Coding Guidelines Tutorials

Contents

Get prepared

1. Download and install VSCode in your device

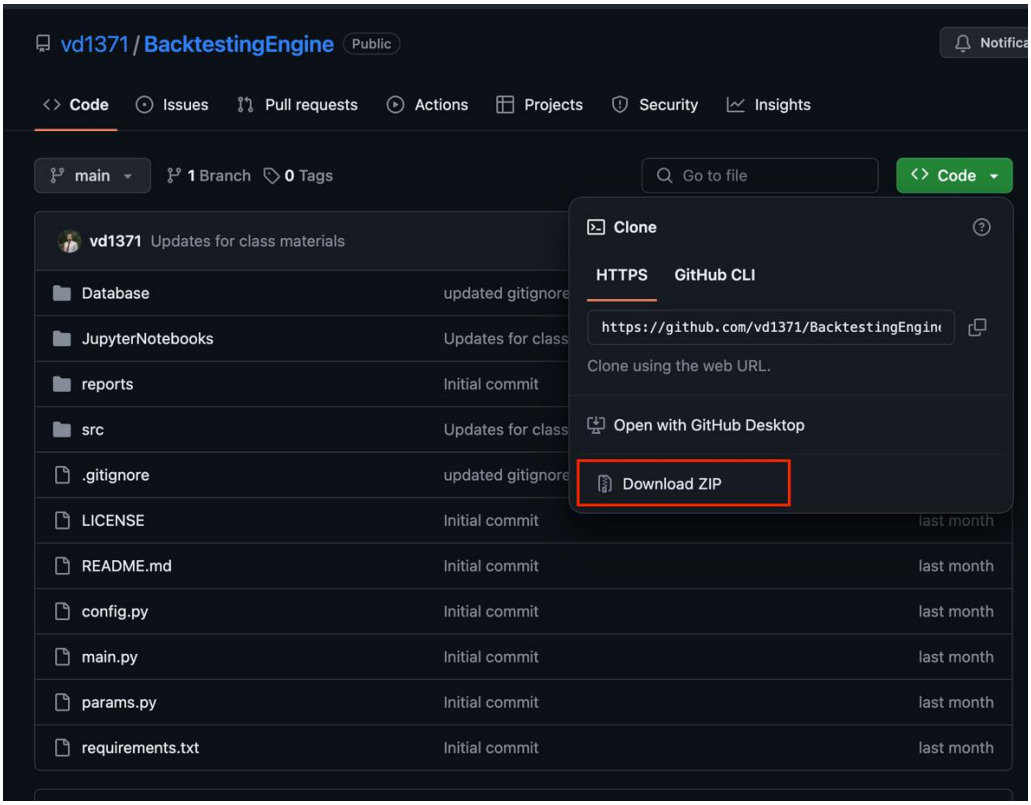
   Link: https://code.visualstudio.com/download

2. Go to Github and find the BacktestingEngine repository

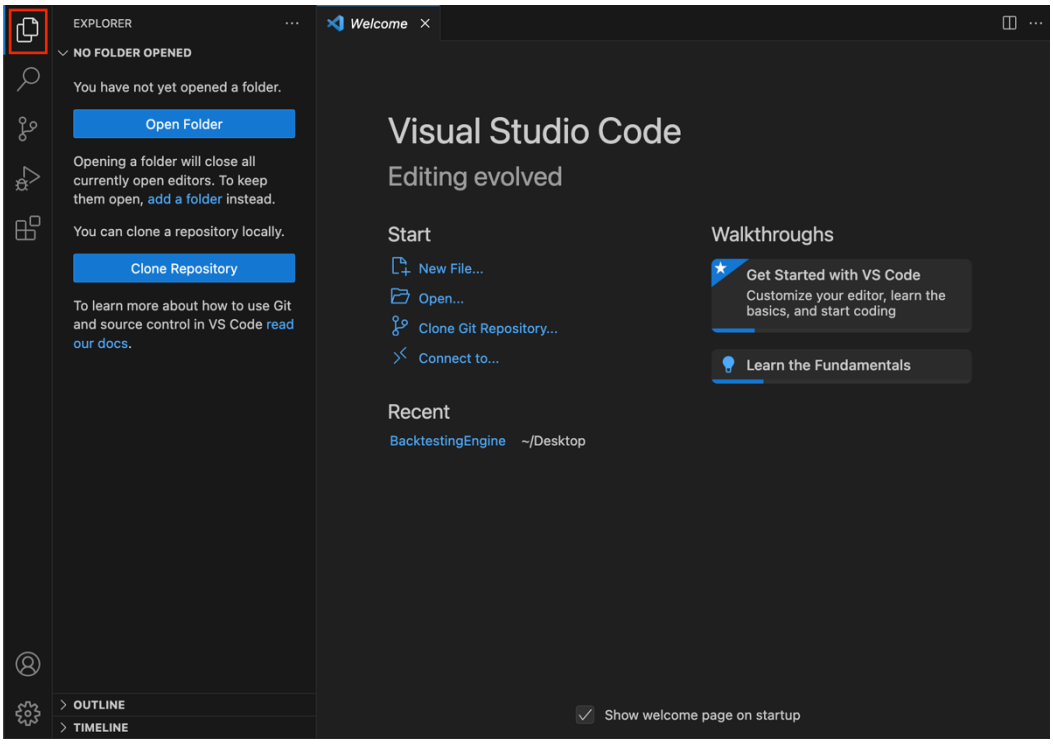
   Link: https://github.com/vd1371/BacktestingEngine.git

3. Download the ZIP, and extract the files from the zip to create a new folder

   Or you can directly clone the url from github to VSCode



4. Open the VSCode, select Explorer

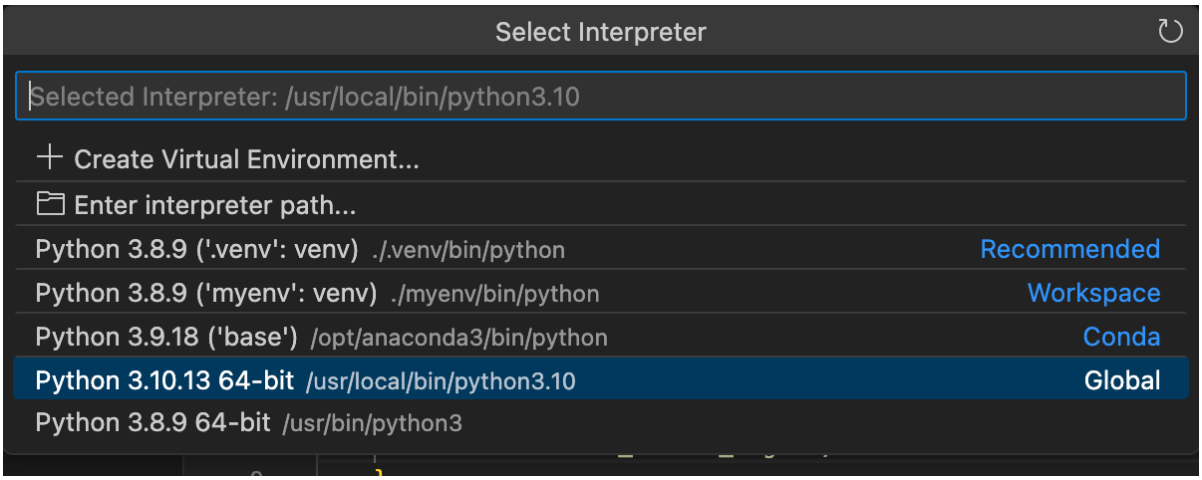


There will be two options:

1. For open folder, click and select the downloaded BacktestingEngine folder
2. For Clone Repository, clone the url you copy from github

In this case, you have set up the environment for coding.

[Notice: please install python3.10 or above version, 3.9 won't work for this project]



Install modules.

Right click ***main.py*** to open in an integrated terminal.

In the terminal, input: ***python-m pip install -r requirements.txt***

Press enter.

Then you can try run the ***main.py*** to check the consistency.

How to simulate in a new universe?

For a new universe, we mean a different market. In **params.py**, you can check that ***market = "US"***, that means we are currently trading on the US market. The list of stocks are in ***Database>Symbols.csv***, currently with 3 different markets: US, HK, and Crypto.

Let's say, you want to switch to Hong Kong market, first, make sure that the ***HK*** column in ***Symbols.csv*** is complete. Then, simply switch the market from ***market = "US"***, to ***market = "HK"***.

Moreover, if you want to invest in other market such as UK, you shall modify in the ***Symbols.csv***, create a new column for UK, and input the codes of stocks. Finally, define ***market = "HK"***.

How to add a new signal for a new trading strategy?

You can find signals folder in ***src>Simulator>Signals***, where you can modify or develop your own strategies to build trading signals in this folder.

At this point, we offer you a signal called ***add_random_signal()***, where you can find in ***_add_random_signal.py***. then we import the function in ***get_alpha_signal_func.py*** file, to build a new function called ***get_alpha_signal_func()***.

You can see in the definition of the function, we build a dictionary called ***strategy_signal_func_dic{}*** to collect and store all the signals that we build. We name the ***add_random_signal*** as 'random'.

Back to ***params.py***, we define the ***strategy_name='random'***, in which case, we apply the strategy signal ***add_random_signa***l to our trading simulation.

To change a new strategy signal, first, create a new .py file in Signals folder, where you can build your trading signal. Then in ***get_alpha_signal_func.py***, import the function you build in the new .py file, ***from .xxxx import xxxx***. Then name and add your new signal function in the dictionary ***strategy_signal_func_dic{}***. ***'name of your signal'=xxxx***. Finally, back to ***params.py***, apply your new strategy, ***strategy_name = 'xxxx'***. Run main.py to see the simulation results

The structure of the data

To view the structure of the data, you can check the following files:

For the structure of ***df***, go to reports->US -> (Whatever strategy)->ExecutedTrades.csv, where you can find the columns listed in **df**

For the output of the portfolio, i.e. ***df_g***, you can refer to reports->US -> (Whatever strategy)->DailyBudget.csv, to check the structure of that data frame.

How to add statistical measures and figures

Sharpe ratio, skewness, and kurtosis

The formats are simple, here I will show you how to modify the output summary report in the project.

In get_statistical_summary_of_trades.py, we define the function to generate the reports and statistical measures for our trade. In ***def add_statistical_reports():***, we need to add the code to generate the sharpe ratio, skewness, and kurtosis. Here is an example of how I implement the format in this function:

```
df_g['log_return'] = np.log(df_g['PortfolioValue']/df_g['PortfolioValue'].shift(1))
mu = df_g['log_return'].mean()
sigma=df_g.log_return.std()
summary['sharpe_ratio']= (mu/sigma)*(252**0.5)
summary['skewness']= df_g['log_return'].skew()
summary['kurtosis']= df_g['log_return'].kurt()
```

As you can see, I first generate a new column in ***df_g***, with the logarithm transformation on the return of the '***PortfolioValue'***. Then compute the mu and sigma for it, in which case prepare all the variables needed for those measures. For skewness and kurtosis, I directly call the functions in ***pandas***.

After redeploying the module, you can simply run the main.py, this shall be what the INFO session is like:

```
-----------------------------------------------------------
                Market   win  long_win  short_win  Trades  n_long  n_short  return(%) annual(%)  sharpe_ratio  skewness  kurtosis
0                   US  0.49      0.56       0.42    1124     589      535     -46.76     -11.8         -1.18     -0.19      2.51
1  US-25/02/19-24/02/20  0.50      0.62       0.35     166      92       74      -6.77                   -0.83     -0.89      5.66
2  US-24/02/20-23/02/21  0.48      0.56       0.39     272     141      131     -14.33                   -1.31      0.04      2.30
3  US-23/02/21-23/02/22  0.47      0.55       0.36     180     100       80     -18.73                   -2.14     -0.78      1.41
4  US-23/02/22-22/02/23  0.52      0.51       0.53     311     161      150      -8.66                   -0.72     -0.28      1.06
5  US-22/02/23-22/02/24  0.49      0.61       0.37     195      95      100     -10.58                   -1.10      0.58      3.48
-----------------------------------------------------------
```

Moreover, you can add whatever other statistical measures that you want to observe in your portfolio trading by replicating the process above. The logic and structure of modifying the code are exactly the same.

Histogram and QQ plot

To add figures for your trade, go to src->Simulator->AlphaSimulator->ReportingUtils, where you can find some of the .py files to generate figures such as plot_duration_of_net_exposure.py, plot_the_budget_vs_time.py. You can add another .py file to plot the figures you need. Here I will show you an example of how to generate the histogram for the portfolio.

1. Create a file in src\Simulator\AlphaSimulator\ReportingUtils, name it plot_histogram.py, where we define the function of ***plot_histogram():***

Here I will show you how I write this function for your reference. You are also highly encouraged to develop your way of coding.

```
import matplotlib.pyplot as plt
import numpy as np
import os
def plot_histogram(df_g, **params):
    enums = params['enums']

    mu = df_g['log_return'].mean()  # mean of distribution
    sigma = df_g['log_return'].std()  # standard deviation of distribution

    x = df_g['log_return'].values
    num_bins = 100
    n, bins, patches = plt.hist(x, num_bins,
                            density = 1,
                            color ='blue',
                            alpha = 0.7)
    y = ((1 / (np.sqrt(2 * np.pi) * sigma)) * np.exp(-0.5 * (1 / sigma * (bins - mu))**2))

    plt.plot(bins, y, '--', color ='black')
    plt.xlabel('log returns')
    plt.ylabel('Frequency')
    plt.title('Histogram of the log return')
    plt.savefig(os.path.join(enums.STAT_FIGURES_DIR, "LogReturnHistogram.png"))
    plt.close()
```

In this code, other than the histogram plot, I also draw a normal fitted curve for the distribution with ***plt.plot()***, which is optional for you to learn.

2. After defining your functions in this module, you need to import the function into the project. So in ReportingUtils->_init_.py, add this:

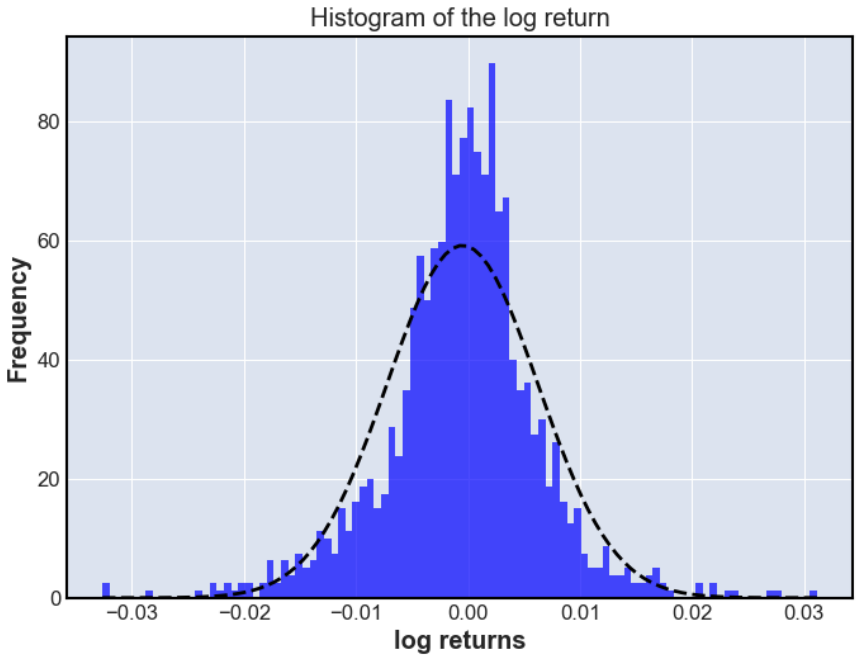
```
from .plot_histogram import plot_histogram
```

to import the built function. Then in src\Simulator\AlphaSimulator\generate_report_for_trades_history.py, you should first import the function `from .ReportingUtils import plot_histogram`, then in:

```
if should_log:
        draw_trades_executions(stock_histories, **params)
        plot_duration_of_net_exposure(df_g, **params)
        plot_histogram(df_g, **params)
```

to add that function.

3. Finally, run main.py. In reports\US\random\StatReports, you shall check that image.



The steps for creating a QQ plot are the same as the above.

The only thing you need to take care of is to standardize your sample, i.e. the portfolio return. Since generally the return of the output PortfolioValue is not bell shape, that will cause the QQ plot to be extremely flat.

Here I will still provide you with my code for plotting QQ plot for your reference:

```python
import statsmodels.api as sm
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale
import numpy as py
import os

def plot_qq(df_g, **params):
    enums = params['enums']
    #standardize the logreturn with scalre() imported from sklearn.preprocessing
    data=scale(df_g['log_return'])
    #create Q-Q plot with 45-degree line added to plot
    sm.qqplot(data,line='45')
    plt.xlabel("Theoretical Quantiles")
    plt.ylabel("Sample Quantiles")
    plt.title('QQ plot of the log return')
    plt.savefig(os.path.join(enums.STAT_FIGURES_DIR, "QQReturnHistogram.png"))
    plt.close()
```

And this shall be the QQ plot:

Tips: to avoid the unchecked status of your project, in the terminal use the commands:

git add -A - To add all the files to the staging area.

git commit -m 'message' - To create a 'snapshot' of the files on the staging area.