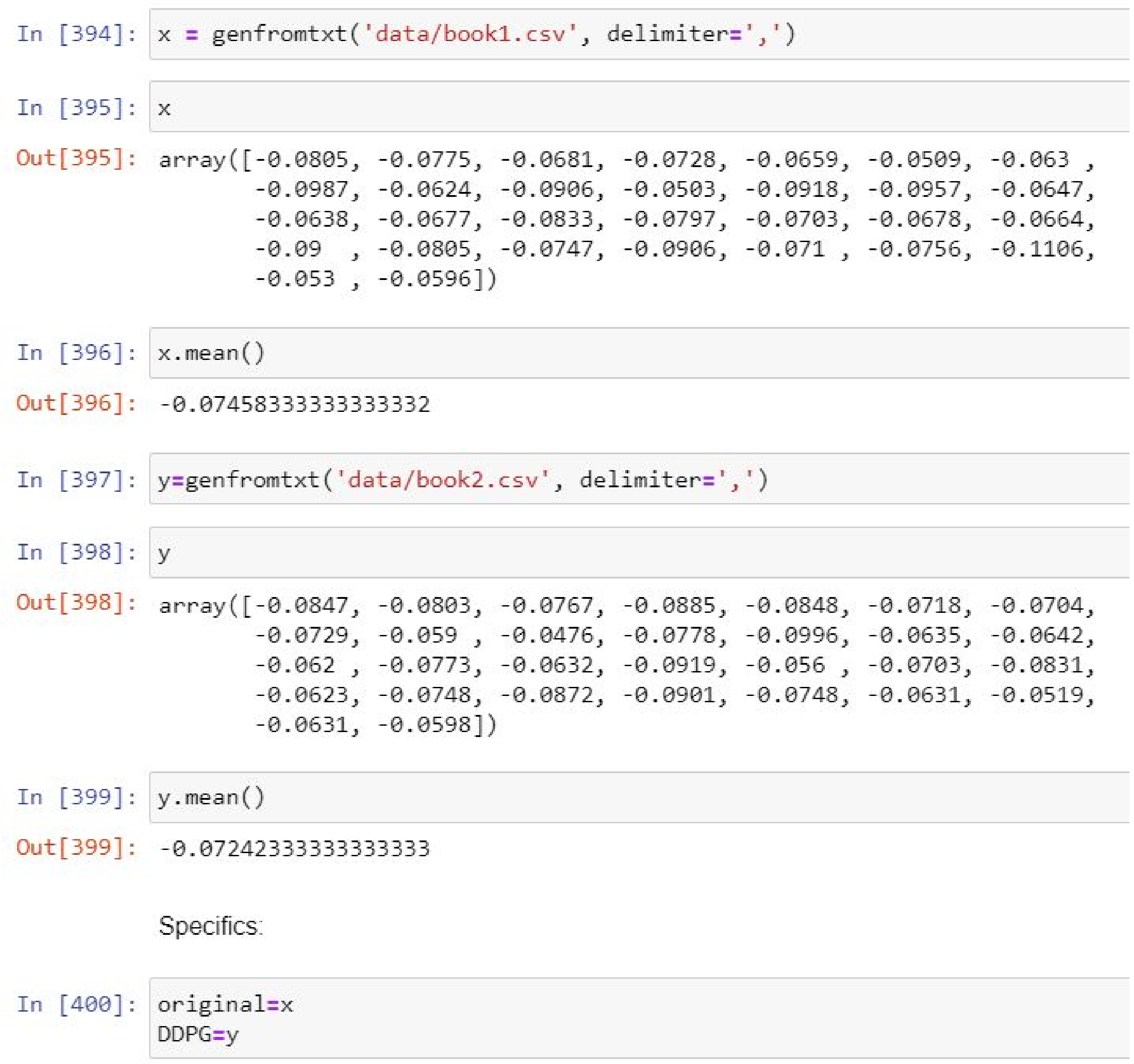


```
In [832]: stats.normaltest(x)

Out[832]: NormaltestResult(statistic=0.7182052148353113, pvalue=0.6983026966390398)
```

A.7.1 t-Test of max drawdowns of original ensemble versus DDPG agent



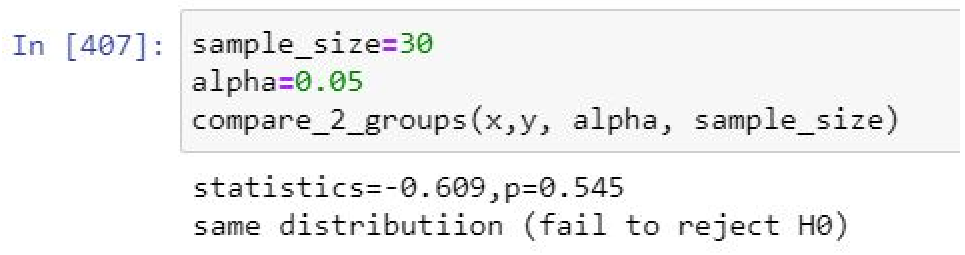
```
In [842]: ensemble_DDs=x
          DDPG_DDs=y

          sample_size=30
          alpha=0.05
          compare_2_groups(x,y, alpha, sample_size)

          statistics=-0.609,p=0.545
          same distributiion (fail to reject H0)
```



```
In [394]: x = genfromtxt('data/book1.csv', delimiter=',')

In [395]: x

Out[395]: array([-0.0805, -0.0775, -0.0681, -0.0728, -0.0659, -0.0509, -0.063 ,
                 -0.0987, -0.0624, -0.0906, -0.0503, -0.0918, -0.0957, -0.0647,
                 -0.0638, -0.0677, -0.0833, -0.0797, -0.0703, -0.0678, -0.0664,
                 -0.09  , -0.0805, -0.0747, -0.0906, -0.071 , -0.0756, -0.1106,
                 -0.053 , -0.0596])

In [396]: x.mean()

Out[396]: -0.07458333333333332

In [397]: y=genfromtxt('data/book2.csv', delimiter=',')

In [398]: y

Out[398]: array([-0.0847, -0.0803, -0.0767, -0.0885, -0.0848, -0.0718, -0.0704,
                 -0.0729, -0.059 , -0.0476, -0.0778, -0.0996, -0.0635, -0.0642,
                 -0.062 , -0.0773, -0.0632, -0.0919, -0.056 , -0.0703, -0.0831,
                 -0.0623, -0.0748, -0.0872, -0.0901, -0.0748, -0.0631, -0.0519,
                 -0.0631, -0.0598])

In [399]: y.mean()

Out[399]: -0.07242333333333333
```

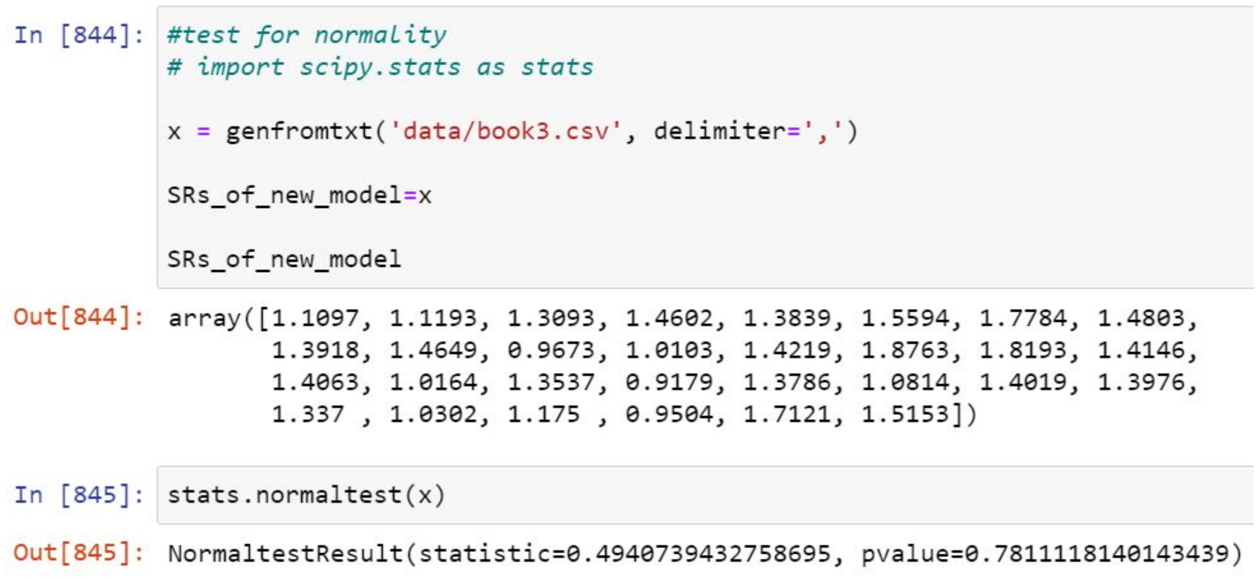
Specifics:

```
In [400]: original=x
          DDPG=y
```



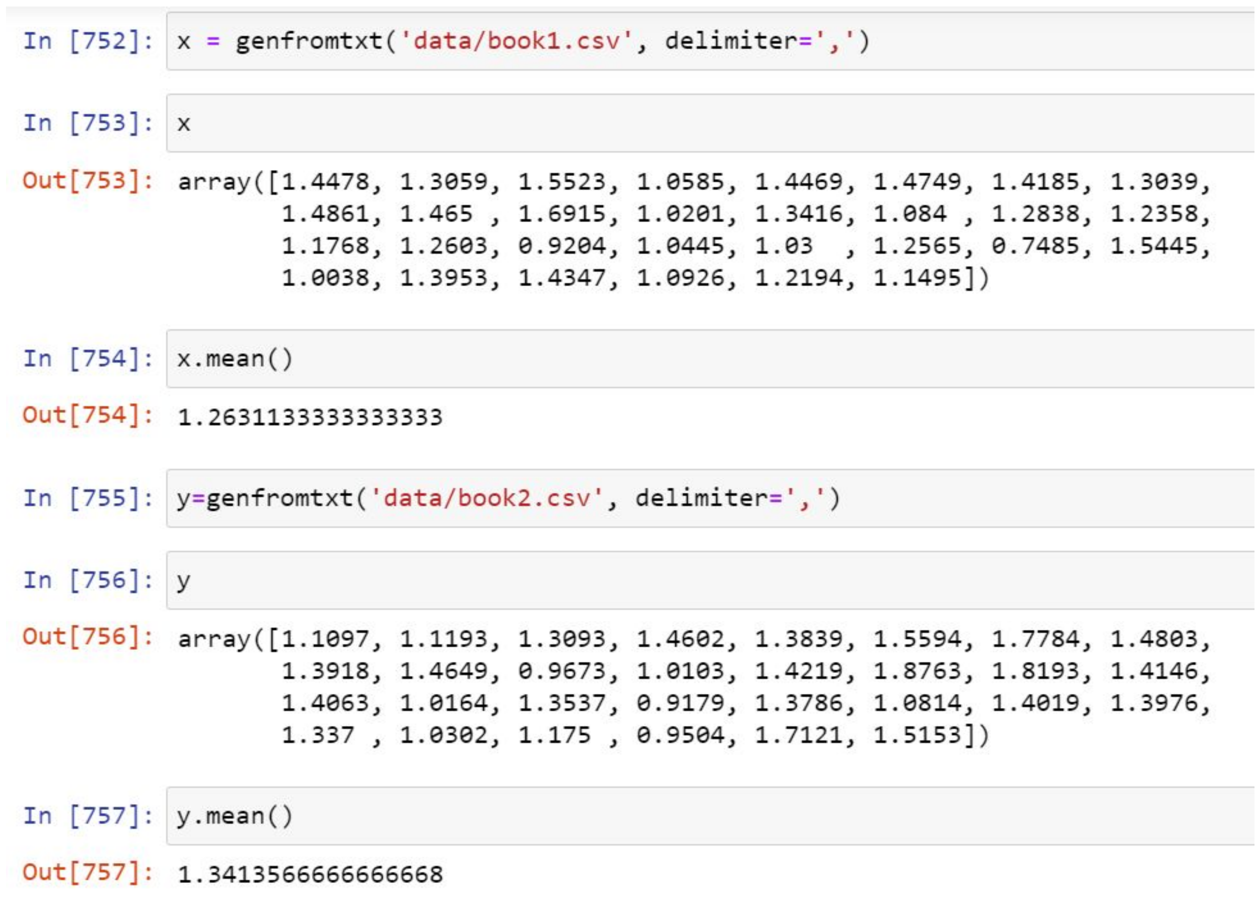
```
In [407]: sample_size=30
          alpha=0.05
          compare_2_groups(x,y, alpha, sample_size)

          statistics=-0.609,p=0.545
          same distributiion (fail to reject H0)
```

A.8 D'Agostino and Pearson test for normal distribution of Sharpe ratios of the new model

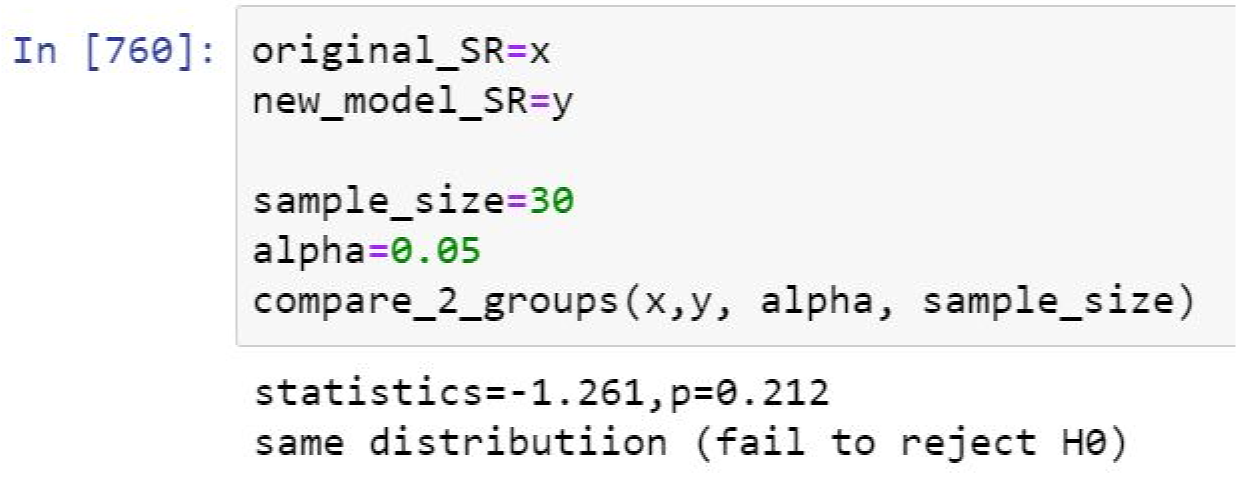


```
In [844]: #test for normality
          # import scipy.stats as stats

          x = genfromtxt('data/book3.csv', delimiter=',')

          SRs_of_new_model=x

          SRs_of_new_model

Out[844]: array([1.1097, 1.1193, 1.3093, 1.4602, 1.3839, 1.5594, 1.7784, 1.4803,
                 1.3918, 1.4649, 0.9673, 1.0103, 1.4219, 1.8763, 1.8193, 1.4146,
                 1.4063, 1.0164, 1.3537, 0.9179, 1.3786, 1.0814, 1.4019, 1.3976,
                 1.337 , 1.0302, 1.175 , 0.9504, 1.7121, 1.5153])

In [845]: stats.normaltest(x)

Out[845]: NormaltestResult(statistic=0.4940739432758695, pvalue=0.7811118140143439)
```

A.8.1 t-Test of original ensemble’s Sharpe ratios versus the new model’s Sharpe ratios

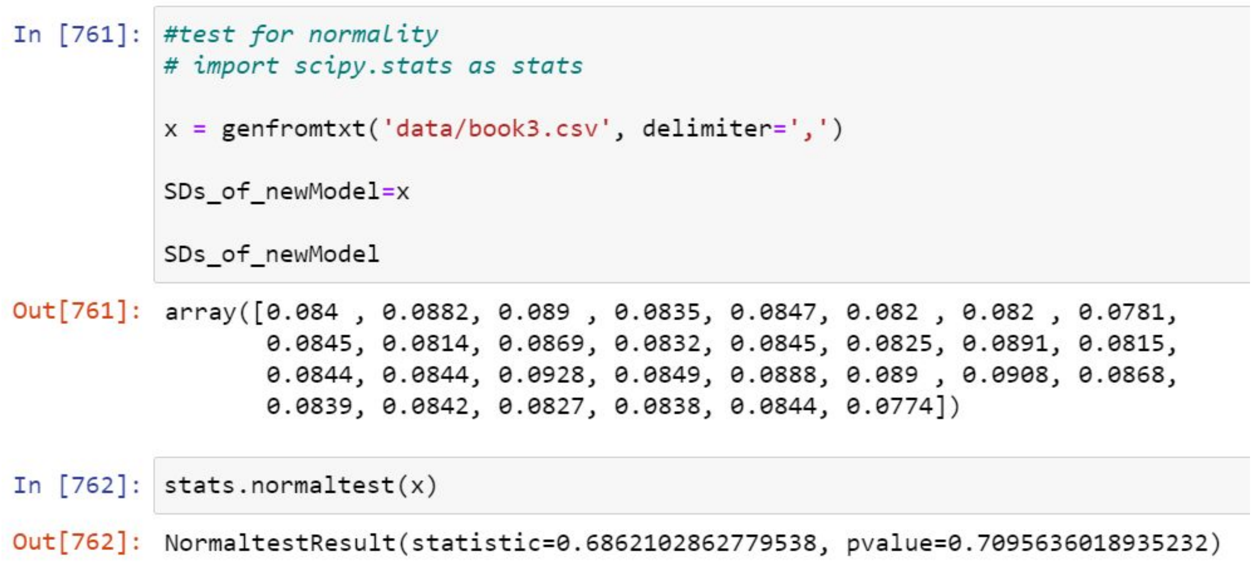


```
In [752]: x = genfromtxt('data/book1.csv', delimiter=',')

In [753]: x

Out[753]: array([1.4478, 1.3059, 1.5523, 1.0585, 1.4469, 1.4749, 1.4185, 1.3039,
                 1.4861, 1.465 , 1.6915, 1.0201, 1.3416, 1.084 , 1.2838, 1.2358,
                 1.1768, 1.2603, 0.9204, 1.0445, 1.03  , 1.2565, 0.7485, 1.5445,
                 1.0038, 1.3953, 1.4347, 1.0926, 1.2194, 1.1495])

In [754]: x.mean()

Out[754]: 1.2631133333333333

In [755]: y=genfromtxt('data/book2.csv', delimiter=',')

In [756]: y

Out[756]: array([1.1097, 1.1193, 1.3093, 1.4602, 1.3839, 1.5594, 1.7784, 1.4803,
                 1.3918, 1.4649, 0.9673, 1.0103, 1.4219, 1.8763, 1.8193, 1.4146,
                 1.4063, 1.0164, 1.3537, 0.9179, 1.3786, 1.0814, 1.4019, 1.3976,
                 1.337 , 1.0302, 1.175 , 0.9504, 1.7121, 1.5153])

In [757]: y.mean()

Out[757]: 1.3413566666666668
```

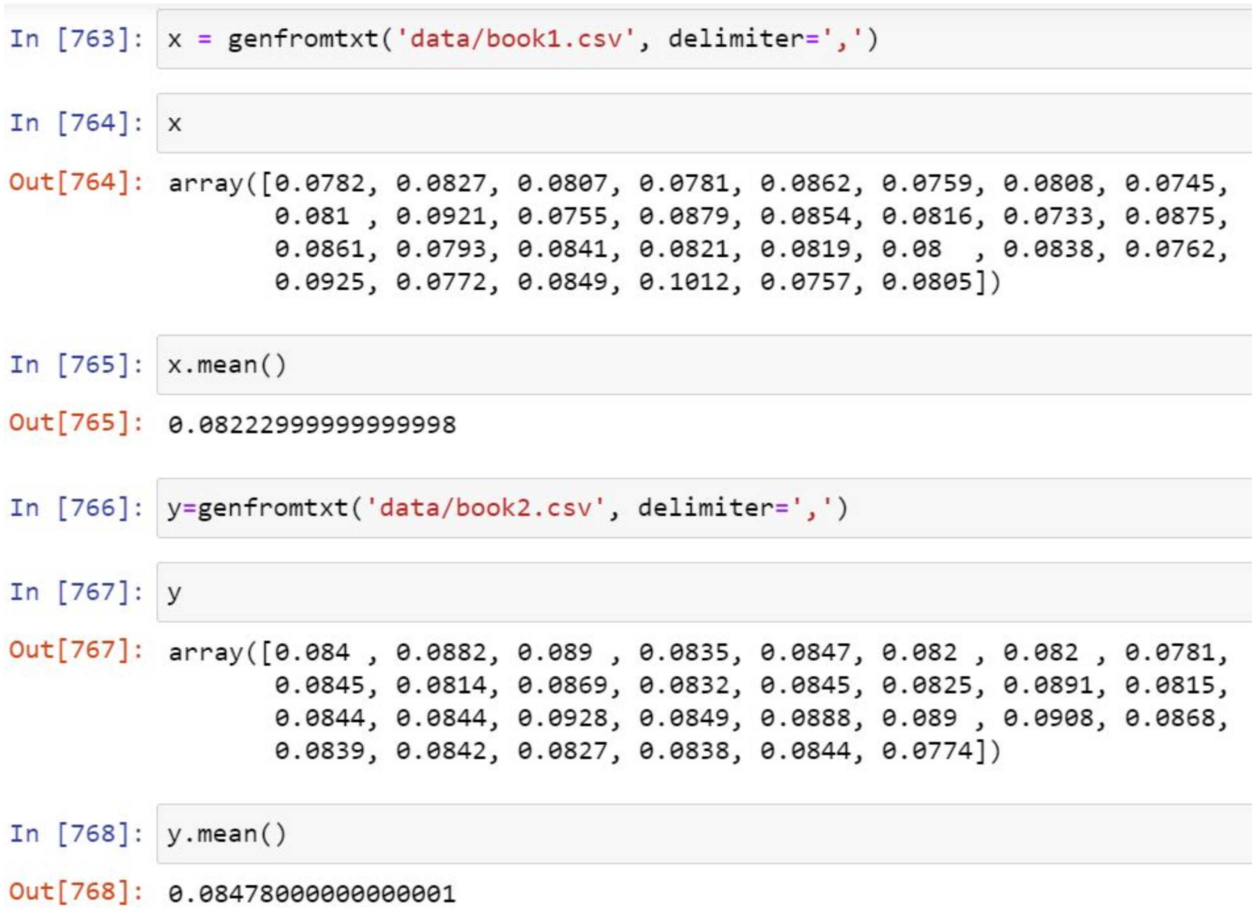


```
In [760]: original_SR=x
          new_model_SR=y

          sample_size=30
          alpha=0.05
          compare_2_groups(x,y, alpha, sample_size)

          statistics=-1.261,p=0.212
          same distributiion (fail to reject H0)
```

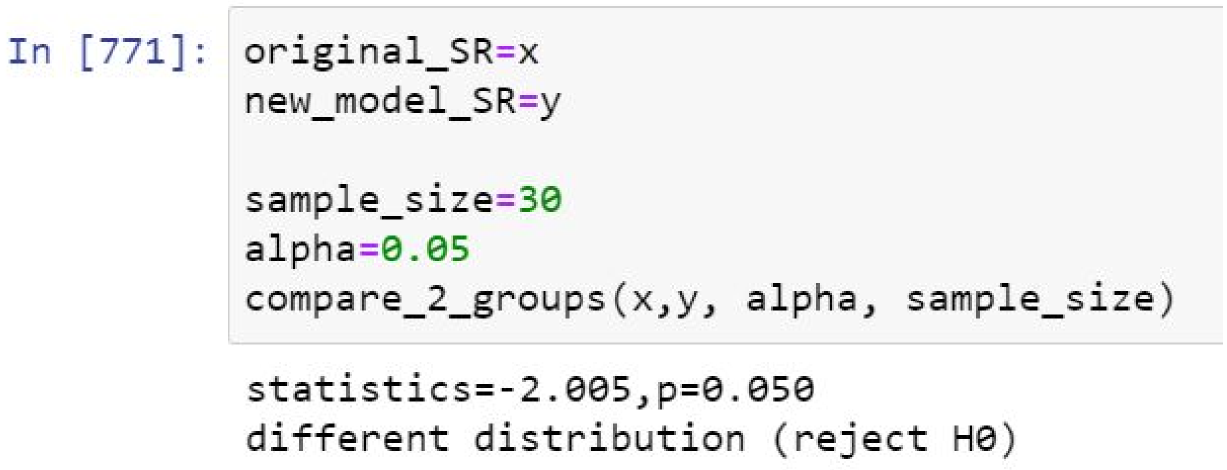
A.9 D'Agostino and Pearson test for normal distribution of standard-deviations of the new model


```
In [761]: #test for normality
          # import scipy.stats as stats

          x = genfromtxt('data/book3.csv', delimiter=',')

          SDs_of_newModel=x

          SDs_of_newModel

Out[761]: array([0.084 , 0.0882, 0.089 , 0.0835, 0.0847, 0.082 , 0.082 , 0.0781,
                 0.0845, 0.0814, 0.0869, 0.0832, 0.0845, 0.0825, 0.0891, 0.0815,
                 0.0844, 0.0844, 0.0928, 0.0849, 0.0888, 0.089 , 0.0908, 0.0868,
                 0.0839, 0.0842, 0.0827, 0.0838, 0.0844, 0.0774])

In [762]: stats.normaltest(x)

Out[762]: NormaltestResult(statistic=0.686210286277953 8, pvalue=0.7095636018935232)
```

A.9.1 t-Test of original ensemble’s volatility (=standard deviation) of returns versus the new model’s volatility of returns

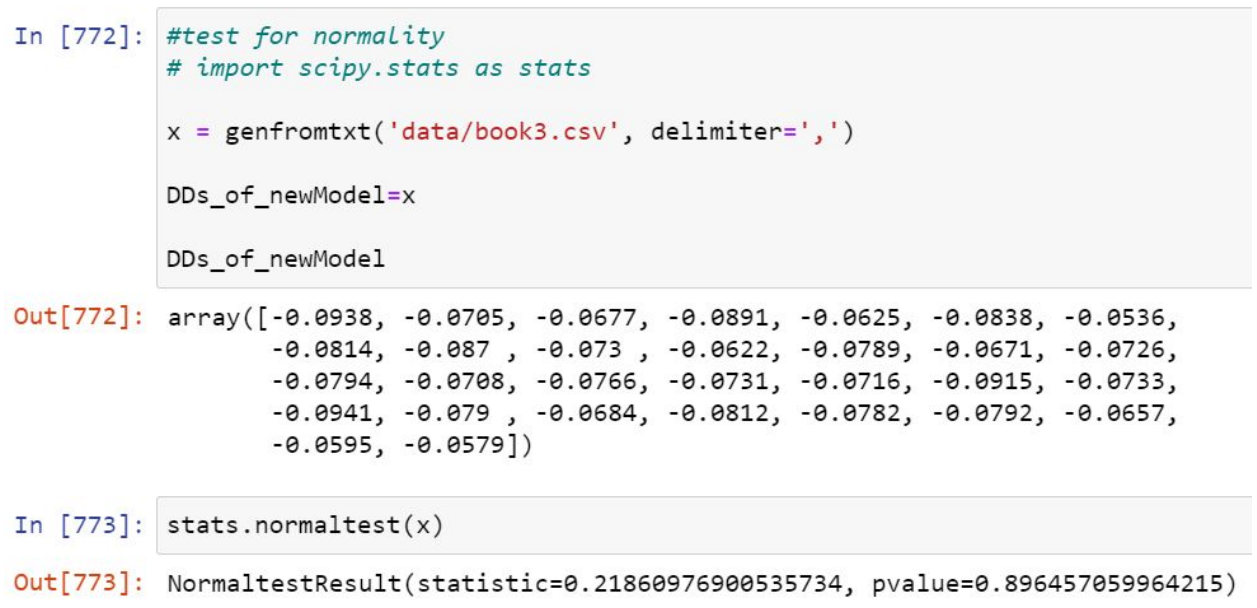


```
In [763]: x = genfromtxt('data/book1.csv', delimiter=',')

In [764]: x

Out[764]: array([0.0782, 0.0827, 0.0807, 0.0781, 0.0862, 0.0759, 0.0808, 0.0745,
                 0.081 , 0.0921, 0.0755, 0.0879, 0.0854, 0.0816, 0.0733, 0.0875,
                 0.0861, 0.0793, 0.0841, 0.0821, 0.0819, 0.08  , 0.0838, 0.0762,
                 0.0925, 0.0772, 0.0849, 0.1012, 0.0757, 0.0805])

In [765]: x.mean()

Out[765]: 0.08222999999999998

In [766]: y=genfromtxt('data/book2.csv', delimiter=',')

In [767]: y

Out[767]: array([0.084 , 0.0882, 0.089 , 0.0835, 0.0847, 0.082 , 0.082 , 0.0781,
                 0.0845, 0.0814, 0.0869, 0.0832, 0.0845, 0.0825, 0.0891, 0.0815,
                 0.0844, 0.0844, 0.0928, 0.0849, 0.0888, 0.089 , 0.0908, 0.0868,
                 0.0839, 0.0842, 0.0827, 0.0838, 0.0844, 0.0774])

In [768]: y.mean()

Out[768]: 0.08478000000000001
```



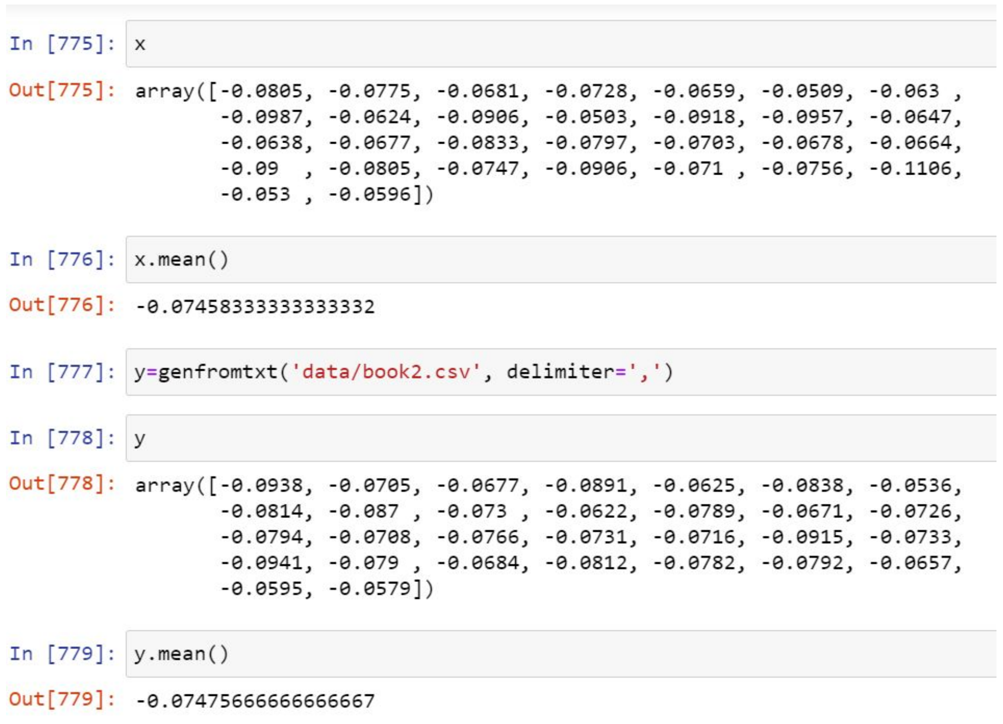
```
In [771]: original_SR=x
          new_model_SR=y

          sample_size=30
          alpha=0.05
          compare_2_groups(x,y, alpha, sample_size)

          statistics=-2.005,p=0.050
          different distribution (reject H0)
```

A.10 D’Agostino and Person test for normal distribution of max drawdowns of the new model

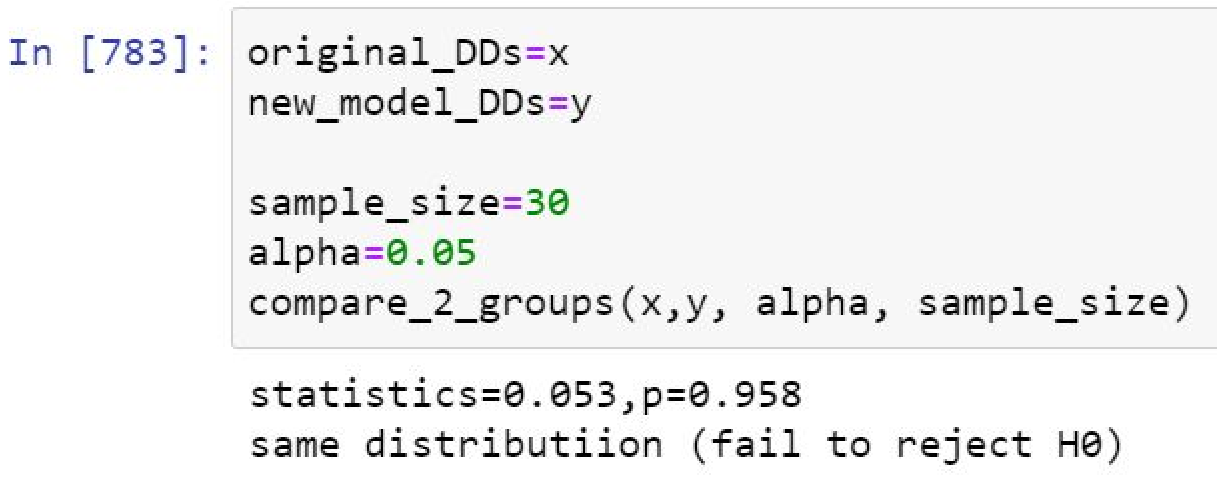


```
In [772]: #test for normality
          # import scipy.stats as stats

          x = genfromtxt('data/book3.csv', delimiter=',')

          DDs_of_newModel=x

          DDs_of_newModel

Out[772]: array([-0.0938, -0.0705, -0.0677, -0.0891, -0.0625, -0.0838, -0.0536,
                 -0.0814, -0.087 , -0.073 , -0.0622, -0.0789, -0.0671, -0.0726,
                 -0.0794, -0.0708, -0.0766, -0.0731, -0.0716, -0.0915, -0.0733,
                 -0.0941, -0.079 , -0.0684, -0.0812, -0.0782, -0.0792, -0.0657,
                 -0.0595, -0.0579])

In [773]: stats.normaltest(x)

Out[773]: NormaltestResult(statistic=0.21860976900535734, pvalue=0.896457059964215)
```

A.10.1 t-Test of max drawdowns of original ensemble versus the new model’s max drawdowns



```
In [775]: x

Out[775]: array([-0.0805, -0.0775, -0.0681, -0.0728, -0.0659, -0.0509, -0.063 ,
                 -0.0987, -0.0624, -0.0906, -0.0503, -0.0918, -0.0957, -0.0647,
                 -0.0638, -0.0677, -0.0833, -0.0797, -0.0703, -0.0678, -0.0664,
                 -0.09  , -0.0805, -0.0747, -0.0906, -0.071 , -0.0756, -0.1106,
                 -0.053 , -0.0596])

In [776]: x.mean()

Out[776]: -0.07458333333333332

In [777]: y=genfromtxt('data/book2.csv', delimiter=',')

In [778]: y

Out[778]: array([-0.0938, -0.0705, -0.0677, -0.0891, -0.0625, -0.0838, -0.0536,
                 -0.0814, -0.087 , -0.073 , -0.0622, -0.0789, -0.0671, -0.0726,
                 -0.0794, -0.0708, -0.0766, -0.0731, -0.0716, -0.0915, -0.0733,
                 -0.0941, -0.079 , -0.0684, -0.0812, -0.0782, -0.0792, -0.0657,
                 -0.0595, -0.0579])

In [779]: y.mean()

Out[779]: -0.07475666666666667
```

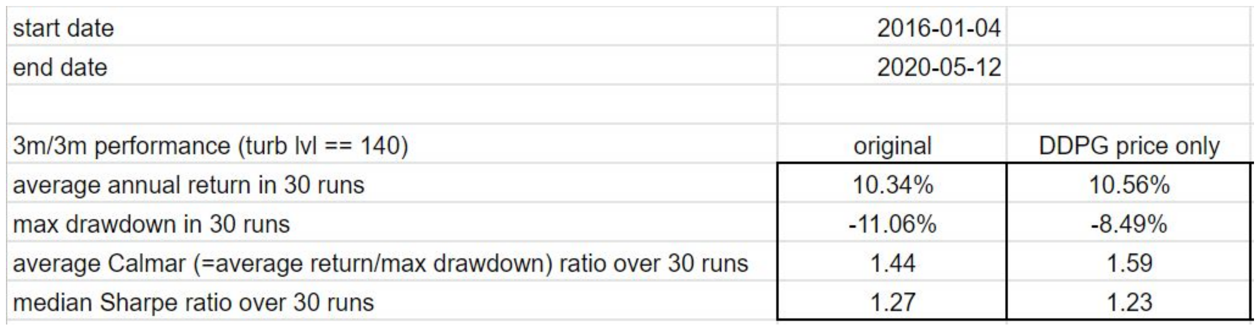


```
In [783]: original_DDs=x
          new_model_DDs=y

          sample_size=30
          alpha=0.05
          compare_2_groups(x,y, alpha, sample_size)

          statistics=0.053,p=0.958
          same distributiion (fail to reject H0)
```

Appendix B: Further experiments

B.1 DDPG agent without any technical indicators, nor VIX data, nor turbulence index, i.e. only price, current portfolio, and money balance in observation space

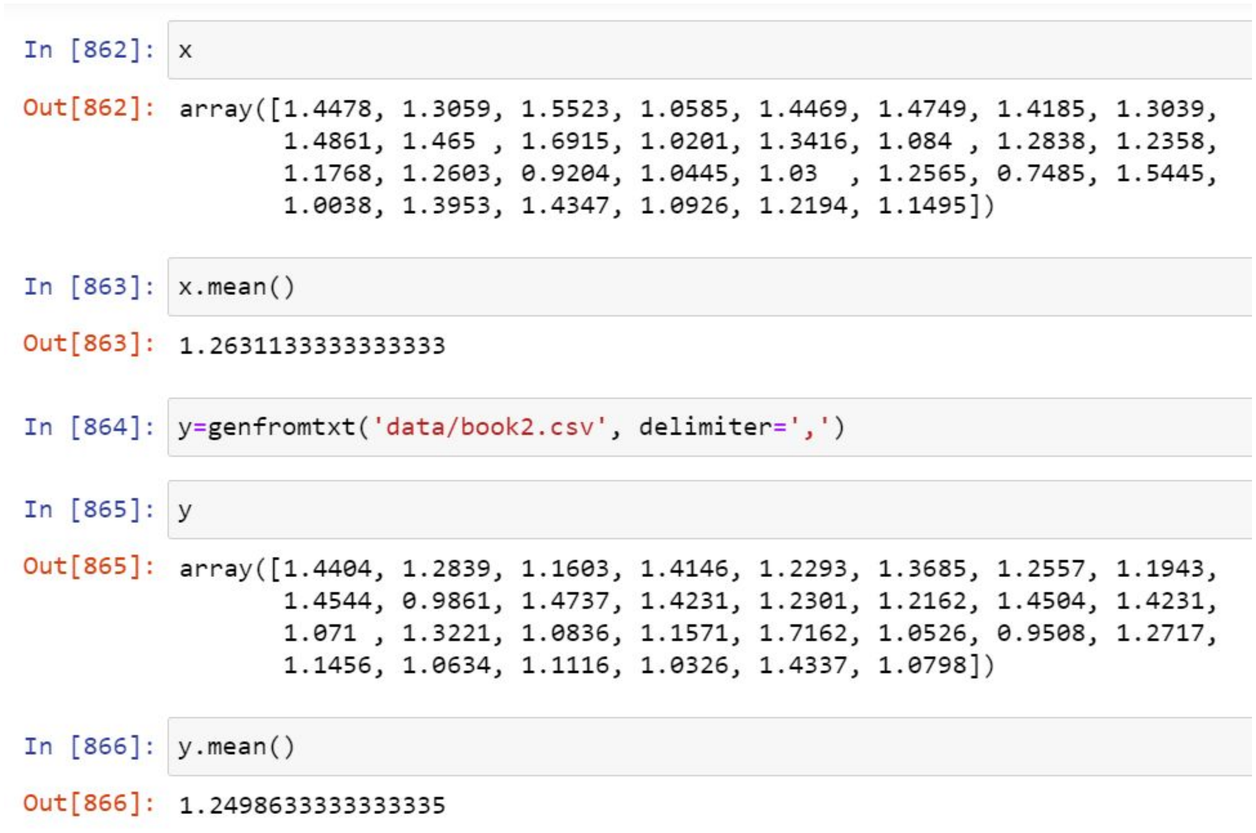
Given that all the technical indicators used are derivatives of the price, we found it worthwhile to test a slimmed down DDPG agent's performance. Here is the result of 30 runs compared to the original ensemble:



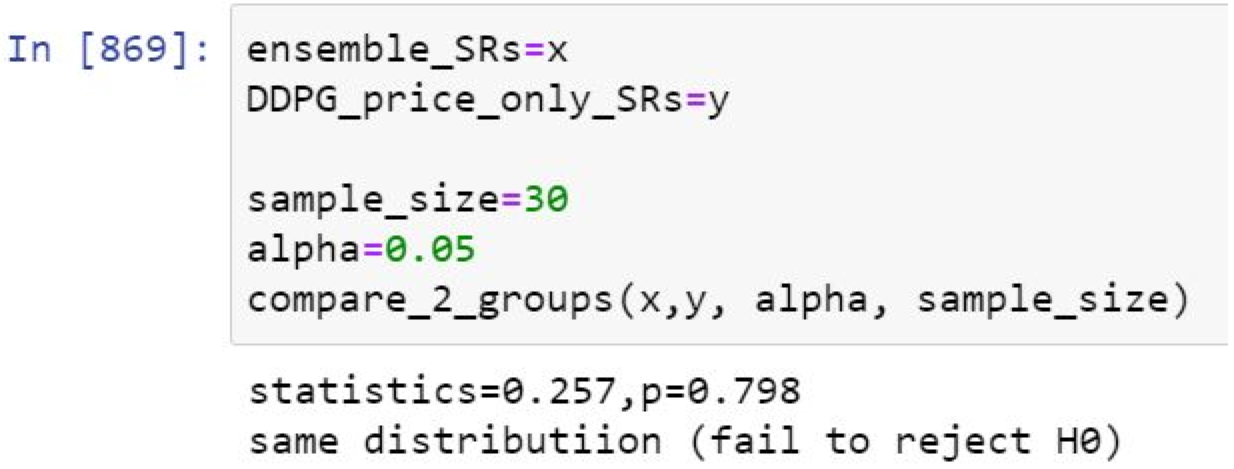
start date	2016-01-04	
end date	2020-05-12	
3m/3m performance (turb lvl == 140)	original	DDPG price only
average annual return in 30 runs	10.34%	10.56%
max drawdown in 30 runs	-11.06%	-8.49%
average Calmar (=average return/max drawdown) ratio over 30 runs	1.44	1.59
median Sharpe ratio over 30 runs	1.27	1.23

Table 7: Performance of the original ensemble versus a slimmed down DDPG agent

Its risk-adjusted performance is no worse than the original ensemble's:



```
In [862]: x

Out[862]: array([1.4478, 1.3059, 1.5523, 1.0585, 1.4469, 1.4749, 1.4185, 1.3039,
                 1.4861, 1.465 , 1.6915, 1.0201, 1.3416, 1.084 , 1.2838, 1.2358,
                 1.1768, 1.2603, 0.9204, 1.0445, 1.03  , 1.2565, 0.7485, 1.5445,
                 1.0038, 1.3953, 1.4347, 1.0926, 1.2194, 1.1495])

In [863]: x.mean()

Out[863]: 1.2631133333333333

In [864]: y=genfromtxt('data/book2.csv', delimiter=',')

In [865]: y

Out[865]: array([1.4404, 1.2839, 1.1603, 1.4146, 1.2293, 1.3685, 1.2557, 1.1943,
                 1.4544, 0.9861, 1.4737, 1.4231, 1.2301, 1.2162, 1.4504, 1.4231,
                 1.071 , 1.3221, 1.0836, 1.1571, 1.7162, 1.0526, 0.9508, 1.2717,
                 1.1456, 1.0634, 1.1116, 1.0326, 1.4337, 1.0798])

In [866]: y.mean()

Out[866]: 1.2498633333333335
```



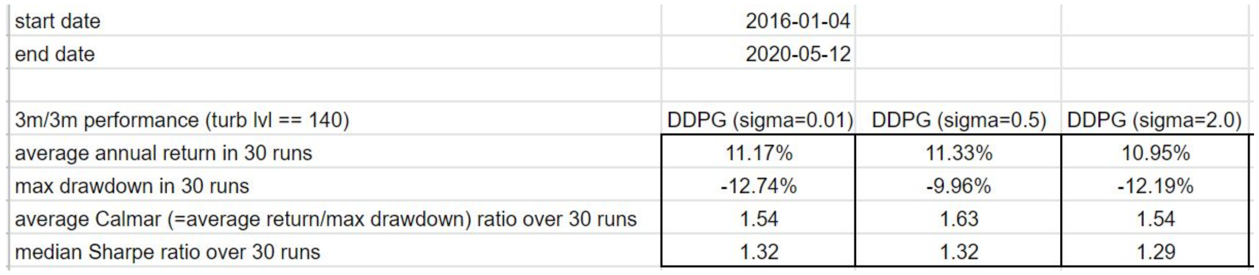
```
In [869]: ensemble_SRs=x
          DDPG_price_only_SRs=y

          sample_size=30
          alpha=0.05
          compare_2_groups(x,y, alpha, sample_size)

          statistics=0.257,p=0.798
          same distributiion (fail to reject H0)
```

B.2 DDPG agent with different noise levels

The original ensemble uses 0.5 for the variability of the DDPG agent's action noise [8]:

```
action_noise = OrnsteinUhlenbeckActionNoise(mean=np.zeros(n_actions), sigma=float(0.5) * np.ones(n_actions))
```

Running 30 simulations of the DDPG agent with different levels of this noise produced the following results:



start date	2016-01-04		
end date	2020-05-12		
3m/3m performance (turb lvl == 140)	DDPG (sigma=0.01)	DDPG (sigma=0.5)	DDPG (sigma=2.0)
average annual return in 30 runs	11.17%	11.33%	10.95%
max drawdown in 30 runs	-12.74%	-9.96%	-12.19%
average Calmar (=average return/max drawdown) ratio over 30 runs	1.54	1.63	1.54
median Sharpe ratio over 30 runs	1.32	1.32	1.29

Table 8: Performance of the original DDPG agent versus a DDPG with lower and higher action noise

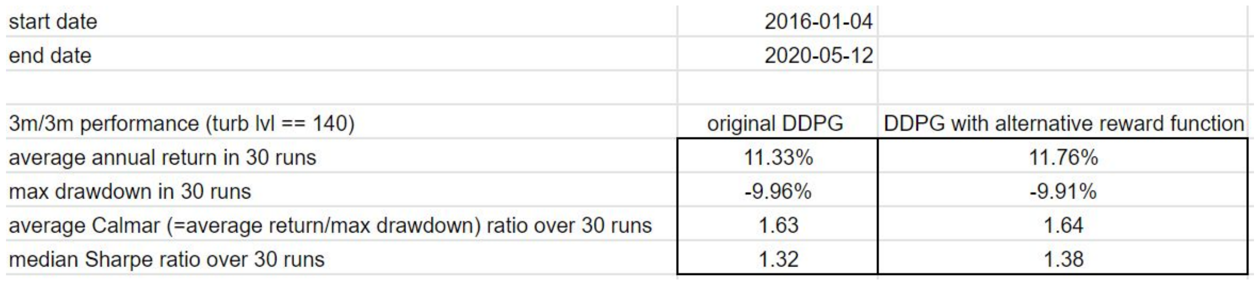
B.3 DDPG agent with alternative reward function

In order to minimize transaction costs further we tried to discourage the agent to trade excessively by amending the reward function.

Original reward function: self.reward = end_total_asset - begin_total_asset

Alternative reward function: self.reward = end_total_asset - begin_total_asset - (self.cost**1.4)

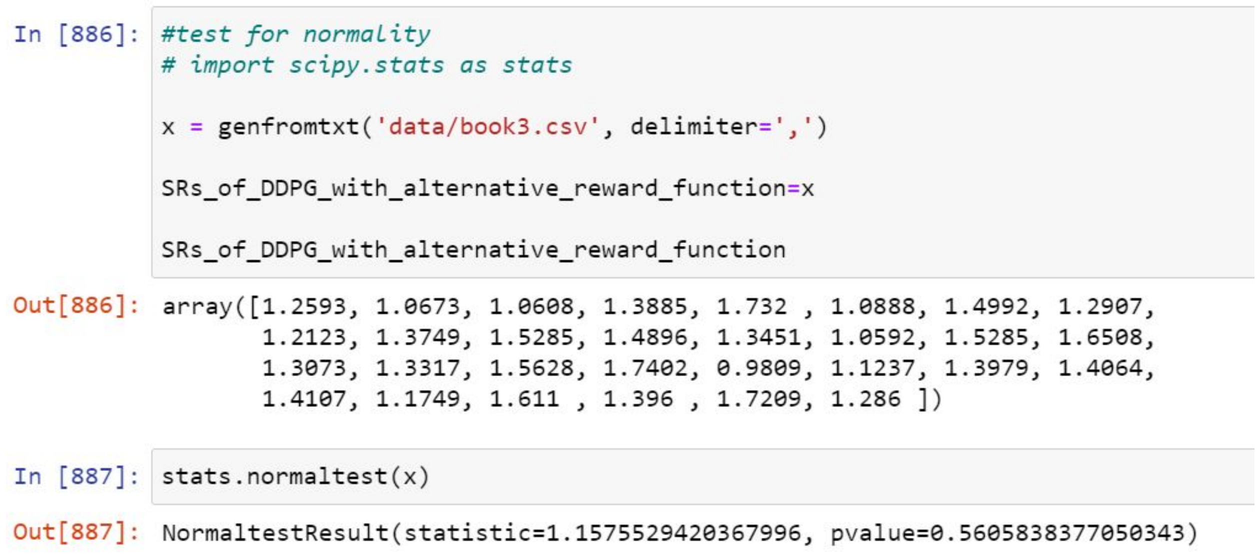
A simulation of 30 runs produced the following results:



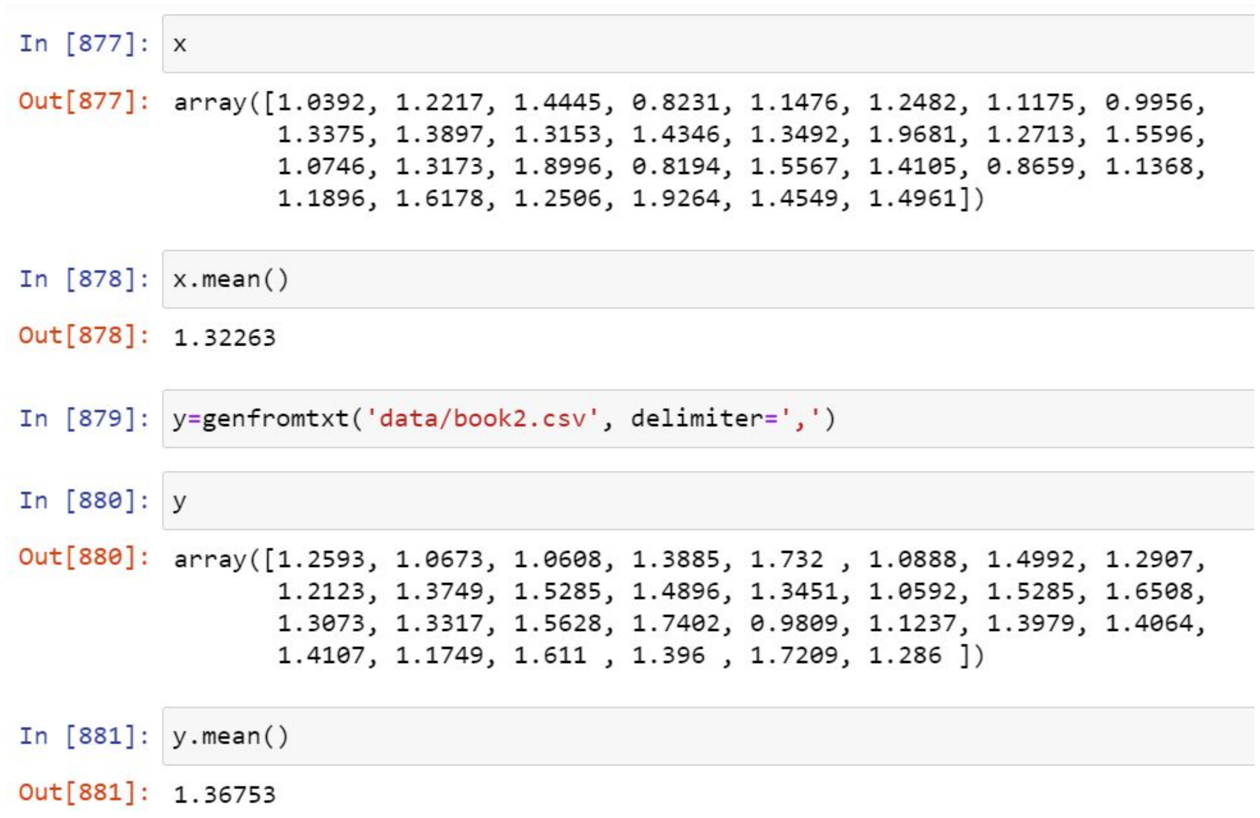
start date	2016-01-04	
end date	2020-05-12	
3m/3m performance (turb lvl == 140)	original DDPG	DDPG with alternative reward function
average annual return in 30 runs	11.33%	11.76%
max drawdown in 30 runs	-9.96%	-9.91%
average Calmar (=average return/max drawdown) ratio over 30 runs	1.63	1.64
median Sharpe ratio over 30 runs	1.32	1.38

Table 9: Performance of the original DDPG agent versus a DDPG agent with an alternative reward function

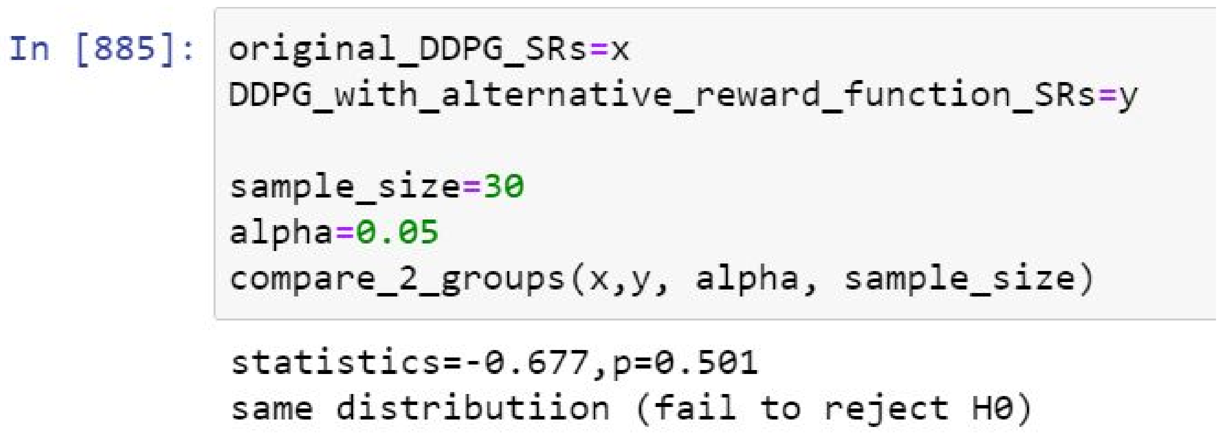
Even though the DDPG agent with the alternative reward function produced higher Sharpe ratios, the difference between the original DDPG risk-adjusted returns and this agent's Sharpe ratios are not significant:



```
In [886]: #test for normality
          # import scipy.stats as stats

          x = genfromtxt('data/book3.csv', delimiter=',')

          SRs_of_DDPG_with_alternative_reward_function=x

          SRs_of_DDPG_with_alternative_reward_function

Out[886]: array([1.2593, 1.0673, 1.0608, 1.3885, 1.732 , 1.0888, 1.4992, 1.2907,
                 1.2123, 1.3749, 1.5285, 1.4896, 1.3451, 1.0592, 1.5285, 1.6508,
                 1.3073, 1.3317, 1.5628, 1.7402, 0.9809, 1.1237, 1.3979, 1.4064,
                 1.4107, 1.1749, 1.611 , 1.396 , 1.7209, 1.286 ])

In [887]: stats.normaltest(x)

Out[887]: NormaltestResult(statistic=1.1575529420367996, pvalue=0.5605838377050343)
```



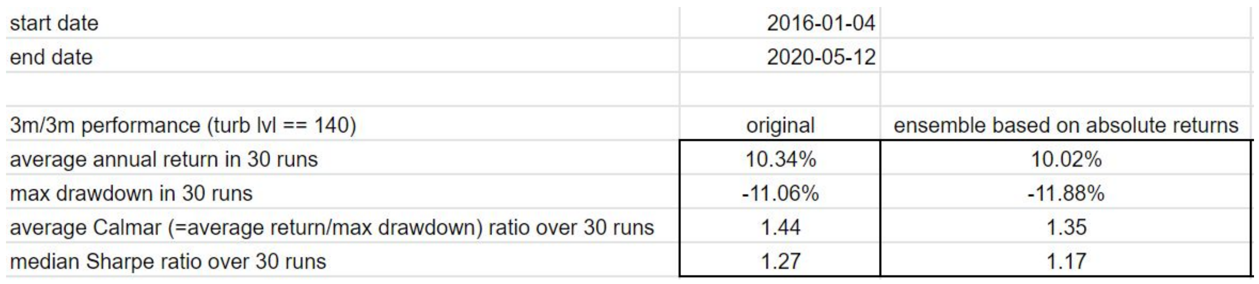
```
In [877]: x

Out[877]: array([1.0392, 1.2217, 1.4445, 0.8231, 1.1476, 1.2482, 1.1175, 0.9956,
                 1.3375, 1.3897, 1.3153, 1.4346, 1.3492, 1.9681, 1.2713, 1.5596,
                 1.0746, 1.3173, 1.8996, 0.8194, 1.5567, 1.4105, 0.8659, 1.1368,
                 1.1896, 1.6178, 1.2506, 1.9264, 1.4549, 1.4961])

In [878]: x.mean()

Out[878]: 1.32263

In [879]: y=genfromtxt('data/book2.csv', delimiter=',')

In [880]: y

Out[880]: array([1.2593, 1.0673, 1.0608, 1.3885, 1.732 , 1.0888, 1.4992, 1.2907,
                 1.2123, 1.3749, 1.5285, 1.4896, 1.3451, 1.0592, 1.5285, 1.6508,
                 1.3073, 1.3317, 1.5628, 1.7402, 0.9809, 1.1237, 1.3979, 1.4064,
                 1.4107, 1.1749, 1.611 , 1.396 , 1.7209, 1.286 ])

In [881]: y.mean()

Out[881]: 1.36753
```



```
In [885]: original_DDPG_SRs=x
          DDPG_with_alternative_reward_function_SRs=y

          sample_size=30
          alpha=0.05
          compare_2_groups(x,y, alpha, sample_size)

          statistics=-0.677,p=0.501
          same distributiion (fail to reject H0)
```

B.4 Ensemble with highest return instead of highest risk-adjusted return as selection criteria after validation period

As the random-choice ensemble of 4.1.4 did no worse than the original ensemble, we explored an alternative agent selection process after the validation period based on the highest absolute return produced during the validation period instead of the highest risk-adjusted return:



start date	2016-01-04	
end date	2020-05-12	
3m/3m performance (turb lvl == 140)	original	ensemble based on absolute returns
average annual return in 30 runs	10.34%	10.02%
max drawdown in 30 runs	-11.06%	-11.88%
average Calmar (=average return/max drawdown) ratio over 30 runs	1.44	1.35
median Sharpe ratio over 30 runs	1.27	1.17

Table 10: Performance of the original ensemble versus the ensemble based on absolute returns during the validation period

Bibliography

[1] Yang, Hongyang and Liu, Xiao-Yang and Zhong, Shan and Walid, Anwar, “Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy”, September 11, 2020. Available at SSRN: https://ssrn.com/abstract=3690996 or http://dx.doi.org/10.2139/ssrn.3690996

[2] Kenechukwu Anadu, Mathias Kruttli, Patrick McCabe, and Emilio Osambela, “The Shift from Active to Passive Investing: Risks to Financial Stability?”, Working Paper | SRA 18-04 | August 27, 2018. Last Revised: May 15, 2020, pp. 2-3

[3] Silver, David, 22nd January 2021, webpage: https://www.youtube.com/watch?v=x5Q79XCxMVc

[4] Zuckerman, Gregory, “The Man Who Solved the Market: How Jim Simons Launched the Quant Revolution”, 2019, pp. 331-332

[5] 22nd January 2021, webpage: https://therobusttrader.com/what-percentage-of-trading-is-algorithmic/#:~:text=In%20the%20U.S.%20stock%20market,is%20generated%20through%20algorithmic%20trading.

[6] Fischer, Thomas G., 2018. "Reinforcement learning in financial markets - a survey," FAU Discussion Papers in Economics 12/2018, Friedrich-Alexander University Erlangen-Nuremberg, Institute for Economics, pp. 2-4, and pp. 38-39

[7] 23rd January 2021, webpage: https://towardsdatascience.com/creating-a-custom-openai-gym-environment-for-stock-trading-be532be3910e

[8] 23rd January 2021, webpage: https://github.com/AI4Finance-LLC/Deep-Reinforcement-Learning-for-Automated-Stock-Trading-Ensemble-Strategy-ICAIF-2020, model folder, models.py, line 57

[9] Bortz, Jürgen “Statistik für Sozialwissenschaftler”, 4th edition, page 91

[10] 24th January 2021, webpage: https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.normaltest.html

[11] 25th January 2021, webpage: https://www.cboe.com/tradable_products/vix/faqs/

[12] Kestner, Lars “Quantitative Trading Strategies. Harnessing the power of quantitative techniques to create a winning trading program”, Chapter 11.

[13] 4th February 2021, webpage:
https://openai.com/blog/better-exploration-with-parameter-noise/