Stock Market Prediction Using Deep Reinforcement learning: Ensemble strategy

2.1. Install all the packages through FinRL library

```
## install finrl library
!pip install FinRL
```

```
from finrl import config
from finrl import config_tickers
import os
if not os.path.exists("./" + config.DATA_SAVE_DIR):
    os.makedirs("./" + config.DATA_SAVE_DIR)
if not os.path.exists("./" + config.TRAINED_MODEL_DIR):
    os.makedirs("./" + config.TRAINED_MODEL_DIR)
if not os.path.exists("./" + config.TENSORBOARD_LOG_DIR):
    os.makedirs("./" + config.TENSORBOARD_LOG_DIR)
if not os.path.exists("./" + config.RESULTS_DIR):
    os.makedirs("./" + config.RESULTS_DIR)
```

2.2. Import Packages

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
# matplotlib.use('Agg')
import datetime

%matplotlib inline
from finrl.finrl_meta.preprocessor.yahoodownloader import YahooDownloader
from finrl.finrl_meta.preprocessor.preprocessors import FeatureEngineer, data_split
from finrl.finrl_meta.env_stock_trading.env_stocktrading import StockTradingEnv
from finrl.agents.stablebaselines3.models import DRLAgent
from finrl.finrl_meta.data_processor import DataProcessor

from finrl.plot import backtest_stats, backtest_plot, get_daily_return, get_baseline
from pprint import pprint

import sys
sys.path.append("../FinRL-Library")
```

```
import itertools
```

```
/usr/local/lib/python3.7/dist-packages/pyfolio/pos.py:27: UserWarning: Module "zipline.a
  'Module "zipline.assets" not found; multipliers will not be applied'
```

Part 3. Download Data

```
# from config.py TRAIN_START_DATE is a string
config.TRAIN_START_DATE
```

```
# from config.py TRAIN_END_DATE is a string
config.TRAIN_END_DATE
```

```
df = YahooDownloader(start_date = '2009-01-01',
                     end_date = '2021-10-31',
                     ticker_list = config_tickers.DOW_30_TICKER).fetch_data()
```

```
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
[*********************100%***********************]  1 of 1 completed
Shape of DataFrame:  (94331, 8)
```

```
print(config_tickers.DOW_30_TICKER)
```

```
['AXP', 'AMGN', 'AAPL', 'BA', 'CAT', 'CSCO', 'CVX', 'GS', 'HD', 'HON', 'IBM', 'INTC', ':
```

```
df.shape
```

```
(94331, 8)
```

```
df.sort_values(['date','tic'],ignore_index=True).head()
```

	date	open	high	low	close	volume	tic	day
0	2009-01-02	3.067143	3.251429	3.041429	2.771174	746015200	AAPL	4
1	2009-01-02	58.590000	59.080002	57.750000	44.867592	6547900	AMGN	4
2	2009-01-02	18.570000	19.520000	18.400000	15.535348	10955700	AXP	4
3	2009-01-02	42.799999	45.560001	42.779999	33.941101	7010200	BA	4
4	2009-01-02	44.910000	46.980000	44.709999	32.164726	7117200	CAT	4

Part 4: Preprocess Data

```
fe = FeatureEngineer(
                    use_technical_indicator=True,
                    tech_indicator_list = config.INDICATORS,
                    use_vix=True,
                    use_turbulence=True,
                    user_defined_feature = False)

processed = fe.preprocess_data(df)
```

```
Successfully added technical indicators
[*********************100%***********************]  1 of 1 completed
Shape of DataFrame:  (3229, 8)
Successfully added vix
Successfully added turbulence index
Successfully added technical indicators
[*********************100%***********************]  1 of 1 completed
Shape of DataFrame:  (3229, 8)
Successfully added vix
Successfully added turbulence index
```

```
list_ticker = processed["tic"].unique().tolist()
list_date = list(pd.date_range(processed['date'].min(),processed['date'].max()).astype(str))
```

```
combination = list(itertools.product(list_date,list_ticker))

processed_full = pd.DataFrame(combination,columns=["date","tic"]).merge(processed,on=["date",
processed_full = processed_full[processed_full['date'].isin(processed['date'])]
processed_full = processed_full.sort_values(['date','tic'])

processed_full = processed_full.fillna(0)
```

```
processed_full.sort_values(['date','tic'],ignore_index=True).head(10)
```

	date	tic	open	high	low	close	volume	day	macd	boll_
0	2009-01-02	AAPL	3.067143	3.251429	3.041429	2.771174	746015200.0	4.0	0.0	2.99
1	2009-01-02	AMGN	58.590000	59.080002	57.750000	44.867592	6547900.0	4.0	0.0	2.99
2	2009-01-02	AXP	18.570000	19.520000	18.400000	15.535348	10955700.0	4.0	0.0	2.99
3	2009-01-02	BA	42.799999	45.560001	42.779999	33.941101	7010200.0	4.0	0.0	2.99
4	2009-01-02	CAT	44.910000	46.980000	44.709999	32.164726	7117200.0	4.0	0.0	2.99
5	2009-01-02	CRM	8.025000	8.550000	7.912500	8.505000	4069200.0	4.0	0.0	2.99

Part 5. Design Environment

```
train = data_split(processed_full, '2009-01-01','2020-07-01')
trade = data_split(processed_full, '2020-07-01','2021-10-31')
print(len(train))
print(len(trade))
```

```
83897
9744
```

```
train.tail()
```

	date	tic	open	high	low	close	volume	day	ma
2892	2020-06-30	UNH	288.570007	296.450012	287.660004	287.776794	2932900.0	1.0	-0.0194
2892	2020-06-30	V	191.490005	193.750000	190.160004	190.737213	9040100.0	1.0	1.0487
2892	2020-06-30	VZ	54.919998	55.290001	54.360001	50.376743	17414800.0	1.0	-0.437
2892	2020-06-30	WBA	42.119999	42.580002	41.759998	39.035736	4782100.0	1.0	-0.0839
2892	2020-06-30	WMT	119.220001	120.129997	118.540001	116.121773	6836400.0	1.0	-0.8865

```
trade.head()
```

	date	tic	open	high	low	close	volume	day	mac
0	2020-07-01	AAPL	91.279999	91.839996	90.977501	89.904602	110737200.0	2.0	3.01460
0	2020-07-01	AMGN	235.520004	256.230011	232.580002	240.153961	6575800.0	2.0	3.63639
0	2020-07-01	AXP	95.250000	96.959999	93.639999	92.086372	3301000.0	2.0	-0.38916
0	2020-07-01	BA	185.880005	190.610001	180.039993	180.320007	49036700.0	2.0	5.44319
0	2020-07-01	CAT	129.380005	129.399994	125.879997	120.651642	2807800.0	2.0	1.27262

```
config.INDICATORS
```

```
['macd',
 'boll_ub',
 'boll_lb',
 'rsi_30',
 'cci_30',
 'dx_30',
 'close_30_sma',
 'close_60_sma']
```

```
stock_dimension = len(train.tic.unique())
state_space = 1 + 2*stock_dimension + len(config.INDICATORS)*stock_dimension
```

```
print(f"Stock Dimension: {stock_dimension}, State Space: {state_space}")
```

```
Stock Dimension: 29, State Space: 291
```

```
buy_cost_list = sell_cost_list = [0.001] * stock_dimension
num_stock_shares = [0] * stock_dimension

env_kwargs = {
    "hmax": 100,
    "initial_amount": 1000000,
    "num_stock_shares": num_stock_shares,
    "buy_cost_pct": buy_cost_list,
    "sell_cost_pct": sell_cost_list,
    "state_space": state_space,
    "stock_dim": stock_dimension,
    "tech_indicator_list": config.INDICATORS,
    "action_space": stock_dimension,
    "reward_scaling": 1e-4
}


e_train_gym = StockTradingEnv(df = train, **env_kwargs)
```

Environment for Training

```
env_train, _ = e_train_gym.get_sb_env()
print(type(env_train))
```

```
<class 'stable_baselines3.common.vec_env.dummy_vec_env.DummyVecEnv'>
```

Part 6: Implement DRL Algorithms

```
agent = DRLAgent(env = env_train)
```

Model Training: 3 models, A2C DDPG, PPO

Model 1: A2C

```
agent = DRLAgent(env = env_train)
model_a2c = agent.get_model("a2c")
```

```
{'n_steps': 5, 'ent_coef': 0.01, 'learning_rate': 0.0007}
Using cuda device
```

```
trained_a2c = agent.train_model(model=model_a2c,
                                tb_log_name='a2c',
                                total_timesteps=50000)
```

```
--------------------------------------
| time/                 |            |
|    fps                | 97         |
|    iterations         | 100        |
|    time_elapsed       | 5          |
|    total_timesteps    | 500        |
| train/                |            |
|    entropy_loss       | -41.3      |
|    explained_variance | 0.461      |
|    learning_rate      | 0.0007     |
|    n_updates          | 99         |
|    policy_loss        | -19.4      |
|    reward             | 0.18487218 |
|    std                | 1          |
|    value_loss         | 0.456      |
--------------------------------------
---------------------------------------
| time/                 |             |
|    fps                | 100         |
|    iterations         | 200         |
|    time_elapsed       | 9           |
|    total_timesteps    | 1000        |
| train/                |             |
|    entropy_loss       | -41.3       |
|    explained_variance | -0.0554     |
|    learning_rate      | 0.0007      |
|    n_updates          | 199         |
|    policy_loss        | -61.1       |
|    reward             | -0.18095054 |
|    std                | 1.01        |
|    value_loss         | 2.58        |
---------------------------------------
------------------------------------
| time/                 |          |
|    fps                | 100      |
|    iterations         | 300      |
|    time_elapsed       | 14       |
|    total_timesteps    | 1500     |
| train/                |          |
|    entropy_loss       | -41.3    |
|    explained_variance | 0        |
|    learning_rate      | 0.0007   |
|    n_updates          | 299      |
|    policy_loss        | -238     |
|    reward             | 4.376703 |
|    std                | 1.01     |
|    value_loss         | 31.3     |
------------------------------------
```

```
------------------------------------
| time/                 |          |
|    fps                | 100      |
|    iterations         | 400      |
|    time_elapsed       | 19       |
|    total_timesteps    | 2000     |
| train/                |          |
|    entropy_loss       | -41.3    |
|    explained_variance | -1.19e-07 |
|    learning_rate      | 0.0007   |
```

Model 2: DDPG

```
agent = DRLAgent(env = env_train)
model_ddpg = agent.get_model("ddpg")
```

```
{'batch_size': 128, 'buffer_size': 50000, 'learning_rate': 0.001}
Using cuda device
```

```
trained_ddpg = agent.train_model(model=model_ddpg,
                             tb_log_name='ddpg',
                             total_timesteps=50000)
```

```
day: 2892, episode: 20
begin_total_asset: 1000000.00
end_total_asset: 4774133.37
total_reward: 3774133.37
total_cost: 5470.92
total_trades: 47732
Sharpe: 0.868
=================================
----------------------------------
| time/              |           |
|    episodes        | 4         |
|    fps             | 71        |
|    time_elapsed    | 162       |
|    total_timesteps | 11572     |
| train/             |           |
|    actor_loss      | 96.6      |
|    critic_loss     | 1.47e+03  |
|    learning_rate   | 0.001     |
|    n_updates       | 8679      |
|    reward          | 2.8689466 |
----------------------------------
----------------------------------
| time/              |           |
|    episodes        | 8         |
|    fps             | 66        |
|    time_elapsed    | 347       |
|    total_timesteps | 23144     |
| train/             |           |
|    actor_loss      | 24.9      |
|    critic_loss     | 14.5      |
```

```
|    learning_rate   | 0.001     |
|    n_updates       | 20251     |
|    reward          | 2.8689466 |
----------------------------------
day: 2892, episode: 30
begin_total_asset: 1000000.00
end_total_asset: 4390924.34
total_reward: 3390924.34
total_cost: 999.00
total_trades: 40488
Sharpe: 0.730
=================================
----------------------------------
| time/              |           |
|    episodes        | 12        |
|    fps             | 65        |
|    time_elapsed    | 532       |
|    total_timesteps | 34716     |
| train/             |           |
|    actor_loss      | 6.26      |
|    critic_loss     | 3.49      |
|    learning_rate   | 0.001     |
|    n_updates       | 31823     |
|    reward          | 2.8689466 |
----------------------------------
----------------------------------
| time/              |           |
|    episodes        | 16        |
```

Model 3: PPO

```
agent = DRLAgent(env = env_train)
PPO_PARAMS = {
    "n_steps": 2048,
    "ent_coef": 0.01,
    "learning_rate": 0.00025,
    "batch_size": 128,
}
model_ppo = agent.get_model("ppo",model_kwargs = PPO_PARAMS)
```

```
{'n_steps': 2048, 'ent_coef': 0.01, 'learning_rate': 0.00025, 'batch_size': 128}
Using cuda device
```

```
trained_ppo = agent.train_model(model=model_ppo,
                             tb_log_name='ppo',
                             total_timesteps=50000)
```

```
-----------------------------------
| time/              |            |
|    fps             | 115        |
|    iterations      | 1          |
|    time_elapsed    | 17         |
```

```
|    total_timesteps | 2048       |
| train/             |            |
|    reward          | 0.10877045 |
----------------------------------
-----------------------------------------
| time/                   |             |
|    fps                  | 112         |
|    iterations           | 2           |
|    time_elapsed         | 36          |
|    total_timesteps      | 4096        |
| train/                  |             |
|    approx_kl            | 0.015766323 |
|    clip_fraction        | 0.226       |
|    clip_range           | 0.2         |
|    entropy_loss         | -41.2       |
|    explained_variance   | -0.0621     |
|    learning_rate        | 0.00025     |
|    loss                 | 2.75        |
|    n_updates            | 10          |
|    policy_gradient_loss | -0.0268     |
|    reward               | 0.44184494  |
|    std                  | 1           |
|    value_loss           | 9.5         |
-----------------------------------------
day: 2892, episode: 40
begin_total_asset: 1000000.00
end_total_asset: 2858099.69
total_reward: 1858099.69
total_cost: 335985.43
total_trades: 80095
Sharpe: 0.673
=================================
-----------------------------------------
| time/                   |             |
|    fps                  | 111         |
|    iterations           | 3           |
|    time_elapsed         | 55          |
|    total_timesteps      | 6144        |
| train/                  |             |
|    approx_kl            | 0.014054896 |
|    clip_fraction        | 0.127       |
|    clip_range           | 0.2         |
|    entropy_loss         | -41.2       |
|    explained_variance   | -0.000786   |
|    learning_rate        | 0.00025     |
|    loss                 | 30.3        |
|    n_updates            | 20          |
|    policy_gradient_loss | -0.0177     |
|    reward               | -1.5774492  |
|    std                  | 1           |
|    value_loss           | 54.6        |
-----------------------------------------
------------------------------------------
```

Trading

Assume that we have $1,000,000 initial capital at 2020-07-01. We use the DDPG model to trade.

```
data_risk_indicator = processed_full[(processed_full.date<'2020-07-01') & (processed_full.dat
insample_risk_indicator = data_risk_indicator.drop_duplicates(subset=['date'])
```

```
insample_risk_indicator.vix.describe()
```

```
count    2893.000000
mean       18.824245
std         8.489311
min         9.140000
25%        13.330000
50%        16.139999
75%        21.309999
max        82.690002
Name: vix, dtype: float64
```

```
insample_risk_indicator.vix.quantile(0.996)
```

```
57.40400183105453
```

```
insample_risk_indicator.turbulence.describe()
```

```
count    2893.000000
mean       34.567961
std        43.790810
min         0.000000
25%        14.962718
50%        24.124747
75%        39.162552
max       652.506566
Name: turbulence, dtype: float64
```

```
insample_risk_indicator.turbulence.quantile(0.996)
```

```
276.45175727554096
```

Trade

```
#trade = data_split(processed_full, '2020-07-01','2021-10-31')
e_trade_gym = StockTradingEnv(df = trade, turbulence_threshold = 70,risk_indicator_col='vix',
```

```
trade.head()
```

	date	tic	open	high	low	close	volume	day	mac
0	2020-07-01	AAPL	91.279999	91.839996	90.977501	89.904602	110737200.0	2.0	3.0146(
0	2020-07-01	AMGN	235.520004	256.230011	232.580002	240.153961	6575800.0	2.0	3.6363(
0	2020-07-01	AXP	95.250000	96.959999	93.639999	92.086372	3301000.0	2.0	-0.3891(
0	2020-07-01	BA	185.880005	190.610001	180.039993	180.320007	49036700.0	2.0	5.4431(
0	2020-07-01	CAT	129.380005	129.399994	125.879997	120.651642	2807800.0	2.0	1.2726(

```
df_account_value, df_actions = DRLAgent.DRL_prediction(
    model=trained_ddpg,
    environment = e_trade_gym)
```

```
hit end!
```

```
df_account_value.shape
```

```
(336, 2)
```

```
df_account_value.tail()
```

	date	account_value
331	2021-10-22	1.424764e+06
332	2021-10-25	1.433647e+06
333	2021-10-26	1.427543e+06
334	2021-10-27	1.426441e+06
335	2021-10-28	1.440410e+06

```
df_actions.head()
```

	AAPL	AMGN	AXP	BA	CAT	CRM	CSCO	CVX	DIS	GS	...	MRK	MSFT	NKE	PG	TRV
date																
2020-07-01	0	0	0	98	99	0	0	0	0	77	...	0	95	0	0	0

Part 7: Backtest Our Strategy

7.1 BackTestStats

```
print("==============Get Backtest Results===========")
now = datetime.datetime.now().strftime('%Y%m%d-%Hh%M')

perf_stats_all = backtest_stats(account_value=df_account_value)
perf_stats_all = pd.DataFrame(perf_stats_all)
perf_stats_all.to_csv("./"+config.RESULTS_DIR+"/perf_stats_all_"+now+'.csv')
```

```
==============Get Backtest Results===========
Annual return          0.314815
Cumulative returns     0.440410
Annual volatility      0.135542
Sharpe ratio           2.094102
Calmar ratio           4.162814
Stability              0.952226
Max drawdown          -0.075625
Omega ratio            1.407491
Sortino ratio          3.063127
Skew                        NaN
Kurtosis                    NaN
Tail ratio             1.091429
Daily value at risk   -0.015950
dtype: float64
```

```
#baseline stats
print("==============Get Baseline Stats===========")
baseline_df = get_baseline(
        ticker="^DJI",
        start = df_account_value.loc[0,'date'],
        end = df_account_value.loc[len(df_account_value)-1,'date'])

stats = backtest_stats(baseline_df, value_col_name = 'close')
```

```
==============Get Baseline Stats===========
[*********************100%***********************]  1 of 1 completed
Shape of DataFrame:  (335, 8)
```

```
Annual return          0.273520
Cumulative returns     0.379084
Annual volatility      0.139248
Sharpe ratio           1.811893
Calmar ratio           3.062662
Stability              0.918651
Max drawdown          -0.089308
Omega ratio            1.351851
Sortino ratio          2.684720
Skew                        NaN
Kurtosis                    NaN
Tail ratio             1.051856
Daily value at risk   -0.016542
dtype: float64
```

```
df_account_value.loc[0,'date']
```

```
'2020-07-01'
```

```
df_account_value.loc[len(df_account_value)-1,'date']
```

```
'2021-10-28'
```

7.2 BackTestPlot

```
print("==============Compare to DJIA===========")
%matplotlib inline
# Dow Jones Index: ^DJI
backtest_plot(df_account_value,
             baseline_ticker = '^DJI',
             baseline_start = df_account_value.loc[0,'date'],
             baseline_end = df_account_value.loc[len(df_account_value)-1,'date'])
```

```
==============Compare to DJIA===========
[*********************100%***********************]  1 of 1 completed
Shape of DataFrame:  (335, 8)
```

Start date	2020-07-01
End date	2021-10-28
Total months	16

	Backtest
Annual return	31.481%
Cumulative returns	44.041%
Annual volatility	13.554%
Sharpe ratio	2.09
Calmar ratio	4.16
Stability	0.95
Max drawdown	-7.563%
Omega ratio	1.41
Sortino ratio	3.06
Skew	NaN
Kurtosis	NaN
Tail ratio	1.09
Daily value at risk	-1.595%
Alpha	0.07
Beta	0.86

Worst drawdown periods	Net drawdown in %	Peak date	Valley date	Recovery date	Durat
0	7.56	2020-09-02	2020-09-23	2020-10-13	

Part 6: Implement DRL Algorithms

```
rebalance_window = 63
validation_window = 63
train_start = '2000-01-01'
train_end = '2019-01-01'
val_test_start = '2019-01-01'
val_test_end = '2021-01-18'

ensemble_agent = DRLEnsembleAgent(df=processed_full,
                 train_period=(train_start,train_end),
```

```
                 val_test_period=(val_test_start,val_test_end),
                 rebalance_window=rebalance_window,
                 validation_window=validation_window,
                 **env_kwargs)

A2C_model_kwargs = {
                    'n_steps': 5,
                    'ent_coef': 0.01,
                    'learning_rate': 0.0005
                    }

PPO_model_kwargs = {
                    "ent_coef":0.01,
                    "n_steps": 2048,
                    "learning_rate": 0.00025,
                    "batch_size": 128
                    }

DDPG_model_kwargs = {
                      "action_noise":"ornstein_uhlenbeck",
                      "buffer_size": 50_000,
                      "learning_rate": 0.000005,
                      "batch_size": 128
                    }

timesteps_dict = {'a2c' : 100_000,
                 'ppo' : 100_000,
                 'ddpg' : 50_000
                 }


df_summary = ensemble_agent.run_ensemble_strategy(A2C_model_kwargs,
                                                 PPO_model_kwargs,
                                                 DDPG_model_kwargs,
                                                 timesteps_dict)
```

```
============Start Ensemble Strategy============
============================================
nan
turbulence_threshold:  5625265.115256409
======Model training from:  2000-01-01 to  2019-01-02
======A2C Training========
{'n_steps': 5, 'ent_coef': 0.01, 'learning_rate': 0.0005}
Using cpu device
Logging to tensorboard_log/a2c/a2c_126_1
------------------------------------
| time/                 |          |
|    fps                | 99       |
|    iterations         | 100      |
|    time_elapsed       | 5        |
|    total_timesteps    | 500      |
| train/                |          |
|    entropy_loss       | -35.6    |
```

```
|    explained_variance | 0.229    |
|    learning_rate      | 0.0005   |
|    n_updates          | 99       |
|    policy_loss        | -77      |
|    std                | 1        |
|    value_loss         | 4.79     |
------------------------------------
-------------------------------------
| time/                 |           |
|    fps                | 99        |
|    iterations         | 200       |
|    time_elapsed       | 10        |
|    total_timesteps    | 1000      |
| train/                |           |
|    entropy_loss       | -35.6     |
|    explained_variance | -1.19e-07 |
|    learning_rate      | 0.0005    |
|    n_updates          | 199       |
|    policy_loss        | -40.5     |
|    std                | 1.01      |
|    value_loss         | 2.02      |
-------------------------------------
------------------------------------
| time/                 |          |
|    fps                | 99       |
|    iterations         | 300      |
|    time_elapsed       | 15       |
|    total_timesteps    | 1500     |
| train/                |          |
|    entropy_loss       | -35.6    |
|    explained_variance | 0.188    |
|    learning_rate      | 0.0005   |
|    n_updates          | 299      |
|    policy_loss        | 250      |
|    std                | 1.01     |
|    value_loss         | 59.4     |
------------------------------------
-------------------------------------
| time/                 |           |
|    fps                | 99        |
```

```
df_summary
```

	Iter	Val Start	Val End	Model Used	A2C Sharpe	PPO Sharpe	DDPG Sharpe
0	126	2019-01-02	2019-04-01	DDPG	-0.230842	-0.0926179	0.320516

Part 7: Backtest Our Strategy

```
unique_trade_date = processed_full[(processed_full.date > val_test_start)&(processed_full.dat


df_trade_date = pd.DataFrame({'datadate':unique_trade_date})

df_account_value=pd.DataFrame()
for i in range(rebalance_window+validation_window, len(unique_trade_date)+1,rebalance_window)
    temp = pd.read_csv('results/account_value_trade_{}_{}.csv'.format('ensemble',i))
    df_account_value = df_account_value.append(temp,ignore_index=True)
sharpe=(252**0.5)*df_account_value.account_value.pct_change(1).mean()/df_account_value.accoun
print('Sharpe Ratio: ',sharpe)
df_account_value=df_account_value.join(df_trade_date[validation_window:].reset_index(drop=Tru
```

```
Sharpe Ratio:  0.17377457928215234
```

```
df_account_value.head()
```

	account_value	date	daily_return	datadate
0	500000.000000	2019-04-01	NaN	2019-04-01
1	498557.292308	2019-04-02	-0.002885	2019-04-02
2	497280.859251	2019-04-03	-0.002560	2019-04-03
3	515844.876585	2019-04-04	0.037331	2019-04-04
4	496792.628660	2019-04-05	-0.036934	2019-04-05

7.1 BackTestPlot

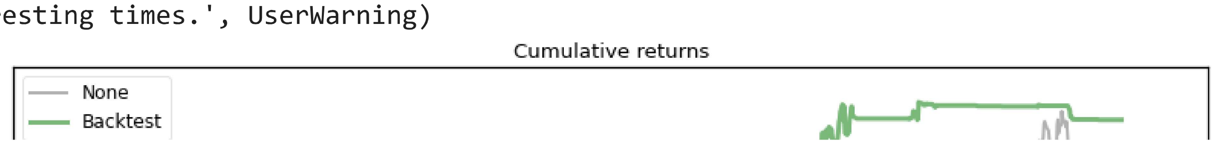
```
%matplotlib inline
BackTestPlot(df_account_value,
             baseline_ticker = '^DJI',
             baseline_start = df_account_value.loc[0,'date'],
             baseline_end = df_account_value.loc[len(df_account_value)-1,'date'])
```

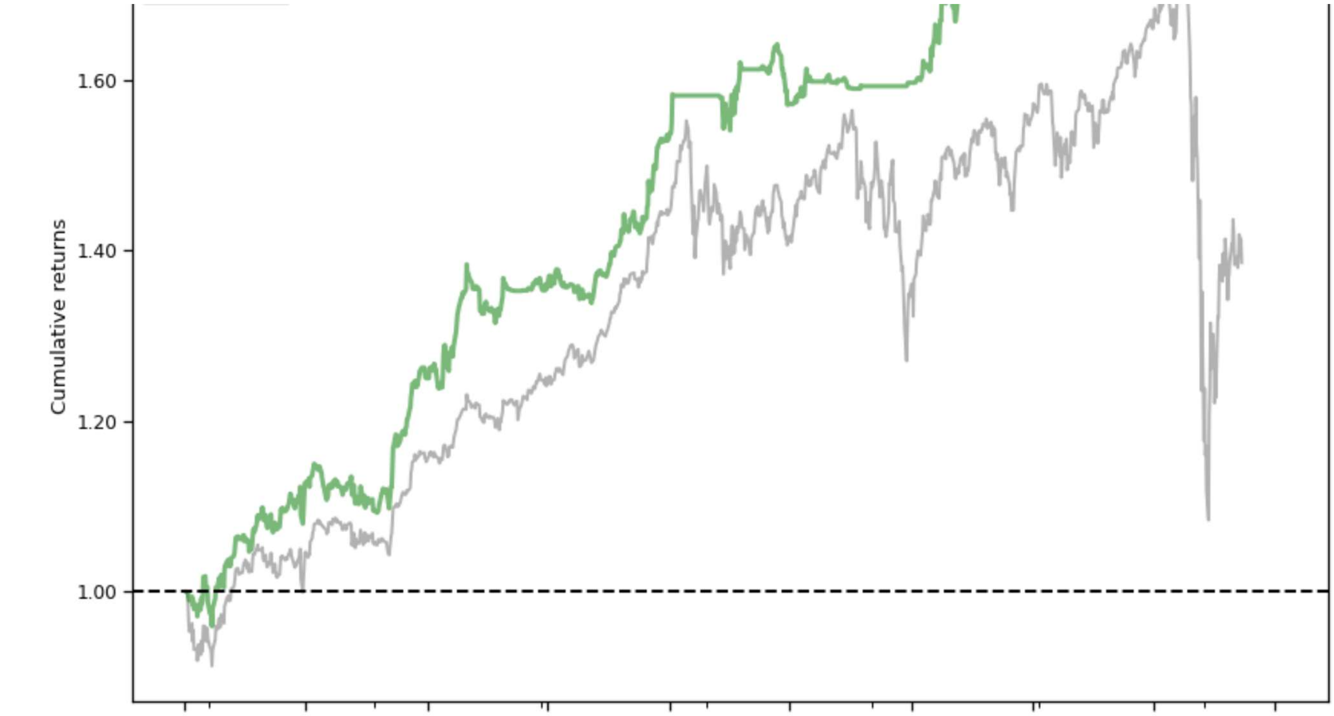
Start date	2016-01-04
End date	2020-05-12
Total months	52

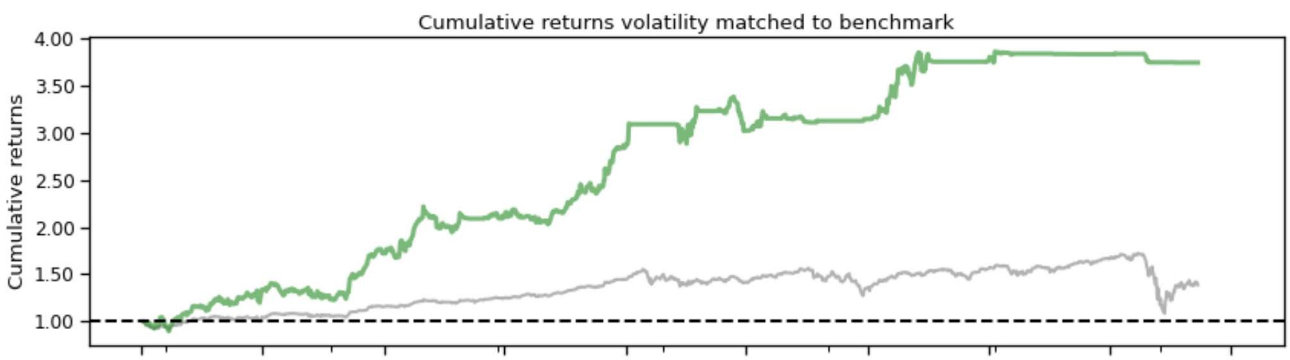
	Backtest
Annual return	13.162%
Cumulative returns	71.305%
Annual volatility	7.868%
Sharpe ratio	1.61
Calmar ratio	2.28
Stability	0.90
Max drawdown	-5.772%
Omega ratio	1.46
Sortino ratio	2.63
Skew	NaN
Kurtosis	NaN
Tail ratio	1.47
Daily value at risk	-0.941%
Alpha	0.11
Beta	0.14

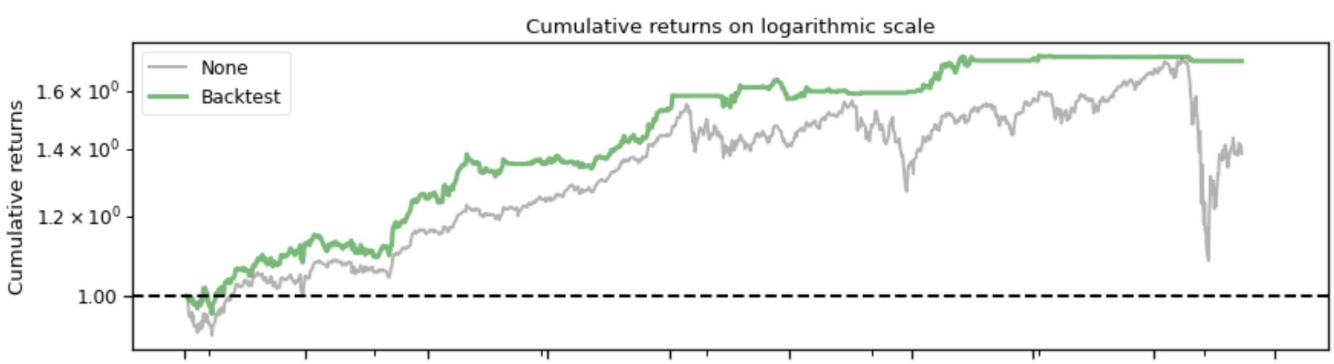
Worst drawdown periods	Net drawdown in %	Peak date	Valley date	Recovery date	Duration
0	5.77	2016-02-01	2016-02-11	2016-02-25	19
1	5.00	2016-07-14	2016-10-17	2016-11-10	86
2	4.96	2017-03-01	2017-04-13	2017-10-04	156
3	4.34	2018-06-12	2018-06-28	2019-02-04	170
4	3.85	2016-06-23	2016-06-27	2016-06-30	6

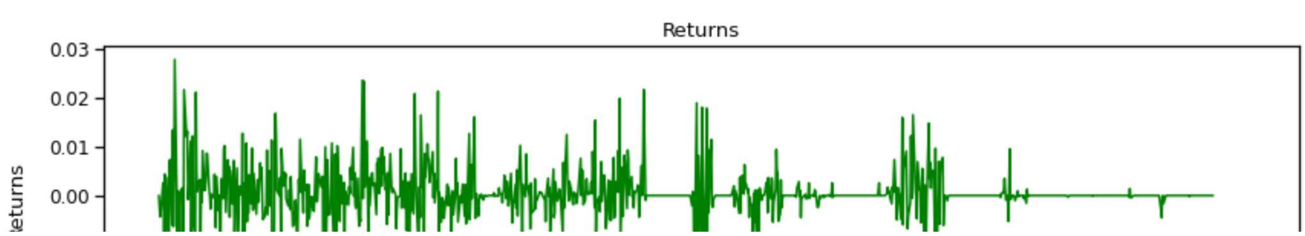
```
/Users/hongyangyang/anaconda3/lib/python3.6/site-packages/pyfolio/tears.py:907: UserWarr
  'interesting times.', UserWarning)
```




1.60
1.40
1.20
1.00
Cumulative returns


Cumulative returns volatility matched to benchmark
4.00
3.50
3.00
2.50
2.00
1.50
1.00
Cumulative returns


Cumulative returns on logarithmic scale
None
Backtest
1.6×10^0
1.4×10^0
1.2×10^0
1.00
Cumulative returns


Returns
0.03
0.02
0.01
0.00