MFE 409: Financial Risk Management, Problem set 3

Huanyu Liu, Yong Jia Tan, Tongsu Peng, Sejal Bharati

4/22/2019

Starting from next page

hw3

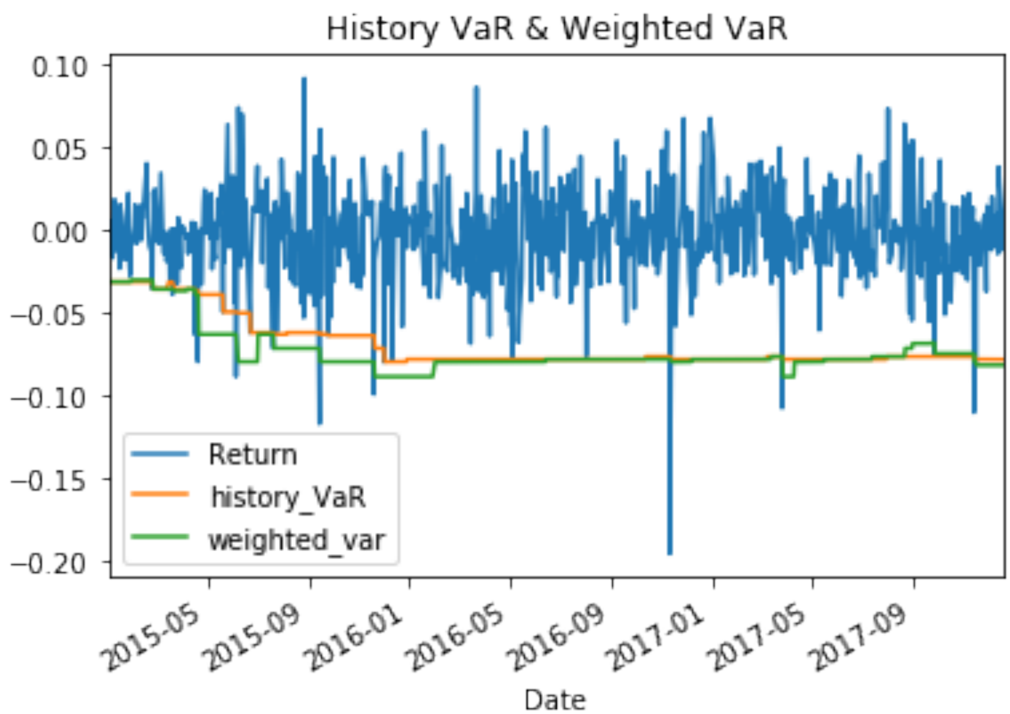
April 22, 2019

1 1

1.

```
In [1]: %matplotlib inline
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import math
        from scipy.stats import norm
        from scipy.stats import chi2
        import seaborn as sns
        c = 0.99
        data = pd.read_csv('/Users/huanyu/Desktop/RiskManagement/hw3/hw3_returns2.csv')
        data['Date'] = pd.to_datetime(data['Date'])
        data.loc[252:,'history_VaR'] = data.index[252:].map(lambda x:
                np.sort(data.loc[:x-1,'Return'])[math.ceil(x * 0.01) - 1])
        lamb = 0.995
        length = len(data)
        for i in range(252,length):
            weight = [lamb ** (i - j) * (1 - lamb) / (1 - lamb ** i) for j in range(1,i+1)]
            sorted_data = pd.DataFrame({'weight':weight,
                'return':data.loc[:i-1,'Return']}).sort_values('return')
            sorted_data.reset_index(drop=True,inplace=True)
            cum_weight = 0
            counter = 0
            while cum_weight < 0.01:
                cum_weight += sorted_data.loc[counter,'weight']
                counter += 1
            data.loc[i,'weighted_var'] = sorted_data.loc[counter - 1, 'return']
        data.loc[252:,('Date','Return','history_VaR','weighted_var')].plot(x='Date')
        plt.title('History VaR & Weighted VaR')
        plt.show()
        hist_exception = (data['Return'] < data['history_VaR'].shift(1)).value_counts()[True]
        weighted_exception = (data['Return'] <
            data['weighted_var'].shift(1)).value_counts()[True]
```

```python
def chi_test(m):
    days = len(data.loc[252:,:])
    chi_critical = chi2.ppf(0.95, 1)
    test_value = -2 * math.log(c ** (days - m) * (1 - c) ** m) + 2 * math.log((1 - m /
    test_result = pd.Series({'Test Value': test_value, 'Chi-square Critical': chi_crit
    print(test_result)

print(hist_exception)
print(weighted_exception)
chi_test(hist_exception)
chi_test(weighted_exception)
```




```
18
13
Test Value              10.723165
Chi-square Critical      3.841459
dtype: float64
Test Value              3.371881
Chi-square Critical     3.841459
dtype: float64
```

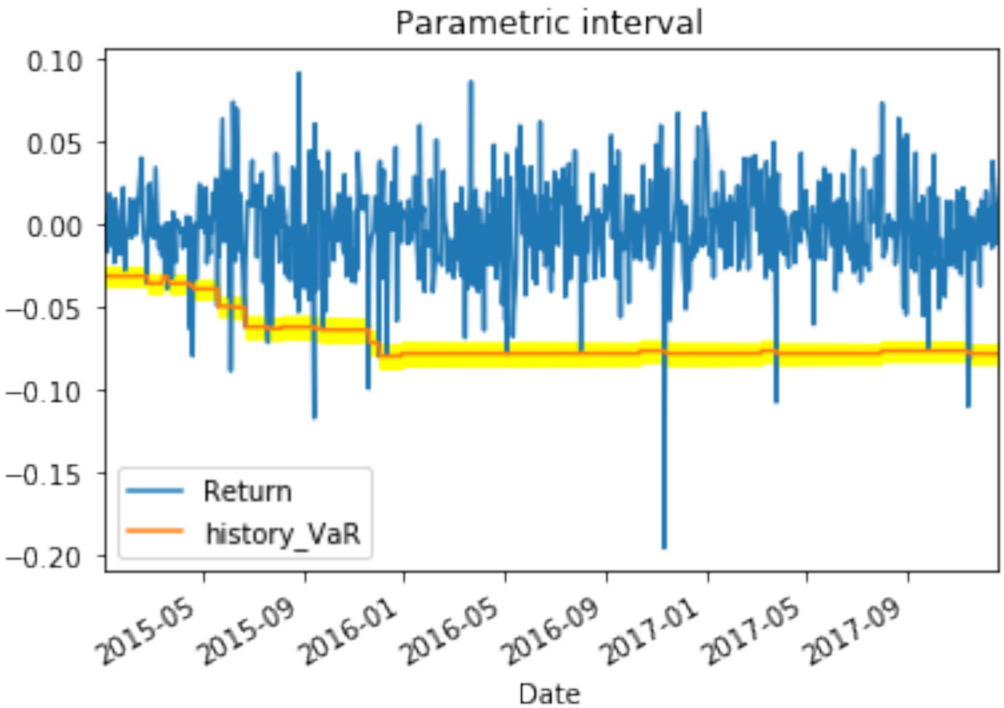
The NO. of historical VaR exception is 18.
The NO. of weighted VaR exception is 13.

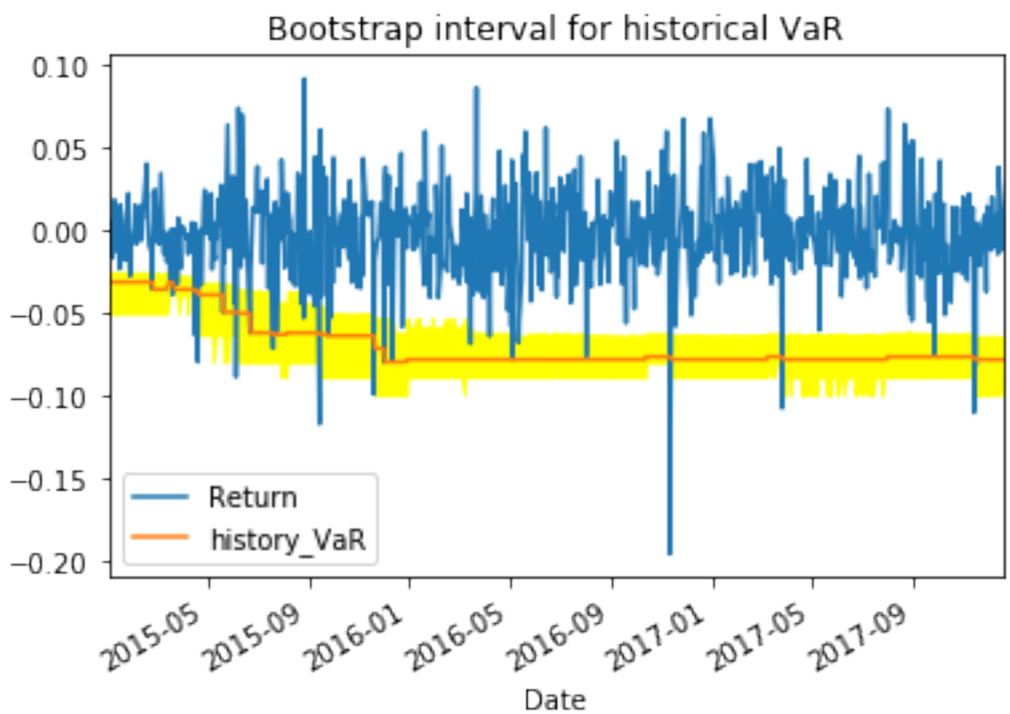
According to the result of chi-square test, the NO. of historical VaR exception is significant, while the NO. of weighted VaR exception can be accepted.

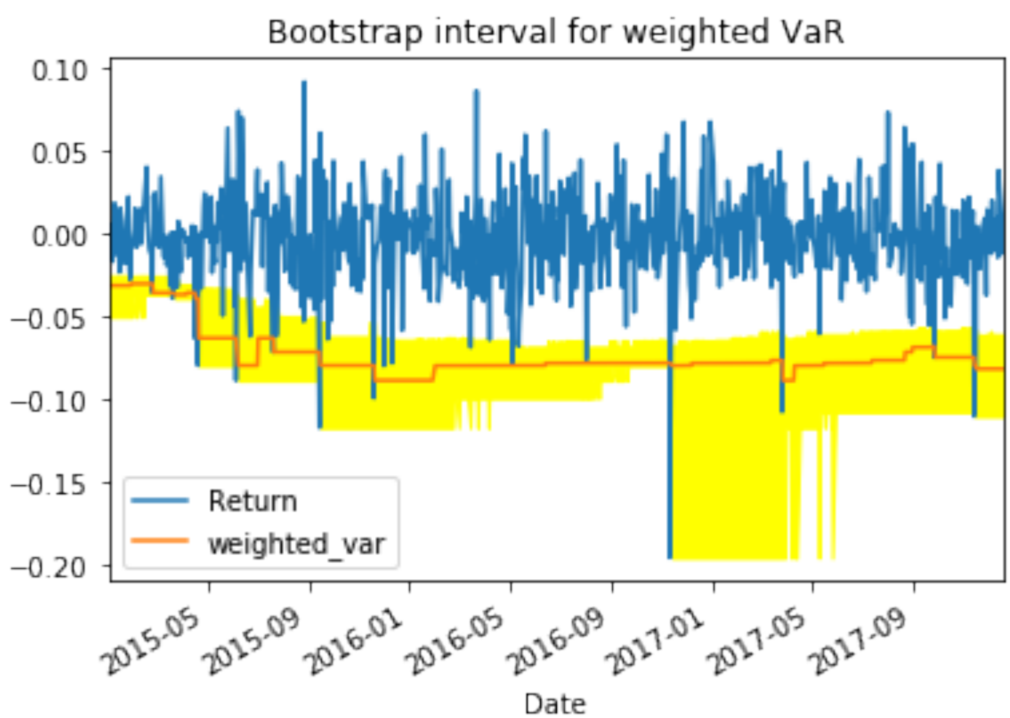
2.

```
In [2]: data.loc[252:,'mean'] = data.index[252:].map(lambda x: data.loc[:x-1,'Return'].mean())
        cdf_inverse = norm.ppf(0.01)
        cdf = norm.ppf(0.975)
        data.loc[252:,'std'] = data.index[252:].map(lambda x: data.loc[:x-1,'Return'].std())
        data['f_x'] = 1 / (data['std'] * math.sqrt(2 * math.pi)) * \
            np.exp(-(data['std'] * cdf_inverse)**2 / (2 * data['std']**2))
        data.loc[252:,'cinv_upper'] = data.index[252:].map(lambda x: 1 / data.loc[x,'f_x']
            * math.sqrt(0.99 * 0.01 / x) * cdf + data.loc[x,'history_VaR'])
        data.loc[252:,'cinv_lower'] = data.index[252:].map(lambda x: 1 / data.loc[x,'f_x']
            * math.sqrt(0.99 * 0.01 / x) * -cdf + data.loc[x,'history_VaR'])

        data.loc[252:,('Date','Return','history_VaR')].plot(x='Date')
        plt.fill_between(data.loc[252:,'Date'],data.loc[252:,'cinv_lower'],
            data.loc[252:,'cinv_upper'],color='yellow')
        plt.title('Parametric interval')
        plt.show()

        var = np.zeros(1000)
        for i in range(252,length):
            for j in range(1000):
                returns = np.random.choice(data.loc[:i-1,'Return'],size=i,replace=True)
                var[j] = np.sort(returns)[math.ceil(i * 0.01) - 1]
            data.loc[i,'bs_hist_upper'] = np.sort(var)[974]
            data.loc[i,'bs_hist_lower'] = np.sort(var)[24]

        data.loc[252:,('Date','Return','history_VaR')].plot(x='Date')
        #data['history_VaR'].plot()
        plt.fill_between(data.loc[252:,'Date'],data.loc[252:,'bs_hist_lower'],data.loc[252:,'bs
                color='yellow')
        plt.title('Bootstrap interval for historical VaR')
        plt.show()


        for i in range(252, length):
            weight = [lamb ** (i - j) * (1 - lamb) / (1 - lamb ** i) for j in range(1, i + 1)]
            for j in range(1000):
                indexes = np.sort(np.random.choice(data.index[:i],replace=True,size=i))
                sorted_data = pd.DataFrame({'weight':weight,
                        'return':data.loc[indexes,'Return']}).sort_values('return')
                sorted_data.reset_index(drop=True, inplace=True)
                cum_weight = 0
                counter = 0
                while cum_weight < 0.01:
```

```
            cum_weight += sorted_data.loc[counter, 'weight']
            counter += 1
        var[j] = sorted_data.loc[counter - 1, 'return']
    data.loc[i, 'bs_wt_upper'] = np.sort(var)[974]
    data.loc[i, 'bs_wt_lower'] = np.sort(var)[24]

data.loc[252:,('Date','Return','weighted_var')].plot(x='Date')
plt.fill_between(data.loc[252:,'Date'],data.loc[252:,'bs_wt_lower'],data.loc[252:,'bs_w
    color='yellow')
plt.title('Bootstrap interval for weighted VaR')
plt.show()

print(data[['cinv_upper','cinv_lower','bs_hist_upper','bs_hist_lower',
            'bs_wt_upper','bs_wt_lower']])
```




Bootstrap interval for historical VaR
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
Return
history_VaR
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
Date


Bootstrap interval for weighted VaR
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
Return
weighted_var
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
Date

	cinv_upper	cinv_lower	bs_hist_upper	bs_hist_lower	bs_wt_upper	\
0	NaN	NaN	NaN	NaN	NaN	
1	NaN	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	
5	NaN	NaN	NaN	NaN	NaN	
6	NaN	NaN	NaN	NaN	NaN	
7	NaN	NaN	NaN	NaN	NaN	
8	NaN	NaN	NaN	NaN	NaN	
9	NaN	NaN	NaN	NaN	NaN	
10	NaN	NaN	NaN	NaN	NaN	
11	NaN	NaN	NaN	NaN	NaN	
12	NaN	NaN	NaN	NaN	NaN	
13	NaN	NaN	NaN	NaN	NaN	
14	NaN	NaN	NaN	NaN	NaN	
15	NaN	NaN	NaN	NaN	NaN	
16	NaN	NaN	NaN	NaN	NaN	
17	NaN	NaN	NaN	NaN	NaN	
18	NaN	NaN	NaN	NaN	NaN	
19	NaN	NaN	NaN	NaN	NaN	
20	NaN	NaN	NaN	NaN	NaN	
21	NaN	NaN	NaN	NaN	NaN	
22	NaN	NaN	NaN	NaN	NaN	
23	NaN	NaN	NaN	NaN	NaN	
24	NaN	NaN	NaN	NaN	NaN	
25	NaN	NaN	NaN	NaN	NaN	
26	NaN	NaN	NaN	NaN	NaN	
27	NaN	NaN	NaN	NaN	NaN	
28	NaN	NaN	NaN	NaN	NaN	
29	NaN	NaN	NaN	NaN	NaN	
..	...	...	...	...	...	
970	-0.071020	-0.082800	-0.063458	-0.089049	-0.058412	
971	-0.071026	-0.082794	-0.062454	-0.089049	-0.058412	
972	-0.071029	-0.082791	-0.062454	-0.089049	-0.056579	
973	-0.071035	-0.082785	-0.063458	-0.080186	-0.056579	
974	-0.071040	-0.082780	-0.062454	-0.089049	-0.056579	
975	-0.071045	-0.082775	-0.063458	-0.089049	-0.056579	
976	-0.072632	-0.084463	-0.064208	-0.099686	-0.058763	
977	-0.072638	-0.084457	-0.064208	-0.099686	-0.058412	
978	-0.072746	-0.084613	-0.064208	-0.099686	-0.063458	
979	-0.072752	-0.084608	-0.064570	-0.099686	-0.062208	
980	-0.072758	-0.084601	-0.064208	-0.099686	-0.060922	
981	-0.072762	-0.084597	-0.064208	-0.099686	-0.060922	
982	-0.072767	-0.084592	-0.064208	-0.099686	-0.062208	
983	-0.072773	-0.084586	-0.064208	-0.099686	-0.060922	
984	-0.072779	-0.084581	-0.064208	-0.099686	-0.062208	
985	-0.072783	-0.084576	-0.064208	-0.099686	-0.060922	

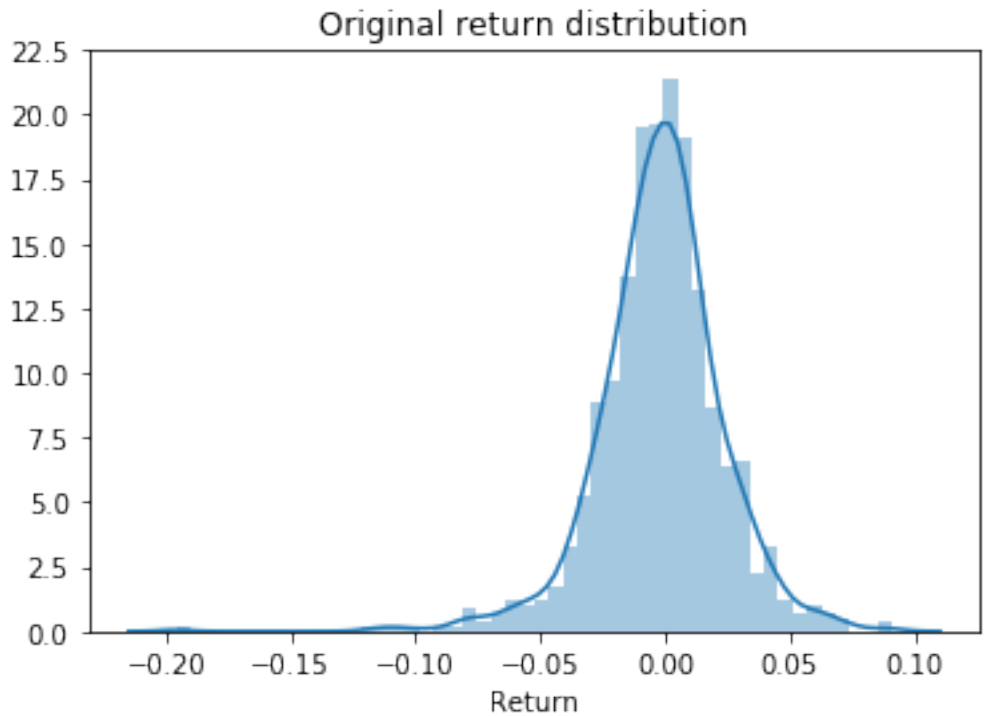
986	-0.072783	-0.084576	-0.064570	-0.099686	-0.062208
987	-0.072788	-0.084571	-0.064570	-0.099686	-0.060922
988	-0.072794	-0.084565	-0.064208	-0.089049	-0.058911
989	-0.072800	-0.084560	-0.064570	-0.099686	-0.060922
990	-0.072805	-0.084554	-0.064208	-0.089049	-0.060922
991	-0.072809	-0.084551	-0.064208	-0.099686	-0.060922
992	-0.072815	-0.084545	-0.064208	-0.089049	-0.060922
993	-0.072821	-0.084539	-0.064208	-0.099686	-0.060922
994	-0.072826	-0.084533	-0.064570	-0.089049	-0.060922
995	-0.072832	-0.084528	-0.064570	-0.099686	-0.060922
996	-0.072837	-0.084523	-0.064208	-0.099686	-0.060922
997	-0.072835	-0.084524	-0.064570	-0.089049	-0.060922
998	-0.072837	-0.084523	-0.064570	-0.099686	-0.060922
999	-0.072842	-0.084518	-0.064208	-0.089049	-0.060922

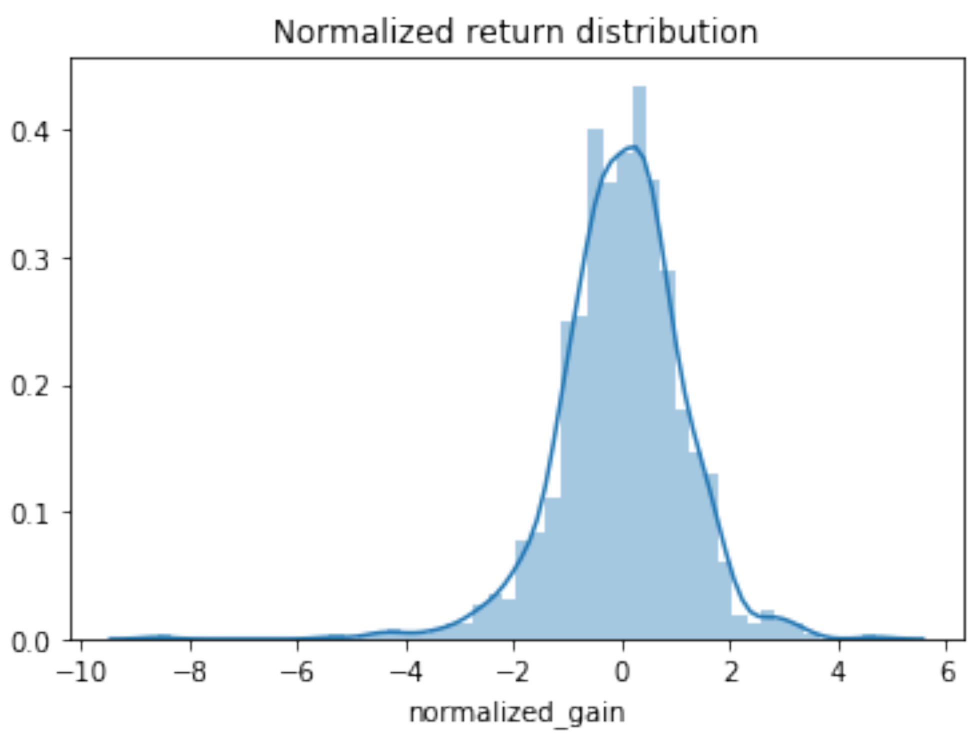
	bs_wt_lower
0	NaN
1	NaN
2	NaN
3	NaN
4	NaN
5	NaN
6	NaN
7	NaN
8	NaN
9	NaN
10	NaN
11	NaN
12	NaN
13	NaN
14	NaN
15	NaN
16	NaN
17	NaN
18	NaN
19	NaN
20	NaN
21	NaN
22	NaN
23	NaN
24	NaN
25	NaN
26	NaN
27	NaN
28	NaN
29	NaN
..	...
970	-0.108056

```
971    -0.108056
972    -0.108056
973    -0.108056
974    -0.108056
975    -0.099686
976    -0.110664
977    -0.110664
978    -0.110664
979    -0.110664
980    -0.110664
981    -0.110664
982    -0.110664
983    -0.110664
984    -0.110664
985    -0.110664
986    -0.110664
987    -0.110664
988    -0.110664
989    -0.110664
990    -0.110664
991    -0.110664
992    -0.110664
993    -0.110664
994    -0.110664
995    -0.110664
996    -0.110664
997    -0.110664
998    -0.110664
999    -0.110664

[1000 rows x 6 columns]
```

3.

```
In [3]: data['monthly_std'] = data['Return'].rolling(22).std().shift(1)
        data['normalized_gain'] = (data['Return'] -
            data['Return'].rolling(22).mean().shift(1)) / data['monthly_std']
        sns.distplot(data['Return'])
        plt.title('Original return distribution')
        plt.show()
        sns.distplot(data.loc[22:,'normalized_gain'])
        plt.title('Normalized return distribution')
        plt.show()
```


Original return distribution
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10
Return


Normalized return distribution
0.4
0.3
0.2
0.1
0.0
−10
−8
−6
−4
−2
0
2
4
6
normalized_gain

4.

```
In [4]: for i in range(252,length):
            weight = [lamb ** (i - j) * (1 - lamb) / (1 - lamb ** i) for j in range(1,i+1)]
            sorted_data = pd.DataFrame({'weight':weight,
                'return':data.loc[:i-1,'normalized_gain']}).sort_values('return')
            sorted_data.reset_index(drop=True,inplace=True)
            cum_weight = 0
            counter = 0
            while cum_weight < 0.01:
                cum_weight += sorted_data.loc[counter,'weight']
                counter += 1
            data.loc[i,'normalized_wt_var'] = sorted_data.loc[counter - 1, 'return']

        normalized_exception = (data['normalized_gain'] <
            data['normalized_wt_var'].shift(1)).value_counts()[True]
        original_skewness = data['Return'].skew()
        original_kurt = data['Return'].kurt()
        normalized_skewness = data['normalized_gain'].skew()
        normalized_kurt = data['normalized_gain'].kurt()
        print('Original skewness is',original_skewness)
        print('Original kurtosis is',original_kurt)
        print('Normalized skewness is',normalized_skewness)
        print('Normalized kurtosis is',normalized_kurt)
        print('The NO. of exceptions for normalized VaR is',normalized_exception)
```

```
Original skewness is -0.7757987755040311
Original kurtosis is 5.213764589841459
Normalized skewness is -0.7452927451944882
Normalized kurtosis is 4.627189932282226
The NO. of exceptions for normalized VaR is 8
```

There are only 8 exceptions of normalized gains.

5. Since the NO. of exceptions of normalized gains are less than the previous. This method is more accurate. Therefore, we could use this normalized VaR to measure risk.

Problem 2.

1. According to pigeonhole principle: for natural numbers k and m. if $n = km+1$. objects are distributed among m sets. then at least one of the sets will contain at least $k+1$ object.

$$8 > 2\times 3+1, \quad k=2,\ m=3.$$

at least $k+1=3$ of them are born within the same one-year period

2. Assume $W - W_0 \sim N(0, \sigma)$, and Losses in successive days are independent.

98% $VaR_5 = (z(0.98)\cdot\sigma - 0)\cdot\sqrt{5} = 10\,m.$

$$\sigma \approx 2.17755$$

$$VaR_{10} = \sqrt{10}\times(z(0.99)\cdot\sigma - 0) \approx 16.019 \text{ million}$$

3. 22 trading days in one month. $c = 0.99$. $n = 22$

$$P(\text{Exception} > 1) = 1 - \sum_{k=0}^{1} C_k^{22}(1-c)^k \cdot c^{n-k}$$

$$= 0.0202$$

For $c=0.99$, the expect NO. of exception is the trading window times 0.01. Then we can use a two-tailed test:

$$-2\ln[c^{n-m}(1-c)^m] + 2\ln[(1-m/n)^{n-m}(m/n)^m] \sim \chi^2(1)$$

If p-value is ~~large~~. small. we can reject null.

Therefore, there is bunching,

4. We can use cross-sectional data to calculate their daily VaR first. Assume losses in successive days are independent.

Annual VaR = 1-day. VaR. $\times\sqrt{252}$

Uber IPO 1-year 99% VaR at 10 million = 10 million $\times$ Annual VaR.

Cross-sectional data should have the same background as Uber. and should have been IPO in recent years.