In [4]:

```
from jqdatasdk import *
import akshare as ak
import baostock as bs
import pandas as pd
import re
import datetime
import time
```

In [5]:

```
from jqdatasdk import bond
auth("13320010236", "991204Ctj")

print(f"Remaining daily data queries allowed on JoinQuant: {get_query_count()}")
```

```
auth success
Remaining daily data queries allowed on JoinQuant: {'total': 1000000, 'spare': 99463
5}
```

In [ ]:

```
# Log in baostock system
lg = bs.login()

# Loggin information
print("login respond error_code:" + lg.error_code)
print("login respond error_msg:" + lg.error_msg)
```

# 1 Crawling interest rates

## 1.1 Setting model parameters

In [14]:

```
# Initial time settings
startDate, endDate = "19950101", "20211231"  # Time interval

# Check whether time interval is valid.
if int(startDate) > int(endDate) or int(endDate) > datetime.date.today().year * 10000 + \
        datetime.date.today().month * 100 + datetime.date.today().day:
    print("Invalid Time Interval")
    quit()
```

In [1]:

```
# Other initial paraters
tradePercent = 0.1  # Long & Short Proportion
laggedPeriod = pd.Timedelta("30 D")  # Lagged Period
windowPeriod = pd.Timedelta(str(30 * 11) + " D")  # Window
holdPeriod = pd.Timedelta("30 D")  # Holding Period
```

```
NameError: name 'pd' is not defined
```

## 1.2 Setting interest-free rates

| Time(t) | Sources |
|---|---|
| t $\leq$ 2002-08-06 | Three-month fixed deposit rates |
| 2002-08-07 $\leq$ t $\leq$ 2006-10-07 | Coupon rate of three-month central bank bills |
| 2006-10-08 $\leq$ t | Shibor |

In [3]:

```
# Transform data type.
def interest_dateformat(date: str) -> str:
    return f"{date[:4]}-{date[4:6]}-{date[6:]}"
```

### 1.2.1 Three-month fixed deposit rates

In [11]:

```python
rs = bs.query_deposit_rate_data(start_date = interest_dateformat(str(int(startDate) - 50000)),
                                end_date = interest_dateformat(endDate))
print("query_deposit_rate_data respond error_code:" + rs.error_code)
print("query_deposit_rate_data respond error_msg:" + rs.error_msg)

# deposit_rate result data set
interest_data_list = []
while (rs.error_code == "0") & rs.next():
    # merge every single data
    interest_data_list.append(rs.get_row_data())
result = pd.DataFrame(interest_data_list, columns=rs.fields)

# save to csv
result.to_csv("C:\\Users\\tianj\\Project 1\\data\\HS300_data\\deposit_interest_rate.csv",
              encoding = "gbk", index = False)
print(result)
```

```
query_deposit_rate_data respond error_code:0
query_deposit_rate_data respond  error_msg:success
       pubDate demandDepositRate fixedDepositRate3Month  \
0   1995-01-01
1   1995-07-01
2   1996-05-01          2.970000               4.860000
3   1996-08-23          1.980000               3.330000
4   1997-10-23          1.710000               2.880000
5   1998-03-25          1.710000               2.880000
6   1998-07-01          1.440000               2.790000
7   1998-12-07          1.440000               2.790000
8   1999-06-10          0.990000               1.980000
9   2002-02-21          0.720000               1.710000
10  2004-10-29          0.720000               1.710000
11  2006-04-28
12  2006-08-19          0.720000               1.800000
13  2007-03-18          0.720000               1.980000
14  2007-05-19          0.720000               2.070000
15  2007-07-21          0.810000               2.340000
16  2007-08-22          0.810000               2.610000
17  2007-09-15          0.810000               2.880000
18  2007-12-21          0.720000               3.330000
19  2008-09-16
20  2008-10-09          0.720000               3.150000
21  2008-10-15
22  2008-10-30          0.720000               2.880000
23  2008-11-27          0.360000               1.980000
24  2008-12-23          0.360000               1.710000
25  2010-10-20          0.360000               1.910000
26  2010-12-26          0.360000               2.250000
27  2011-02-09          0.400000               2.600000
28  2011-04-06          0.500000               2.850000
29  2011-07-07          0.500000               3.100000
30  2012-06-08          0.400000               2.850000
31  2012-07-06          0.350000               2.600000
32  2014-11-22          0.350000               2.350000
33  2015-03-01          0.350000               2.100000
34  2015-05-11          0.350000               1.850000
35  2015-06-28          0.350000               1.600000
36  2015-08-26          0.350000               1.350000
37  2015-10-24          0.350000               1.100000
```

```
   fixedDepositRate6Month fixedDepositRate1Year fixedDepositRate2Year  \
0
1
2                7.200000              9.180000              9.900000
3                5.400000              7.470000              7.920000
4                4.140000              5.670000              5.940000
5                4.140000              5.220000              5.580000
6                3.960000              4.770000              4.860000
7                3.330000              3.780000              3.960000
8                2.160000              2.250000              2.430000
9                1.890000              1.980000              2.250000
10               2.070000              2.250000              2.700000
11
12               2.250000              2.520000              3.060000
13               2.430000              2.790000              3.330000
14               2.610000              3.060000              3.690000
15               2.880000              3.330000              3.960000
16               3.150000              3.600000              4.230000
17               3.420000              3.870000              4.500000
18               3.780000              4.140000              4.680000
19
20               3.510000              3.870000              4.410000
21
22               3.240000              3.600000              4.140000
23               2.250000              2.520000              3.060000
24               1.980000              2.250000              2.790000
25               2.200000              2.500000              3.250000
26               2.500000              2.750000              3.550000
27               2.800000              3.000000              3.900000
28               3.050000              3.250000              4.150000
29               3.300000              3.500000              4.400000
30               3.050000              3.250000              4.100000
31               2.800000              3.000000              3.750000
32               2.550000              2.750000              3.350000
33               2.300000              2.500000              3.100000
34               2.050000              2.250000              2.850000
35               1.800000              2.000000              2.600000
36               1.550000              1.750000              2.350000
37               1.300000              1.500000              2.100000

   fixedDepositRate3Year fixedDepositRate5Year  \
0
1
2              10.800000             12.060000
3               8.280000              9.000000
4               6.210000              6.660000
5               6.210000              6.660000
6               4.950000              5.220000
7               4.140000              4.500000
8               2.700000              2.880000
9               2.520000              2.790000
10              3.240000              3.600000
11
12              3.690000              4.140000
13              3.960000              4.410000
14              4.410000              4.950000
15              4.680000              5.220000
16              4.950000              5.490000
17              5.220000              5.760000
18              5.400000              5.850000
```

```
19
20            5.130000            5.580000
21
22            4.770000            5.130000
23            3.600000            3.870000
24            3.330000            3.600000
25            3.850000            4.200000
26            4.150000            4.550000
27            4.500000            5.000000
28            4.750000            5.250000
29            5.000000            5.500000
30            4.650000            5.100000
31            4.250000            4.750000
32            4.000000
33            3.750000
34            3.500000
35            3.250000
36            3.000000
37            2.750000

   installmentFixedDepositRate1Year installmentFixedDepositRate3Year  \
0
1
2                          7.200000                         9.180000
3                          5.400000                         7.470000
4                          4.140000                         5.670000
5                          4.140000                         5.220000
6                          3.960000                         4.770000
7                          3.330000                         3.780000
8                          1.980000                         2.160000
9                          1.710000                         1.890000
10                         1.710000                         2.070000
11
12                         1.800000                         2.250000
13                         1.980000                         2.430000
14                         2.070000                         2.610000
15                         2.340000                         2.880000
16                         2.610000                         3.150000
17                         2.880000                         3.420000
18                         3.330000                         3.780000
19
20                         3.150000                         3.510000
21                         3.150000                         3.510000
22                         2.880000                         3.240000
23                         1.980000                         2.250000
24                         1.710000                         1.980000
25                         1.910000                         2.200000
26                         2.250000                         2.500000
27                         2.600000                         2.800000
28                         2.850000                         3.050000
29                         3.100000                         3.300000
30                         2.850000                         3.050000
31                         2.600000                         2.800000
32                         2.350000                         2.550000
33                         2.100000                         2.300000
34                         1.850000                         2.050000
35                         1.600000                         1.800000
36                         1.350000                         1.550000
37                         1.100000                         1.300000

   installmentFixedDepositRate5Year
```

```
0
1
2                         10.800000
3                          8.280000
4                          6.210000
5                          6.210000
6                          4.950000
7                          4.140000
8                          2.250000
9                          1.980000
10                         2.250000
11
12                         2.520000
13                         2.790000
14                         3.060000
15                         3.330000
16                         3.600000
17                         3.870000
18                         4.140000
19
20                         3.870000
21                         3.870000
22                         3.600000
23                         2.520000
24                         2.250000
25                         2.500000
26                         2.750000
27                         3.000000
28                         3.250000
29                         3.500000
30                         3.250000
31                         3.000000
32
33
34
35
36
37
```

### 1.2.2 Coupon rate of three-month central bank bills

In [ ]:

```
central_bank_bill = bond.run_query(query(bond.BOND_BASIC_INFO).filter(
    bond.BOND_BASIC_INFO.bond_type_id == "703019").limit(5000))
central_bank_bill["maturity"] = central_bank_bill["maturity_date"] - central_bank_bill["interes

# Filter out coupons with three-month periodicity.
central_bank_bill = central_bank_bill[("80d" < central_bank_bill["maturity"]) &
                                      (central_bank_bill["maturity"] < "100d")]

# save to csv file
central_bank_bill.to_csv("C:\\Users\\tianj\\Project 1\\data\\HS300_data\\central_bank_bill.csv"
                         encoding="gbk", index=False)
```

### 1.2.3 Shibor

In [13]:

```
rs = bs.query_shibor_data(start_date = interest_dateformat(startDate),
                          end_date = interest_dateformat(endDate))
print("query_shibor_data respond error_code:" + rs.error_code)
print("query_shibor_data respond  error_msg:" + rs.error_msg)

# shibor result list
interest_data_list = []
while (rs.error_code == "0") & rs.next():
    # mearge each single data
    interest_data_list.append(rs.get_row_data())
result = pd.DataFrame(interest_data_list, columns = rs.fields)

# save to csv file
result.to_csv("C:\\Users\\tianj\\Project 1\\data\\HS300_data\\shibor.csv",
              encoding = "gbk", index = False)
print(result)
```

```
query_shibor_data respond error_code:0
query_shibor_data respond  error_msg:success
            date  shiborON  shibor1W  shibor2W  shibor1M  shibor3M  shibor6M  \
0     2006-10-08  2.118400  2.293000  2.384800  2.531900  2.611000  2.740400
1     2006-10-09  2.099000  2.296000  2.397200  2.552200  2.624800  2.743100
2     2006-10-10  2.092200  2.297100  2.423600  2.573900  2.632500  2.745400
3     2006-10-11  2.095500  2.293200  2.493000  2.586400  2.633800  2.747500
4     2006-10-12  2.094300  2.290400  2.524000  2.590000  2.638000  2.747000
...          ...       ...       ...       ...       ...       ...       ...
3759  2021-10-25  1.610000  2.243000  2.410000  2.380000  2.438000  2.522000
3760  2021-10-26  1.546000  2.262000  2.424000  2.387000  2.443000  2.525000
3761  2021-10-27  1.921000  2.268000  2.441000  2.392000  2.445000  2.529000
3762  2021-10-28  1.940000  2.301000  2.441000  2.396000  2.448000  2.533000
3763  2021-10-29  2.142000  2.299000  2.414000  2.398000  2.449000  2.536000

      shibor9M  shibor1Y
0     2.852100  2.954300
1     2.851400  2.954900
2     2.854400  2.953100
3     2.854700  2.955900
4     2.857000  2.955000
...        ...       ...
3759  2.661000  2.759000
3760  2.664000  2.766000
3761  2.664000  2.773000
3762  2.666000  2.778000
3763  2.669000  2.782000

[3764 rows x 9 columns]
```

## 2 Data pre-processing of HS300 stocks

In [14]:

```
rs = bs.query_hs300_stocks()
print("query_hs300 error_code:" + rs.error_code)
print("query_hs300 error_msg:" + rs.error_msg)

# HS300 result list
hs300_stocks = []
while (rs.error_code == "0") & rs.next():
    # merge every single data
    hs300_stocks.append(rs.get_row_data())
result = pd.DataFrame(hs300_stocks, columns=rs.fields)

# save to csv
result.to_csv("C:\\Users\\tianj\\Project 1\\data\\hs300_stocks.csv",
              encoding="gbk", index=False)
print(result)
```

```
query_hs300 error_code:0
query_hs300  error_msg:success
     updateDate       code code_name
0    2022-01-17  sh.600000      浦发银行
1    2022-01-17  sh.600009      上海机场
2    2022-01-17  sh.600010      包钢股份
3    2022-01-17  sh.600011      华能国际
4    2022-01-17  sh.600015      华夏银行
..          ...        ...       ...
295  2022-01-17  sz.300782       卓胜微
296  2022-01-17  sz.300866      安克创新
297  2022-01-17  sz.300888      稳健医疗
298  2022-01-17  sz.300896       爱美客
299  2022-01-17  sz.300999       金龙鱼

[300 rows x 3 columns]
```

In [ ]:

```
# log out
bs.logout()
```

In [20]:

```
def GetCodeLst(fromWhat: str) -> list:
    if fromWhat == "HS300":
        hs300_Stocks = pd.read_csv("C:\\Users\\tianj\\Project 1\\data\\HS300_data\\hs300_stocks
                           encoding="gbk").set_index("code")
        return list(map(lambda x: re.search(pattern="[0-9]+", string=x).group(),
                        list(hs300_Stocks.index)))

    elif fromWhat == "A":
        return list(ak.stock_info_sh_name_code(indicator="主板A股")["代码"]) + \
               list(ak.stock_info_sh_name_code(indicator="科创板")["代码"]) + \
               list(ak.stock_info_sz_name_code(indicator="A股列表")["A股代码"])
```

In [23]:

```
codeLst = GetCodeLst(fromWhat="HS300")
```

Out[23]:

```
300
```

In [16]:

```
# get the trading calendar
def GetTradeCalender(start: str, end: str) -> pd.Series:
    cal = ak.tool_trade_date_hist_sina()
    return cal["trade_date"][
        (datetime.date(int(start[:4]),int(start[4:6]), int(start[6:])) <= cal["trade_date"]) &
        (cal["trade_date"] <= datetime.date(int(end[:4]), int(end[4:6]), int(end[6:])))]

calender = GetTradeCalender(startDate, endDate)

# save the trading calendar
calender.to_csv("C:\\Users\\tianj\\Project 1\\data\\HS300_data\\calender.csv", index=True, hea
```

In [18]:

```
# Construct dataframe for data of closed prices, returns, book-to-market ratios and market valu
close_df, return_df, BM_df, MV_df = pd.DataFrame(index=calender), pd.DataFrame(index=calender),
                                    pd.DataFrame(index=calender), pd.DataFrame(index=calender)
```

Out[18]:

| trade_date |
|---|
| **1995-01-03** |
| **1995-01-04** |
| **1995-01-05** |
| **1995-01-06** |
| **1995-01-09** |
| **...** |
| **2021-12-27** |
| **2021-12-28** |
| **2021-12-29** |
| **2021-12-30** |
| **2021-12-31** |

6558 rows × 0 columns

## 3 Save the information of each single stock

In [ ]:

```python
# transform the type to datetime.date
def ParseDate(date: str) -> datetime.date:
    date = list(map(int, re.findall(pattern="[0-9]+", string=str(date))))
    return datetime.date(date[0], date[1], date[2])
```

In [ ]:

```python
# what_now means the postion in our while loop, exceptionLst contains the stock codes which can
# traded on that given trade date, failure counts the time that we failed to crawl for the data
# any single stock, maximum_failure_allowed represents the maximum time that we allowed for con
# failures.
what_now, exceptionLst, failure, maximum_failure_allowed, length = 0, [], 0, 3, len(codeLst)
print(f"Data of {length} stocks in total need to be collected, waiting......")
while what_now < length:
    code = codeLst[what_now]

    try:
        # Crawl for closed prices and daily returns from 1995-01-01 to 2021-12-31.
        this_stock_hist_daily = ak.stock_zh_a_hist(symbol=code, period="daily",
                                start_date=startDate, end_date=endDate,
                                adjust="hfq")[["日期", "收盘", "涨跌幅"]].set_index("日期")

        # Crawl for BMMVs and market values from 1995-01-01 to 2021-12-31.
        this_stock_BMMV_daily = \
            ak.stock_a_lg_indicator(symbol=code)[["trade_date", "pb", "total_mv"
                                                  ]].set_index("trade_date")

        # Transform data type.
        this_stock_hist_daily.index = map(ParseDate, list(this_stock_hist_daily.index))
        this_stock_BMMV_daily.index = map(ParseDate, list(this_stock_BMMV_daily.index))

        failure = 0

        # Merge the data
        try:
            close_df[code] = this_stock_hist_daily["收盘"]
            return_df[code] = this_stock_hist_daily["涨跌幅"]
            BM_df[code] = 1 / this_stock_BMMV_daily["pb"]
            MV_df[code] = this_stock_BMMV_daily["total_mv"]
            print(f"{what_now}/{length}. Data collected and merged for code: {code}")

        except:
            # If we met failures when merging the data
            print(f"{what_now}/{length}. Met an unknown error when merging data of code: {code}
            exceptionLst.append((what_now, code))

        # Write into csv file after we crawling for 100 sets of data
        if what_now % 100 == 0:
            close_df.to_csv("C:\\Users\\tianj\\Project 1\\data\\HS300_temp_data\\close_temp.csv
                            index=True, header=True)
            return_df.to_csv("C:\\Users\\tianj\\Project 1\\data\\HS300_temp_data\\return_temp.c
                             index=True, header=True)
            BM_df.to_csv("C:\\Users\\tianj\\Project 1\\data\\HS300_temp_data\\BM_temp.csv",
                         index=True, header=True)
            MV_df.to_csv("C:\\Users\\tianj\\Project 1\\data\\HS300_temp_data\\MV_temp.csv",
                         index=True, header=True)

            pd.Series(exceptionLst).to_csv(
                "C:\\Users\\tianj\\Project 1\\data\\HS300_temp_data\\exceptionLst.csv",
                index=False, header=True)

            # Whether we saved the temp data successfully.
            print(f"Temporary file is saved at: {code}. Position is: {what_now}")

        # Sleep for 45 seconds for every 30 sets of data (each including 4375 * 4 lines of data
        if what_now % 30 == 0:
            print("Resuming in 45 seconds......")
            time.sleep(45)

        what_now += 1

    except:
        # Print the break point and return the total number of failed requests.
        failure += 1
        print(f"{what_now}/{length}. Problem encountered at code: {code}. Failure = {failure}")

        if failure > maximum_failure_allowed:
            break  # Quit the program if we receive too many failures.
        else:
            print(f"Retrying in {60 * failure} seconds......")
            time.sleep(60 * failure)  # Sleep for 1-3 minutes.
            continue


# Write into csv file.
else:
    close_df.to_csv("C:\\Users\\tianj\\Project 1\\data\\HS300_data\\close.csv", index=True, hea
    return_df.to_csv("C:\\Users\\tianj\\Project 1\\data\\HS300_data\\return.csv", index=True, 
    BM_df.to_csv("C:\\Users\\tianj\\Project 1\\data\\HS300_data\\BM.csv", index=True, header=T
    MV_df.to_csv("C:\\Users\\tianj\\Project 1\\data\\HS300_data\\MV.csv", index=True, header=T

    if len(exceptionLst) == 0:
        print("All data are collected and merged successfully")
    else:
        print("exceptionLst is not empty: failed to merge some data")
```