A Simple Back Test System of Quantitative Trading Realized by Jupyter Notebook & Spark

Sujia Yin, Jiahao Shi

Electronics and Computer Engineering
University of Waterloo
Waterloo, Canada

I. Introduction and Project Objective

The market is one of the greatest inventions ever made by mankind. Adam Smith used the words "invisible hand" in Wealth of Nations to vividly explain how the market and rational people interact with each other and ultimately benefit the entire society. The core of the market is exchange, and the development of human society begins with the exchange of goods and materials. The original way of "exchanging goods with goods" was created before the birth of currency. However, this method is very inefficient and inconvenient for circulation. Therefore, universal equivalent has appeared and social division of labor has gradually emerged. People exchange the goods they produce for universal equivalent, and then exchange them for other goods they need.

The core of the exchange is buying and selling. When the buyer and seller expect the price to be equal, the transaction is concluded. With the development of finance and technology, more and more financial instruments such as stocks, bonds, options, and futures have gradually appeared, and those financial derivatives have also developed to be more and more complicated. Does the market have any rules to follow? Sure it is, but there are only some traces to follow instead of a definite rule. The diversity of transactions and the complexity of human nature make financial data extremely noisy. We cannot simply infer market changes from one or two factors. One could make wrong decision easily because of his or her psychological weakness. So, is there any way to circumvent this type of risk?

The answer is quantitative trading. Quantitative trading usually refers to the use of mathematics, statistics or even machine learning methods to find the right time to buy and sell. Under the definition of this dimension, algorithmic trading, high-frequency trading, and statistical arbitrage can be counted as quantitative trading. The benefits of quantifying trading are obvious. In the most intuitive way, the computer does not sleep, and does not require traders to operate in real time, satisfying the vision of making money by doing nothing. Common electronic transactions (such as stocks and digital currencies) directly send buy and sell requests to the exchange through a broker or software; The bottom layer of algorithmic trading is to let programs automatically implement such operations. Brokers or exchanges usually provide API interfaces to investors.

For algorithmic trading systems, the API just lays at the lowest level. In general, a basic quantitative trading system should include: market module, strategy module, transaction module and back test module. Their relationship is roughly as follow:

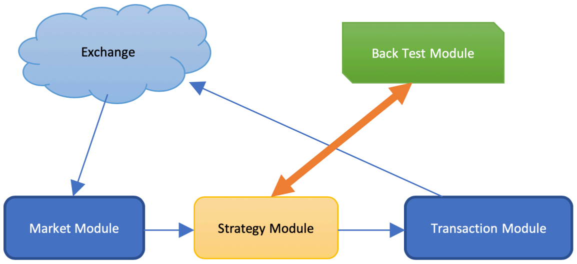


Fig 1. Components of a basic quantitative trading system

Their functions are described as follow:

- Market module: Try to obtain market data; Usually also be responsible for obtaining the status of the trading account.
- Strategy module: Subscribe to market data and issue buy and sell instructions to the transaction module according to a particular algorithm.
- Transaction module: Accept and encapsulate the buy and sell instructions sent by the strategy module and forward them to the exchange; At the same time monitor and ensure the complete execution of the buy and sell instructions.
- Back test module: Be used to aid strategy development by testing the algorithm on old price

dataset and evaluating the performance like total profits and efficacy.

In this project, we will use Python to implement a simple back test system with a strategy module, and use SMA (Simple Moving Average) as the core algorithm to simulate the buying and selling behavior based on past stock price data. Here we have two main objectives:

1) We are going to use the “ipywidgets” module to create an interactive interface to change parameters and display outputs. With the input widgets and output charts we can easily modify some key parameters of the strategy, set the status of our virtual account and analyze the output data and charts, which gives us an intuitive feeling on the strategy performance.

2) We will select some sample data sets for testing and analysis. With some different combinations of parameters, we get some interesting features of SMA strategy by which we evaluate the algorithm performance and estimate the using condition and validation of this strategy.

II. SMA Cross Strategy

The development of trading strategies is very complicated. Because market prices are the result of a combination of internal and external conditions, trading strategies also need to include as many various factors as possible. Since we are implementing a simple strategy module, we just start from the price curve trend and adopt a simple strategy - Simple Moving Average Cross Strategy.

In order to understand this strategy, let's first understand what is called Simple Moving Average (SMA for short). Suppose there is a sequence X that has N numbers x [0], x [1], ..., x [N-1].The average of these N numbers could be written as:

$$A(X) = \frac{\sum_{i=0}^{N-1} X[i]}{N}$$

Now we pick a relatively small number K that is much smaller than N, and use a sliding window of size K to slide over the original sequence. By averaging the K elements that are captured each time, we can get the SMA value of the original sequence with a window of size K. Suppose the new sequence is XSMA, the jth element of XSMA can be written as:

$$X_{SMA}[j] = A(X[j-K+1:j]) = \frac{\sum_{i=j-K+1}^{j} X[i]}{K}$$

SMA is essentially a simple smoothing of the original sequence. It is easy to understand that if the size of window is larger, the SMA curve should be smoother and change more slowly; conversely, if the SMA is smaller, the short-term changes of original sequence will be reflected on the SMA curve. According to this feature, we just introduce two indicators for the price of an investment product: One is a SMA sequence with a small window, and the other is another SMA sequence with a large window. If the SMA curve of the small window pierces or passes through the SMA of the large window from below, it means that the price of this investment product rises rapidly in the short term, which may be a signal to buy; Otherwise, if the large window The SMA broke through the small window SMA from the top, then it means that the price of investment product has fallen rapidly in the short term, and we should consider selling out. Fig 2 shows the process described above.

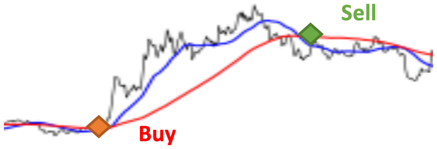


Fig 2. Example of SMA cross strategy

III. Architecture of the Back Test System

In our back test system, there are three classes and several independent functions that operate closely with the classes. However, before talking about the architecture of back test system, we need to take a quick look at the format of source dataset.

In quantitative trading the most useful data are open price, highest price, lowest price, close price and the trading volume. Any price curve must have these five indicators in any period of time, such as one minute, one hour, one day, one month… All the five indicators together comprise the data format called OHLCV. In our project, we read a bunch of tables and turn them into this type of format. Actually, we only use the column of “close price” as our price sequence. Thus our system starts with reading a such kind of file.

In the system we use a function “read_file()” to read .csv files from current directory. This function transforms input file to a Spark dataframe, format it to be an OHLCV and then clean the content data. After the input data is processed properly, it is passed to a class called “BackTest”. In this class we create instances of another two classes – “SmaCross” and “Exchange”. The data then will be passed to both instances to make some preparations. The “SmaCross” calls function “SMA()” and takes as input the data and two different integers (representing window size) to calculate two SMA curves, while the “Exchange” updates current price at each time tick by the input data. Finally “BackTest” calls internal function “run()” to traverse the whole input sequence. If there is an intersection between the two SMA curves, the system then calls “buy()” or “sell()” to execute the transaction. The basic components and their interactions are shown in Fig 3.

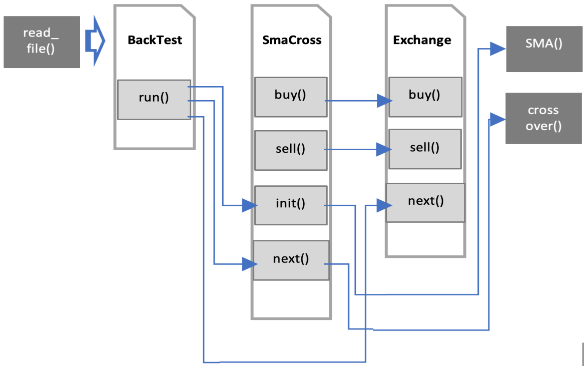


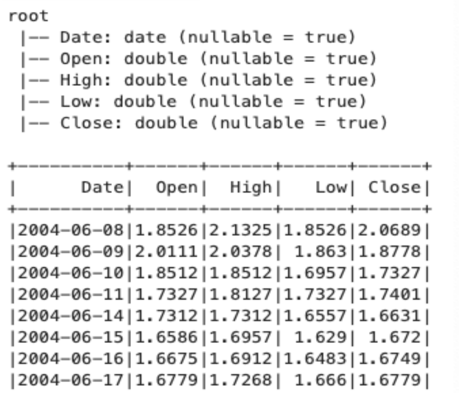
Fig 3. Architecture of the back test system and the interaction between major components

IV. Parallelization of Core Functions

Among the core functions of the back test system, there are two functions that can be parallelized by using Spark to improve computing efficiency. These two functions are "read_file ()" and "SMA ()". Here, "read_file ()" reads and formats the .csv files through Spark dataframe, while "SMA ()" uses Spark RDD to perform SMA calculations on the input sequence. The process of reading and processing input data is as follows:

- Create Spark dataframe object “sdf” via using SparkSession;
- Read the .csv files through “sdf” to get the raw dataframe – “df_raw”;
- Drop unused column data and assign the dataframe a new header;
- Update the data type of the current field according to the required schema, then sort the price data by date;
- Filter out rows with empty values and return the final dataframe.

Fig 4 indicates the processed schema and table data samples according to the above process.



```
root
 |-- Date: date (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)

+----------+------+------+------+------+
|      Date|  Open|  High|   Low| Close|
+----------+------+------+------+------+
|2004-06-08|1.8526|2.1325|1.8526|2.0689|
|2004-06-09|2.0111|2.0378| 1.863|1.8778|
|2004-06-10|1.8512|1.8512|1.6957|1.7327|
|2004-06-11|1.7327|1.8127|1.7327|1.7401|
|2004-06-14|1.7312|1.7312|1.6557|1.6631|
|2004-06-15|1.6586|1.6957| 1.629| 1.672|
|2004-06-16|1.6675|1.6912|1.6483|1.6749|
|2004-06-17|1.6779|1.7268| 1.666|1.6779|
```

Fig 4. The schema and table content sample of processed input data

The main function of "SMA ()" is essentially to calculate the moving average of the input sequence. Actually, By using "rolling ().mean ()" of the "pandas" module, one can get the result very conveniently. But as this algorithm needs to traverse all the elements of the input array in sequence, it is difficult to work properly when the number of elements of input sequence is huge. Since the factors required for the calculation - the size of sliding window and the corresponding sequence elements - are already known, we can use Spark to transform the input sequence to an RDD and calculate multiple slices simultaneously on multiple partitions. The specific processing flow is as follows:

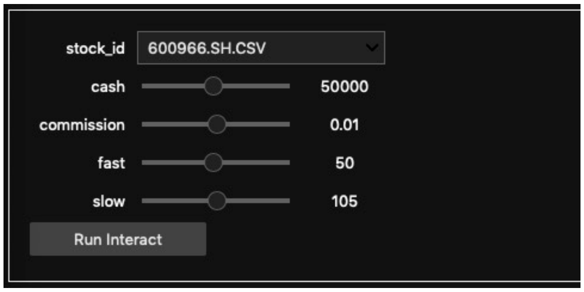
- Since the function input is a column of Spark dataframe, it needs to be transformed into a list first;
- Then construct a reference sequence with the same length as the input sequence, whose elements equals to their indices, and transform it to RDD;
- Regard the elements of the reference sequence as indices, and replace them with the corresponding slices of input sequence;
- Replace each list slice with its average, and finally transform the RDD back to list and return it.

V. Visualization and Interactive Interface

To facilitate the modification of system parameters, we use the "ipywidgets" module to build input widgets and use Jupyter Notebook as the front-end interactive interface. There are five parameters that can be modified in our interactive interface, which are:

- Stock_id — Each id corresponds to the price data of a stock, and also corresponds to a separate input file in .csv format;
- Cash — Represents the amount of initial funds in the virtual account;
- Commission — Represents the rate charged by the exchange for each transaction;
- Fast — Represents the window size of the SMA curve that changes fast in the SMA Cross strategy;
- Slow — Represents the window size of the SMA curve that changes slowly in the SMA Cross strategy.

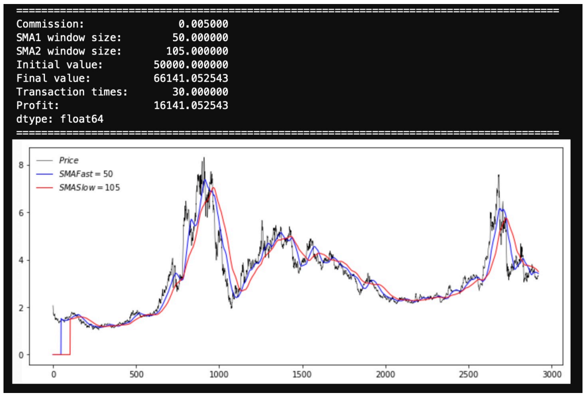
All widgets are called by “ipywidgets.interactive” uniformly, and then displayed together with output data and charts after being packed by “ipywidgets.Box”. Figure 5 is a sample of input widgets in the interface.



stock_id 600966.SH.CSV
cash 50000
commission 0.01
fast 50
slow 105

Fig 5. A sample of input widgets in Jupyter Notebook interface

For the output data, we mainly display it in two parts. The first part is the text output of specific data, which includes: Commission, SMA1 window size, SMA2 window size, Initial value, Final value, Transaction times and Profit. The second part is the chart. We put the original price curve and the SMA1 and SMA2 curves in the same chart such that we can have a very strong sense of the relationship between price trends and SMA, which also helps us analyze the impact of SMAs with different window sizes on the whole strategy. Figure 6 shows sample output data and chart.



```
==============================================================================
Commission:              0.005000
SMA1 window size:       50.000000
SMA2 window size:      105.000000
Initial value:       50000.000000
Final value:         66141.052543
Transaction times:      30.000000
Profit:              16141.052543
dtype: float64
==============================================================================
```

Fig 6. A sample of output data and chart of price, SMA1 and SMA2

VI. Analysis of Data Sets and Strategy Evaluation

Through the output data and charts, we can easily find that the SMA Cross strategy indeed help us obtain benefits from our investment. But when we start to look deep inside, we will find that there is a problem that needs to be solved: How can we obtain the theoretical maximum benefit by modifying input parameters? In this section we are going to explore some input parameters with a sample input data[1] "600966.SH.CSV" which is a ten-year's price record of a stock.

A. Commission

In order to solve this problem, we must first analyze the impact of commission. Since the intersection of two SMA curves can indeed indicate the trend of the price curve, the most direct idea is to increase the number of intersections as much as possible, that is, to increase the number of transactions to obtain higher profits. However, this operation is obviously not feasible due to the existence of commission. During frequent transactions, profits will be continuously lost under the repeated friction of transaction fees. Table 1 shows the changes in profit with and without commission respectively. It can be seen that when there is no commission, the profit is still positive, and when we change the commission value to 8 ‰, which is reasonable in real world, the profit becomes negative, meaning that the investment is loss-making. Thus, we will set the commission to a reasonable value of 5 ‰, then continue to use this commission in later analysis.

Table 1. Profit performance of SMA Cross with and without commission

Instructor	**Without commission**	**With commission**
Commission:	0	0.008
SMA1 window size:	35	35
SMA2 window size:	70	70
Initial value:	50000	50000
Final value:	68049.05701	48563.76421
Transaction frequency:	42	42
Profit:	18049.05701	-1436.235792

B. Difference between the size of SMA windows

According to the principle of the SMA Cross strategy, the window size of SMA1 and SMA2 curves determine the smoothness differently. So we believe that the difference of window sizes between the two SMA curves will definitely affect the final profit. To verify this idea, we fixed the size of fast window and gradually increased the size of slow window to get the relationship between the final profit and the window size difference (marked as d = W (slow)-W (fast)) as shown in Figure 7 (fast window size is fixed at 40, slow window size ranges from 45 to 200).

[1] Data source: http://dataju.cn/Dataju/web/datasetInstanceDetail/37. There are 1096 files in the data set that represent 1096 stocks in Shanghai Stock Market, most of which record their historical prices for more than a decade. All the data is recorded in units of days, and can be selected from the dropdown manual in the interactive interface.

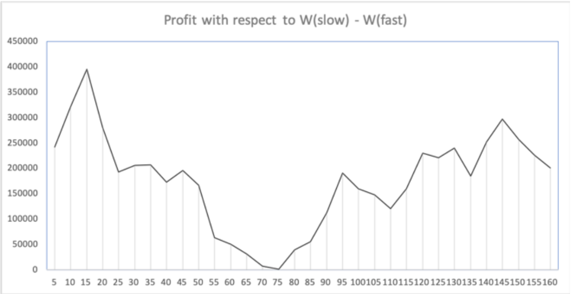


Fig 7. The total profit with respect to the difference between the size of fast and slow window

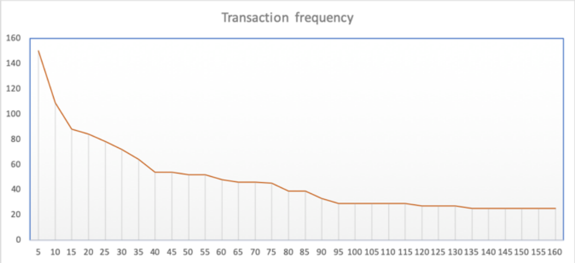


Fig 8. Transaction frequency with respect to the difference between the size of fast and slow window

It can be seen from the figure that the profit reaches a maximum when the difference d = 15 and a minimum when d = 75. The difference between the maximum profit and the minimum profit is more than 400,000, which is very huge compared to the initial value of 50,000. The reason for this phenomenon can be explained by the curve of transaction frequency with respect to d (Fig 8). When d <75, the total profit increases with the frequency of the transaction. However, when the transaction frequency exceeds 88, the income decreases rapidly again. This is because the transaction fees also increase rapidly due to the increase in the number of transactions, which offsets some profits. When d> 75, the income will increase and show a certain randomness. This is because the larger the SMA window value is, the farther the curve deviates from the original curve, which cannot reflect the true trend of the original curve. So at this time, the profit obtained by trading at the intersection of the slow and fast curves is more due to random factors.

C. *The absolute size of SMA windows*

After understanding the impact of the difference in window size on profit, we naturally notice that even if the relative difference between the SMA1 and SMA2 windows is fixed, the final profit will still be affected by the absolute size of the two windows. So, we set the window difference d of SMA1 and SMA2 to 15, change the absolute size of the two windows (range [10, 100] and [25, 115] respectively) and record the corresponding total profit. See Fig 9 for the relationship between profit and fast window size.

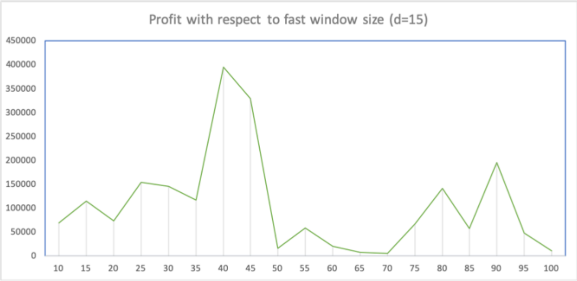


Fig 9. The total profit with respect to fast window size, where the difference between fast and slow windows is d = 15

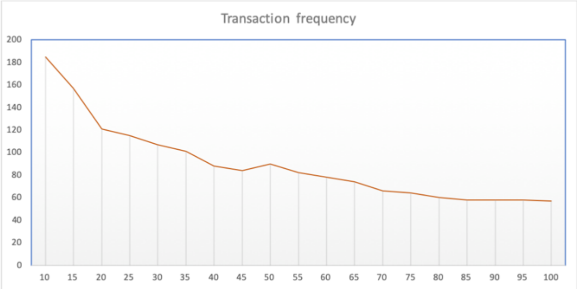


Fig 10. Transaction frequency with respect to fast window size

Here we also analyze it with transaction frequency (Fig 10). It can be seen from Fig 10 that the trading frequency still decreases as the SMA window size increases. When W (fast) = 40, the profit reaches its maximum value. With reference to the previous analysis, it can be known that as the number of transactions decreases, the revenue will decrease accordingly, until W (fast) reaches about 60 (corresponding to W (slow) = 75) and starts to fluctuate randomly. However, when W (fast) < 40, the revenue will decrease to a certain extent due to the increase in transaction fees. For the situation where the window is too large, we can also analyze by comparing the original price curve and the SMA curves. Fig. 11 shows the cases of W (fast) = 30, W (slow) = 45 and W (fast) = 60, W (slow) = 75, respectively. It can be easily found from the figure that as the window size increases, the SMA curves have a significant deviation from the original curve. If the sizes of both windows are relatively large, the intersection of SMA1 and SMA2 will also have a relatively large offset. This causes a delay in the time of buying and selling and makes the overall strategy deviate from the original price trend.

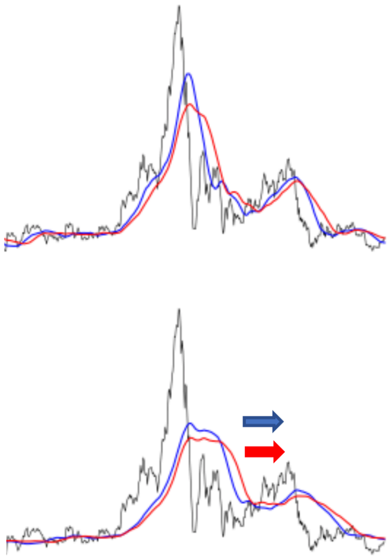


Fig 11. Offset between SMA curves and the original curve

VII. CONCLUSION

In this project, we build a simple quantitative trading back test system, in which we use the SMA Cross strategy to simulate stock trading, and determine the applicable conditions of the strategy based on the total profit obtained. After analyzing some output data, we get the following conclusions:

- The SMA Cross strategy is very sensitive to the size of window size, so one needs to be careful when choosing the fast window and slow window parameters;
- There is a theoretical maximum value of the profit obtained by executing transactions under the SMA Cross strategy;
- The final profit may decrease because of may factors, such as higher transaction fees due to high transaction frequency and problems like intersection-shift. Therefore, the SMA Cross strategy is not suitable for high-frequency trading scenarios.

Appendix: Source Code

utils_spark.py

```python
from numbers import Number
import os.path as path
import findspark
findspark.init("/Users/Evan/spark-3.0.0-preview-bin-hadoop2.7")

from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession

sc = SparkContext(appName="MySpark", master="local", conf=SparkConf().set('spark.ui.port', 4000))
sdf = SparkSession.builder.appName("MySparkDF").master("local").config('spark.ui.port', 5000).getOrCreate()

def assert_msg(condition, message):
    if not condition:
        raise Exception(message)


def read_file(filename):
    # Get the absolute path of file
    file_path = path.join(path.dirname(__file__), filename)

    # Check whether the file exists or not
    assert_msg(path.exists(file_path), 'File does not exist!{0}'.format(file_path))

    # Read CSV file
    df_raw = sdf.read.csv(file_path,sep=',',inferSchema=True)
    df = df_raw.drop('_c8','_c9','_c10','_c11','_c12','_c13','_c14','_c15',
                     '_c16','_c17','_c18','_c19','_c20','_c21','_c22','_c23', '_c24')
    df = df.toDF('ID','Name','Date','Pre_Close','Open','High','Low','Close')
    df = df.withColumn('Date', df.Date.cast('date'))\
            .withColumn('Open', df.Open.cast('double'))\
            .withColumn('High', df.High.cast('double'))\
            .withColumn('Low', df.Low.cast('double'))\
            .withColumn('Close', df.Close.cast('double'))\
            .sort(df.Date)\
            .select('Date','Open','High','Low','Close')\
            .na.drop()
    #df.printSchema()
    #df.show()
    return df


def SMA(col, k):
    """
    Return the Simple Moving Average
    """
    values = col.rdd.map(lambda x: x[0]).collect()
    ref = list(range(len(values)))
    mean_list = sc.parallelize(ref)\
                .map(lambda x: (x, values[x-k+1:x+1] if x >= k-1 else [0]))\
                .map(lambda x: sum(x[1])/k)\
                .collect()
    return mean_list


def crossover(series1, series2) -> bool:
    """
    Check whether two series cross at the end
    :param series1:  series1
    :param series2:  series2
    :return:         True when cross, False otherwise
    """
    return series1[-2] < series2[-2] and series1[-1] > series2[-1]
```

strategy.py

```python
from utils_spark import  assert_msg, crossover, SMA

class SmaCross:
    def __init__(self, broker, data, fast, slow):
        self._broker = broker;
        self._data = data;
        self._tick = 0;
        condition = (fast > 0 and fast <200 and slow > 0 and slow < 500 and fast < slow)
        assert_msg(condition, "Error: Invalid size of rolling windows")
        self._fast = fast
        self._slow = slow
        self.count = 0
        self.sma1 = []
        self.sma2 = []

    def buy(self):
        self._broker.buy()
        self.count += 1

    def sell(self):
        self._broker.sell()
        self.count += 1

    def init(self):
        self.sma1 = SMA(self._data.select(self._data.Close), self._fast)
        self.sma2 = SMA(self._data.select(self._data.Close), self._slow)

    def next(self, tick):
        if crossover(self.sma1[:tick], self.sma2[:tick]):
            self.buy()
        elif crossover(self.sma2[:tick], self.sma1[:tick]):
            self.sell()
        else:
            pass
```

exchange.py

```python
from utils_spark import assert_msg

class Exchange:
    def __init__(self, data, cash, commission):
        assert_msg(cash > 0, 'Initial cash must be greater than 0!')
        assert_msg(0 <= commission <= 0.05,
                   'Commission must be a positive integer which is usually smaller that 0.05')
        self._initial_cash = cash
        self._cash = cash
        self._position = 0
        self._data = data
        self._commission = commission
        self._tick = 0
        self.values = self._data.select(self._data.Close).rdd.map(lambda x: x[0]).collect()

    @property
    def initial_cash(self):
        return self._initial_cash;

    @property
    def current_price(self):
        return self.values[self._tick]

    @property
    def market_value(self):
        return self._cash + self.current_price * self._position

    def buy(self):
        self._position = float((self._cash * (1 - self._commission)) / self.current_price)
        self._cash = 0

    def sell(self):
        self._cash += float(self._position * self.current_price * (1 - self._commission))
        self._position = 0

    def next(self, tick):
        self._tick = tick
```

backtest.py

```python
from strategy import SmaCross
from exchange import Exchange
from utils_spark import assert_msg, crossover, SMA, read_file


class Backtest:
    def __init__(self, data, cash=10000, commission=0.0005, fast=30, slow=90):
        self.sma1 = []
        self.sma2 = []
        self._data = data
        self._cash = cash
        self._commission = commission
        self._fast = fast
        self._slow = slow
        self._results = {}
        self._broker = Exchange(data, cash, commission)
        self._strategy = SmaCross(self._broker, data, fast, slow)
        self.values = self._broker.values

    def run(self):
        self._strategy.init()
        self.sma1 = self._strategy.sma1
        self.sma2 = self._strategy.sma2

        start = self._slow + 1
        end = len(self._broker.values)
        for i in range(start, end):
            self._broker.next(i)
            self._strategy.next(i)

        self._results['Commission:'] = self._commission
        self._results['SMA1 window size:'] = self._fast
        self._results['SMA2 window size:'] = self._slow
        self._results['Initial value:'] = self._broker.initial_cash
        self._results['Final value:'] = self._broker.market_value
        self._results['Transaction frequency:'] = self._strategy.count
        self._results['Profit:'] = self._broker.market_value - self._broker.initial_cash

        return self._results

    def print_results(self):
        for k, v in self._results.items():
            print('  ', k, v)

def main():
    data = read_file('data/600602.SH.csv')
    backtest = Backtest(data, 10000.0, 0, 30, 90)
    profit = backtest.run()
    backtest.print_results()


if __name__ == '__main__':
    main()
```

interface.ipynb

```
import matplotlib.pyplot as plt
import pandas as pd
from backtest import Backtest
from ipywidgets import *
from utils_spark import read_file
import ipywidgets as widgets
import os


files = []
for path in os.walk('data'):
    files = path[2]

def para(stock_id, cash, commission, fast, slow):
    df = read_file('data/{0}'.format(stock_id))
    backtest = Backtest(df, cash = cash, commission = commission, fast=fast, slow=slow)
    ret = backtest.run()
    chart0 = pd.DataFrame(backtest.values)
    chart1 = pd.DataFrame(backtest.sma1)
    chart2 = pd.DataFrame(backtest.sma2)
    figsize = 12, 5
    plt.subplots(figsize = figsize)
    ax1 = plt.subplot()
    plt.plot(chart0, color='black', label='$Price$', linewidth=0.5)
    plt.legend(loc='upper left', frameon=False)
    plt.plot(chart1, color='blue', label='$SMA Fast = {0}$'.format(fast), linewidth=1)
    plt.legend(loc='upper left', frameon=False)
    plt.plot(chart2, color='red', label='$SMA Slow = {0}$'.format(slow), linewidth=1)
    plt.legend(loc='upper left', frameon=False)
    print('Stock ID: {0}'.format(stock_id[:6]))
    print('=======================================================================================')
    print(pd.Series(ret))
    print('=======================================================================================')

form_item_layout = Layout(
    justify_content='space-around',
    margin='2%'
)
form_items = [
    VBox(children=interactive(
        para,
        {'manual': True},
        stock_id = files,
        cash = (5000, 100000, 5000),
        commission = (0, 0.01, 0.001),
        fast=(5, 100, 5),
        slow=(10, 200, 5)
    ).children, layout=form_item_layout)
]
form = Box(form_items, layout=Layout(
    display='flex',
    flex_flow='column',
    border='solid 1px',
    align_items='stretch',
    width='750px',
    margin='1.3%'
))

form
```