<https://towardsdatascience.com/finrl-for-quantitative-finance-tutorial-for-single-stock-trading-37d6d7c30aac>

**FinRL for Quantitative Finance: Tutorial for Single Stock Trading**

**Note from Towards Data Science’s editors:** While we allow independent authors to publish articles in accordance with our [rules and guidelines](https://towardsdatascience.com/questions-96667b06af5), we do not endorse each author’s contribution. You should not rely on an author’s works without seeking professional advice. See our [Reader Terms](https://towardsdatascience.com/readers-terms-b5d780a700a4) for details.

This blog is a tutorial for our paper: **FinRL: A Deep Reinforcement Learning Library for Automated Stock Trading in Quantitative Finance**, presented at **NeurIPS 2020: Deep RL Workshop**.

## [FinRL: A Deep Reinforcement Learning Library for Automated Stock Trading in Quantitative Finance](https://arxiv.org/abs/2011.09607" \t "_blank)

### **[As deep reinforcement learning (DRL) has been recognized as an effective approach in quantitative finance, getting…](https://arxiv.org/abs/2011.09607" \t "_blank)**

[arxiv.org](https://arxiv.org/abs/2011.09607" \t "_blank)

The Jupyter notebook codes are available on our [**Github**](https://github.com/AI4Finance-LLC/FinRL-Library)and [**Google Colab**](https://colab.research.google.com/github/AI4Finance-LLC/FinRL-Library/blob/master/FinRL_single_stock_trading.ipynb).

## [AI4Finance-LLC/FinRL-Library](https://github.com/AI4Finance-LLC/FinRL-Library" \t "_blank)

### **[FinRL-Library, a DRL library designed specifically for automated stock trading with an effort to close sim-real gap…](https://github.com/AI4Finance-LLC/FinRL-Library" \t "_blank)**

[github.com](https://github.com/AI4Finance-LLC/FinRL-Library" \t "_blank)

## [Google Colaboratory](https://colab.research.google.com/github/AI4Finance-LLC/FinRL-Library/blob/master/FinRL_single_stock_trading.ipynb" \t "_blank)

### **[Edit description](https://colab.research.google.com/github/AI4Finance-LLC/FinRL-Library/blob/master/FinRL_single_stock_trading.ipynb" \t "_blank)**

[colab.research.google.com](https://colab.research.google.com/github/AI4Finance-LLC/FinRL-Library/blob/master/FinRL_single_stock_trading.ipynb" \t "_blank)

A more complete application of FinRL for multiple stock trading can be found in our [**previous blog**](https://towardsdatascience.com/deep-reinforcement-learning-for-automated-stock-trading-f1dad0126a02).

# Overview

As deep reinforcement learning (DRL) has been recognized as an effective approach in quantitative finance, getting hands-on experiences is attractive to beginners. However, to train a practical DRL trading agent that decides where to trade, at what price, and what quantity involves error-prone and arduous development and debugging.

We introduce a DRL library **FinRL** that facilitates **beginners** to expose themselves to quantitative finance and to **develop their own stock trading strategies**. Along with easily-reproducible tutorials, FinRL library allows users to streamline their own developments and to compare with existing schemes easily.

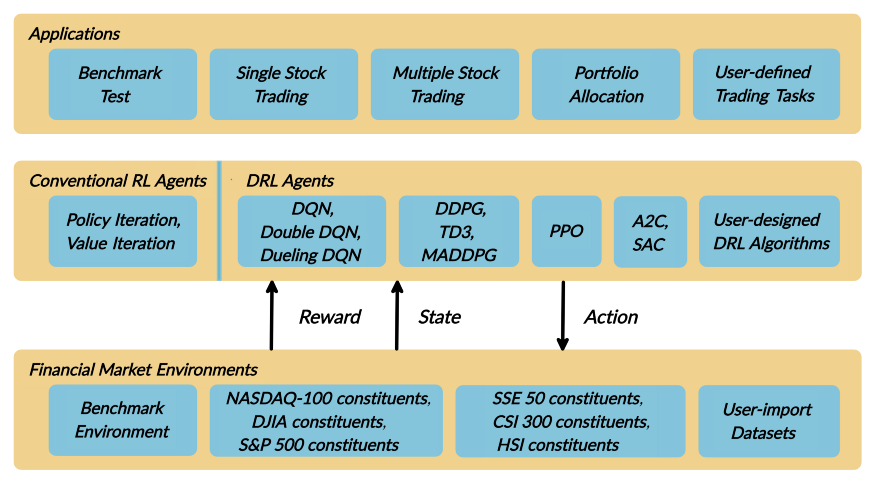
FinRL is a **beginner-friendly library** with fine-tuned standard DRL algorithms. It has been developed under three primary principles:

* **Completeness**. Our library shall cover components of the DRL framework completely, which is a fundamental requirement;
* **Hands-on tutorial**s. We aim for a library that is friendly to beginners. Tutorials with detailed walk-through will help users to explore the functionalities of our library;
* **Reproducibility**. Our library shall guarantee reproducibility to ensure the transparency and also provide users with confidence in what they have done.

This article is focusing on one of the use cases in our paper: **Single Stock Trading**. We use one Jupyter notebook to include all the necessary steps.

We use **Apple Inc.** stock: **AAPL** as an example throughout this article, because it is one of the most popular stocks.

https://miro.medium.com/max/60/1*nzN2C8uP0CJCgCNdhGglUA.png?q=20



Copyright by AI4Finance LLC

## **Table of Contents**

1. Problem Definition
2. Load Python Packages
3. Download Data
4. Preprocess Data
5. Build Environment
6. Implement DRL Algorithms
7. Backtesting Performance

# Part 1: Problem Definition

This problem is to design an automated trading solution for single stock trading. We model the stock trading process as a Markov Decision Process (MDP). We then formulate our trading goal as a maximization problem.

The components of the reinforcement learning environment are:

* **Action**: The action space describes the allowed actions that the agent interacts with the environment. Normally, a ∈ A includes three actions: a ∈ {−1, 0, 1}, where −1, 0, 1 represent selling, holding, and buying one stock. Also, an action can be carried upon multiple shares. We use an action space {−k, …, −1, 0, 1, …, k}, where k denotes the number of shares. For example, “Buy 10 shares of AAPL” or “Sell 10 shares of AAPL” are 10 or −10, respectively
* **Reward function**: r(s, a, s′) is the incentive mechanism for an agent to learn a better action. The change of the portfolio value when action a is taken at state s and arriving at new state s’, i.e., r(s, a, s′) = v′ − v, where v′ and v represent the portfolio values at state s′ and s, respectively
* **State**: The state space describes the observations that the agent receives from the environment. Just as a human trader needs to analyze various information before executing a trade, so our trading agent observes many different features to better learn in an interactive environment.
* **Environment**: single stock trading for AAPL

The data of the single stock that we will be using for this case study is obtained from Yahoo Finance API. The data contains Open-High-Low-Close price and volume.

# Part 2. Getting Started- Load Python Packages

Install the unstable development version of FinRL:

# Install the unstable development version in Jupyter notebook:  
**!pip install git+https://github.com/AI4Finance-LLC/FinRL-Library.git**

Install individual packages in Jupyter notebook if missing:

Import Packages:

# Part 3: Download Data

[**Yahoo Finance**](https://finance.yahoo.com/) is a website that provides stock data, financial news, financial reports, etc. All the data provided by Yahoo Finance is **free**. [**This Medium blog**](https://towardsdatascience.com/free-stock-data-for-python-using-yahoo-finance-api-9dafd96cad2e) explains how to use Yahoo Finance API to extract data directly in Python.

* **FinRL**uses a class **YahooDownloader** to fetch data from Yahoo Finance API
* **Call Limit**: Using the Public API (without authentication), you are limited to 2,000 requests per hour per IP (or up to a total of 48,000 requests a day).

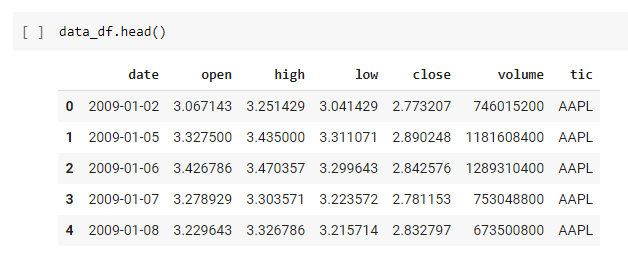
We can either download the stock data like **open-high-low-close price** manually by entering a **stock** **ticker** **symbol** like **AAPL** into the website search bar, or we just use Yahoo Finance API to extract data automatically.

FinRL uses a [**YahooDownloader**](https://github.com/AI4Finance-LLC/FinRL-Library/blob/master/finrl/marketdata/yahoodownloader.py)class to extract data.

**class YahooDownloader**:  
 """Provides methods for retrieving daily stock data from  
 Yahoo Finance API**Attributes**  
 ----------  
 **start\_date** : str  
 start date of the data (modified from config.py)  
 **end\_date** : str  
 end date of the data (modified from config.py)  
 **ticker\_list** : list  
 a list of stock tickers (modified from config.py)**Methods**  
 -------  
 **fetch\_data**()  
 Fetches data from yahoo API"""

**Download and save the data in a pandas DataFrame**:

https://miro.medium.com/max/60/1*5hX5TM-8_4eiTC3zQlTS8Q.png?q=20



# Part 4: Preprocess Data

**Data preprocessing** is a crucial step for training a high quality machine learning model. We need to check for missing data and do feature engineering in order to convert the data into a model-ready state.

* FinRL uses a **FeatureEngineer** class to preprocess the data
* Add **technical indicators**. In practical trading, various information needs to be taken into account, for example the historical stock prices, current holding shares, technical indicators, etc.

## **Calculate technical indicators:**

In practical trading, various information needs to be taken into account, for example the historical stock prices, current holding shares, technical indicators, etc.

* FinRL uses stockstats to calcualte technical indicators such as **Moving Average Convergence Divergence (MACD)**, **Relative Strength Index (RSI)**, **Average Directional Index (ADX)**, **Commodity Channel Index (CCI)** and other various indicators and stats.
* **stockstats**: supplies a wrapper StockDataFrame based on the **pandas.DataFrame** with inline stock statistics/indicators support.
* we store the stockstats technical indicator column names in **config.py**
* config.TECHNICAL\_INDICATORS\_LIST = [‘macd’, ‘rsi\_30’, ‘cci\_30’, ‘dx\_30’]
* User can add more technical indicators, check <https://github.com/jealous/stockstats> for different names

FinRL uses a [**FeatureEngineer**](https://github.com/AI4Finance-LLC/FinRL-Library/blob/master/finrl/preprocessing/preprocessors.py)class to preprocess data.

**class FeatureEngineer**:  
 """Provides methods for preprocessing the stock price data**Attributes**  
 ----------  
 **df**: DataFrame  
 data downloaded from Yahoo API  
 **feature\_number** : int  
 number of features we used  
 **use\_technical\_indicator** : boolean  
 we technical indicator or not  
 **use\_turbulence** : boolean  
 use turbulence index or not**Methods**  
 -------  
 **preprocess\_data**()  
 main method to do the feature engineering"""

## **Perform Feature Engineering:**

# Part 5: Build Environment

Considering the stochastic and interactive nature of the automated stock trading tasks, a financial task is modeled as a **Markov Decision Process (MDP)**problem. The training process involves observing stock price change, taking an action and reward’s calculation to have the agent adjusting its strategy accordingly. By interacting with the environment, the trading agent will derive a trading strategy with the maximized rewards as time proceeds.

Our trading environments, based on **OpenAI Gym framework**, simulate live stock markets with real market data according to the principle of time-driven simulation.

Environment design is one of the most important part in DRL, because it varies a lot from applications to applications and from markets to markets. **We can’t use an environment for stock trading to trade bitcoin, and vice versa**.

The action space describes the allowed actions that the agent interacts with the environment. Normally, action a includes three actions: **{-1, 0, 1}**, where -1, 0, 1 represent **selling, holding, and buying** **one** **share**. Also, an action can be carried upon multiple shares. We use an action space **{-k,…,-1, 0, 1, …, k}**, where k denotes the number of shares to buy and -k denotes the number of shares to sell. For example, “Buy 10 shares of AAPL” or “Sell 10 shares of AAPL” are 10 or -10, respectively. The continuous action space needs to be normalized to **[-1, 1]**, since the policy is defined on a Gaussian distribution, which needs to be normalized and symmetric.

In this article, I set k=2**00**, the entire action space is 200\*2+1 = 401 for AAPL.

FinRL uses a [**EnvSetup**](https://github.com/AI4Finance-LLC/FinRL-Library/blob/master/finrl/env/environment.py)class to setup environment.

**class EnvSetup**:  
 """Provides methods for retrieving daily stock data from  
 Yahoo Finance API**Attributes**  
 ----------  
 **stock\_dim**: int  
 number of unique stocks  
 **hmax** : int  
 maximum number of shares to trade  
 **initial\_amount**: int  
 start money  
 **transaction\_cost\_pct** : float  
 transaction cost percentage per trade  
 **reward\_scaling**: float  
 scaling factor for reward, good for training  
 **tech\_indicator\_list**: list  
 a list of technical indicator names (modified from config.py)**Methods**  
 -------  
 **create\_env\_training**()  
 create env class for training  
 **create\_env\_validation**()  
 create env class for validation  
 **create\_env\_trading**()  
 create env class for trading"""

Initialize an environment class:

**User-defined Environment**: a simulation environment class.

FinRL provides blueprint for [**single stock trading environment**](https://github.com/AI4Finance-LLC/FinRL-Library/blob/master/finrl/env/EnvSingleStock.py)**.**

**class SingleStockEnv**(gym.Env):  
 """A single stock trading environment for OpenAI gym **Attributes**  
 ----------  
 **df**: DataFrame  
 input data  
 **stock\_dim** : int  
 number of unique stocks  
 **hmax** : int  
 maximum number of shares to trade  
 **initial\_amount** : int  
 start money  
 **transaction\_cost\_pct**: float  
 transaction cost percentage per trade  
 **reward\_scaling**: float  
 scaling factor for reward, good for training  
 **state\_space**: int  
 the dimension of input features  
 **action\_space**: int  
 equals stock dimension  
 **tech\_indicator\_list**: list  
 a list of technical indicator names  
 **turbulence\_threshold**: int  
 a threshold to control risk aversion  
 **day**: int  
 an increment number to control date **Methods**  
 -------  
 **\_sell\_stock**()  
 perform sell action based on the sign of the action  
 **\_buy\_stock**()  
 perform buy action based on the sign of the action  
 **step**()  
 at each step the agent will return actions, then   
 we will calculate the reward, and return the next   
 observation.  
 **reset**()  
 reset the environment  
 **render**()  
 use render to return other functions  
 **save\_asset\_memory**()  
 return account value at each time step  
 **save\_action\_memory**()  
 return actions/positions at each time step"""

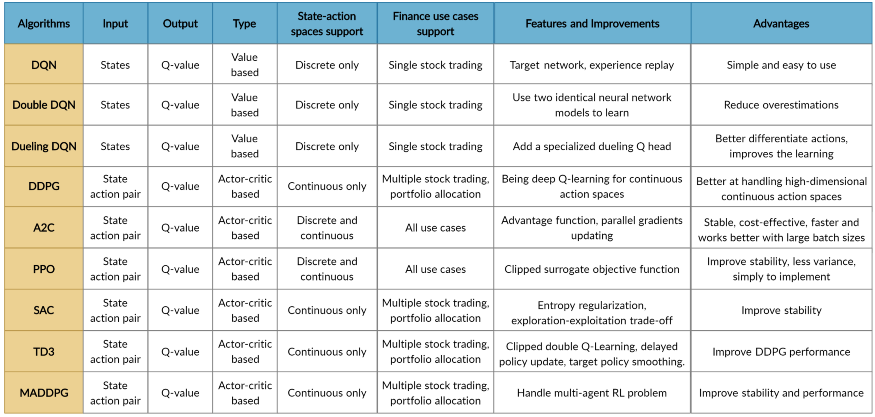
I will write another article to teach how to design a customized trading environment in the future.

# Part 6: Implement DRL Algorithms

The implementation of the DRL algorithms are based on [**OpenAI Baselines**](https://github.com/openai/baselines)and [**Stable Baselines**](https://github.com/hill-a/stable-baselines). Stable Baselines is a fork of OpenAI Baselines, with a major structural refactoring, and code cleanups.

* FinRL library includes fine-tuned standard DRL algorithms, such as DQN, DDPG, Multi-Agent DDPG, PPO, SAC, A2C and TD3. We also allow users to design their own DRL algorithms by adapting these DRL algorithms.

https://miro.medium.com/max/60/1*-IYr8MKNheVNwTjBNrKtFA.png?q=20



Copyright by AI4Finance LLC

FinRL uses a [**DRLAgent**](https://github.com/AI4Finance-LLC/FinRL-Library/blob/master/finrl/model/models.py)class to implement the algorithms.

**class DRLAgent:** """Provides implementations for DRL algorithms**Attributes**  
 ----------  
 **env**: gym environment class  
 user-defined class**Methods**  
 -------  
 **train\_PPO**()  
 the implementation for PPO algorithm  
 **train\_A2C**()  
 the implementation for A2C algorithm  
 **train\_DDPG**()  
 the implementation for DDPG algorithm  
 **train\_TD3**()  
 the implementation for TD3 algorithm   
 **DRL\_prediction**()   
 make a prediction in a test dataset and get results  
 """

## **Model Training:**

We use 4 DRL models in this article, namely [**PPO**](https://arxiv.org/abs/1707.06347)**,**[**A2C**](https://arxiv.org/abs/1602.01783)**,**[**DDPG**](https://arxiv.org/abs/1509.02971)**, and**[**TD3**](https://spinningup.openai.com/en/latest/algorithms/td3.html#background)**.**I introduced these models in the[**previous article**](https://towardsdatascience.com/deep-reinforcement-learning-for-automated-stock-trading-f1dad0126a02)**.**[**TD3**](https://towardsdatascience.com/td3-learning-to-run-with-ai-40dfc512f93)is an improvement over **DDPG**.

## **Tensorboard: reward and loss function plot**

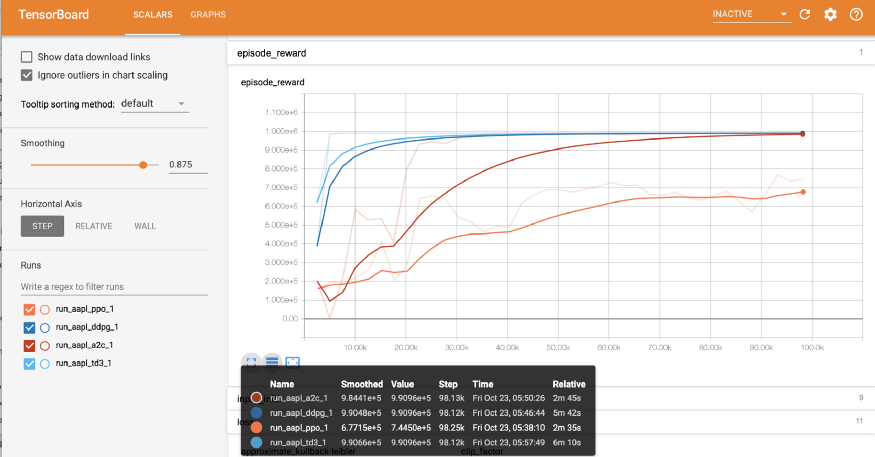
We use [**tensorboard integration**](https://stable-baselines.readthedocs.io/en/master/guide/tensorboard.html) for hyperparameter tuning and model picking. Tensorboard generates nice looking charts.

Once the learn function is called, you can monitor the RL agent during or after the training, with the following bash command:

# cd to the tensorboard\_log folder, run the following command   
**tensorboard --logdir ./A2C\_20201127-19h01/**# you can also add past logging folder  
**tensorboard --logdir ./a2c\_tensorboard/;./ppo2\_tensorboard/**

**Total rewards for each of the algorithm:**

https://miro.medium.com/max/60/1*lh_CP1Fa4aqZgDEV6LCvKQ.png?q=20



Copyright by AI4Finance LLC

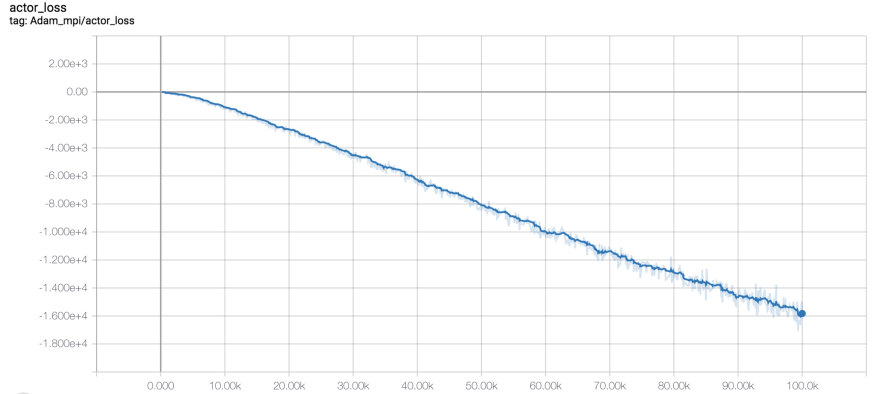
[**total\_timesteps (int)**](https://stackoverflow.com/questions/56700948/understanding-the-total-timesteps-parameter-in-stable-baselines-models): the total number of samples to train on. It is one of the most important hyperparameters, there are also other important parameters such as **learning rate, batch size, buffer size**, etc.

To compare these algorithms, I set the **total\_timesteps = 100k**. If we set the total\_timesteps too large, then we will face a risk of overfitting.

By observing the **episode\_reward** chart, we can see that these algorithms will converge to an optimal policy eventually as the step grows. TD3 converges very fast.

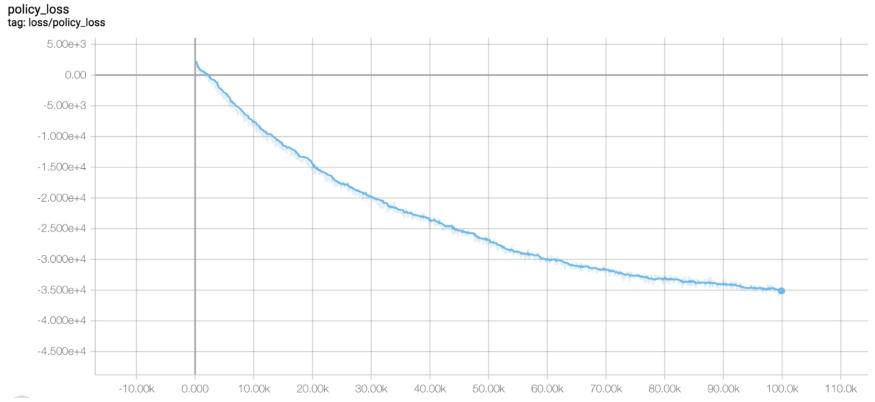
**actor\_loss for DDPG and policy\_loss for TD3:**

https://miro.medium.com/max/60/1*sWcIoCym6pylXwir8Fz_GA.png?q=20



Copyright by AI4Finance LLC

https://miro.medium.com/max/60/1*oOPeZ8l8G_V49eH0PtN-Ng.png?q=20



Copyright by AI4Finance LLC

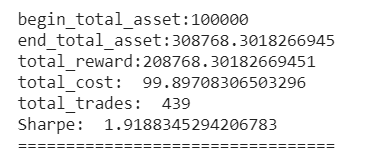
## **Picking models:**

I pick the **TD3** model, because it converges pretty fast and it’s a state of the art model over DDPG. By observing the **episode\_reward** chart, **TD3** doesn’t need to reach full 100k total\_timesteps to converge.

## **Trading:**

Assume that we have **$100,000 initial capital**at 2019/01/01. We use the **TD3** model to trade **AAPL**.

https://miro.medium.com/max/60/1*wDKKIhUvaEyur30xqnme-w.png?q=20

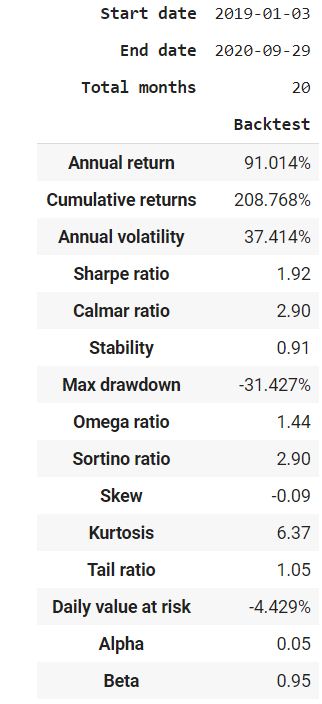


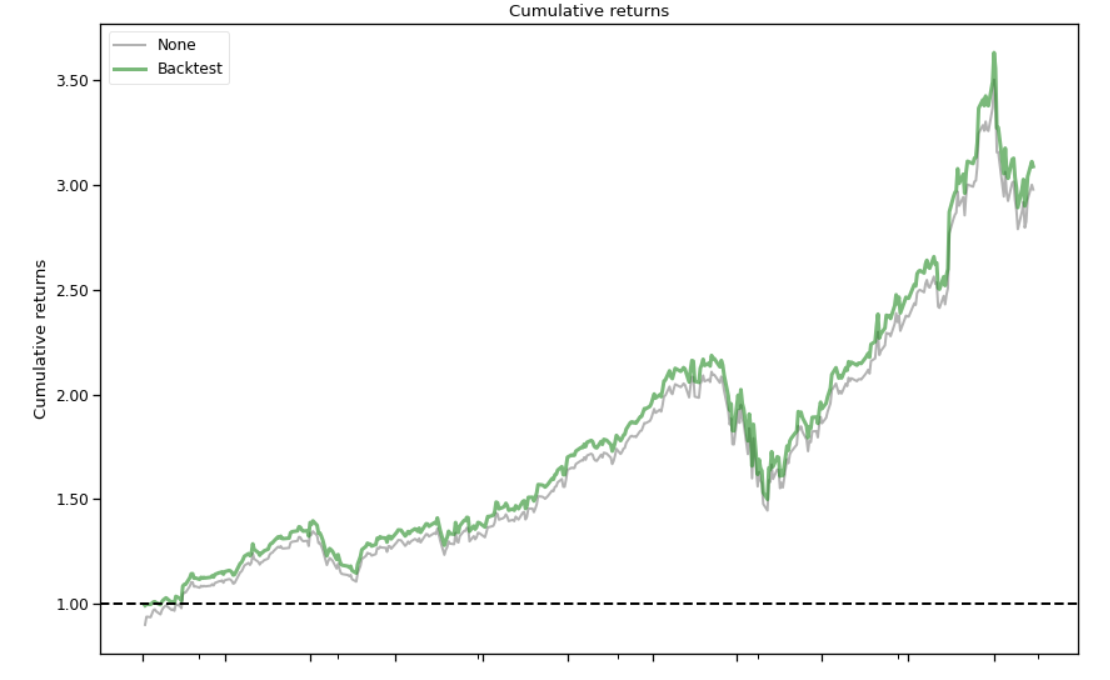
# Part 7: Backtesting Performance

**Backtesting**plays a key role in evaluating the performance of a trading strategy. Automated backtesting tool is preferred because it reduces the human error. We usually use the **Quantopian** [**pyfolio**](https://github.com/quantopian/pyfolio) package to backtest our trading strategies. It is easy to use and consists of various individual plots that provide a comprehensive image of the performance of a trading strategy.

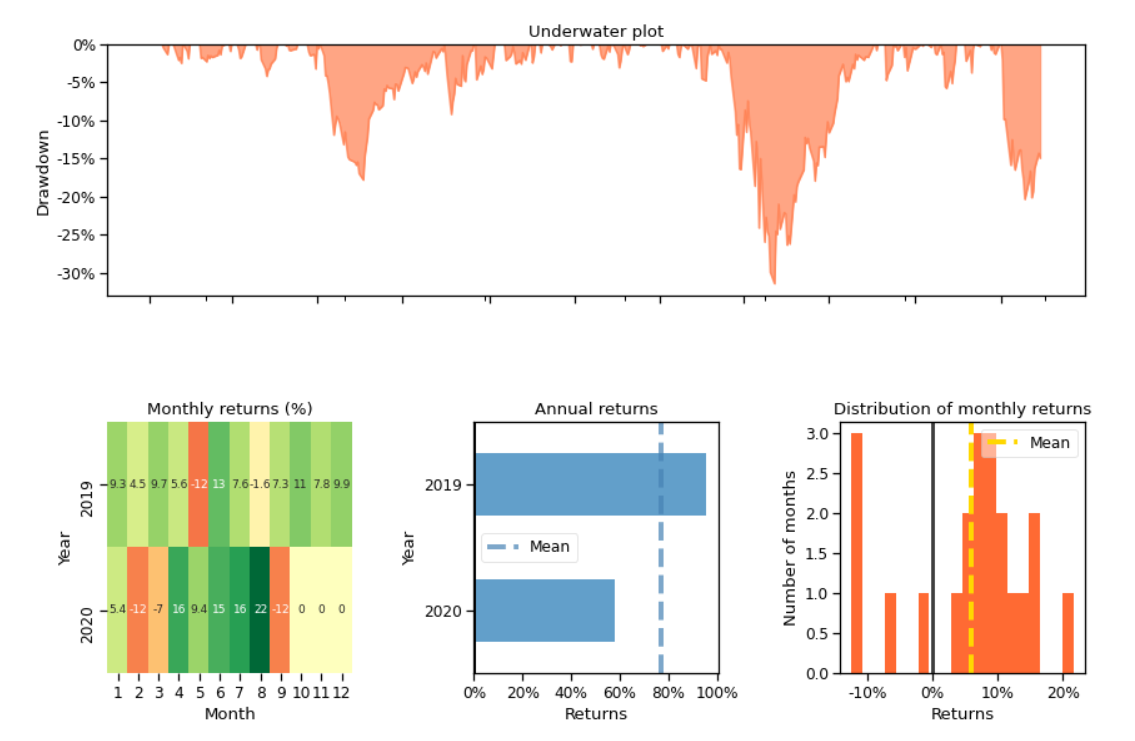
FinRL uses a[**set of functions**](https://github.com/AI4Finance-LLC/FinRL-Library/blob/master/finrl/trade/backtest.py)to do the backtesting with pyfolio.

## **Plots:**





Copyright by AI4Finance LLC



Copyright by AI4Finance LLC

# Conclusion

Congratulations, we have covered everything you need to start implementing DRL for single stock trading. We use Jupyter notebook in Python to walk through the training and trading process step by step.

**Contributions of FinRL:**

* FinRL is an open source library specifically designed and implemented for quantitativefinance. Trading environments incorporating market frictions are used and provided.
* Trading tasks accompanied by hands-on tutorials with built-in DRL agents are available in a beginner-friendly and reproducible fashion using Jupyter notebook. Customization of trading time steps is feasible.
* FinRL has good scalability, with a broad range of fine-tuned state-of-the-art DRL algorithms. Adjusting the implementations to the rapid changing stock market is well supported.
* Typical use cases are selected and used to establish a benchmark for the quantitative finance community. Standard backtesting and evaluation metrics are also provided for easy and effective performance evaluation.

I hope you found this article helpful and learned something about using DRL for stock trading!

Please report any issues to our [**Github**](https://github.com/AI4Finance-LLC/FinRL-Library/issues).