A Study on S&P500 Index

Changhyn Ahn
Xi Edward Cai
Biwei Betti Tao

May 9, 2014

1 SP 500 Analysis

This is a group project for the Master's Capstone course STAT222 at UC Berkeley, Statistics Department. We have three group members: Changhyn Ahn, Xi Edward Cai, and Biwei Betti Tao; our research object is the public SP500 data, and our public github is at https://github.com/changhyunahn/STAT222_SP500 .

We splitted the research into two modules: 1) volatility comparison, and 2) weekend effect.

In modeule 1, we intend to find out if the market was more responsive to bad news as opposed to good news? Restoring from a recent financial crisis in 2008, the world witnessed a volatile stock market - in particular, in the notion of the financial sector - during the period of 2008 to 2013. One perhaps might wonder, "In chain of unprecedented bad news in that meltdown, has the market suffered from an everlasting increased systemic risk in contrast to pre-event state?" Looking at the historical quotes of global stock market indices (e.g. SP500, DJ, NASDAQ), we firstly attempt to identify world-renown infamous financial crisis (such as dot-com and sub-prime) occurred during the last decades - in particular, the late 90's and 00's. On the above datasets, we are going to apply time-series analytics (such as ARIMA and GARCH) to construct models, which can illustrate then-market volatilities with their corresponding parameters. Then, we compare the above figures against those of the pre-event times, referred as the control group.

In module 2, we intend to employ a series of conventional statistical methods to analyze whether or not the stock market, resembled by the SP500 Index, exhibited a tendency against the weekend evenings. Our intention is to change minds of the traders who believe in the Weekend Effect, defined as, "A phenomenon in financial markets in which stock returns on Mondays are often significantly lower than those of the immediate preceding Fridays."

2 Volatility Comparison

2.1 Price data infile

In [57]:
```
import os
print os.getcwd()
os.chdir("/Users/edwsurewin/Dropbox/Berkeley MA/222/STAT222-SP500")

from pandas import read_csv
df = read_csv("daily.csv", )
Date= tuple(df['Date'])
Price= tuple(df['Adj Close'])

price = range(16143)
```

```
date = range(16143)
for x in range(16143):
price[x] = Price[16142-x]
date[x] = Date[16142-x]
```

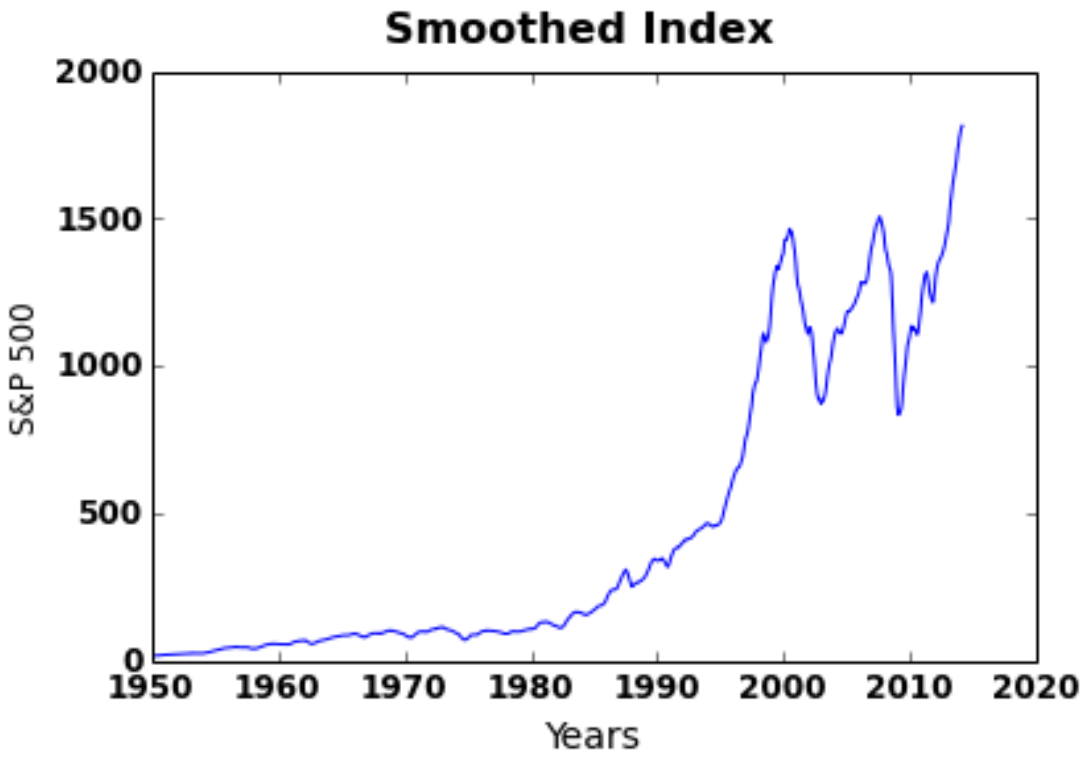
```
/Users/edwsurewin/Dropbox/Berkeley MA/222/STAT222-SP500
```

Financial markets definitely react to bad news, but how? We think that once a financial market suffers from a crisis, then it will be changed inherently and thereafter will not adjust back to previous state any time soon. We will show this by some time series analysis techniques. Basically, we are using S&P 500 index rather than using an arbitrary stock price. One reason why we choose this particular dataset as our primary source is that composed indiced, such as S&P500 itself, are highly representative to the global economies. S&P 500 is from the U.S. stock markets and the U.S. Stock markets heavily influence the world. In addition to the above rational, a stock market is relatively appropriate to serve the research objective than other financial instruments, including real estates, bonds and derivatives. Stock is usually more liquid and flexible than fixed income sercurities or long term bonds. Liquidity makes the index directly reflect upon news just in time. However, it is neither too risky nor too complex than futures or derivatives markets whose results might not be applied to a general idea as to how good news and bad news affect the economies. Finally, individual price is not used for our research because we are hoping to avoid having troubles from non-systematic risks or idiosyncratic risk. In other words, we intend primarily to find out the effects on not a single company but rather the whole economy.

2.2 Smoothed index

In [58]:
```
from decimal import *
getcontext().prec= 6
enddate = 2014 + Decimal(59)/Decimal(365)
begindate = 1950 + Decimal(3)/Decimal(365)
gap = (enddate - begindate)/(len(date)-1)
numdate = range(len(date))
for i in range(len(date)):
numdate[i] = begindate + i * gap

sprice = range(16143)
import numpy
for x in range(16143):
if x-60 < 0:
sprice[x] = numpy.mean(price[0:x+60])
else:
if x+60 > 16142:
sprice[x] = numpy.mean(price[x-60:16142])
else:
sprice[x] = numpy.mean(price[x-60:x+60])

from pylab import *
%matplotlib inline


from matplotlib import pyplot as plt

font = {'family' : 'normal',
        'weight' : 'bold',
        'size'   : 12}

matplotlib.rc('font', **font)

fig = plt.figure()
plt.plot(numdate,sprice)
fig.suptitle('Smoothed Index', fontsize=16)
plt.xlabel('Years', fontsize=14)
plt.ylabel('S&P 500', fontsize=12)
```

```
fig.savefig('test.jpg')
```

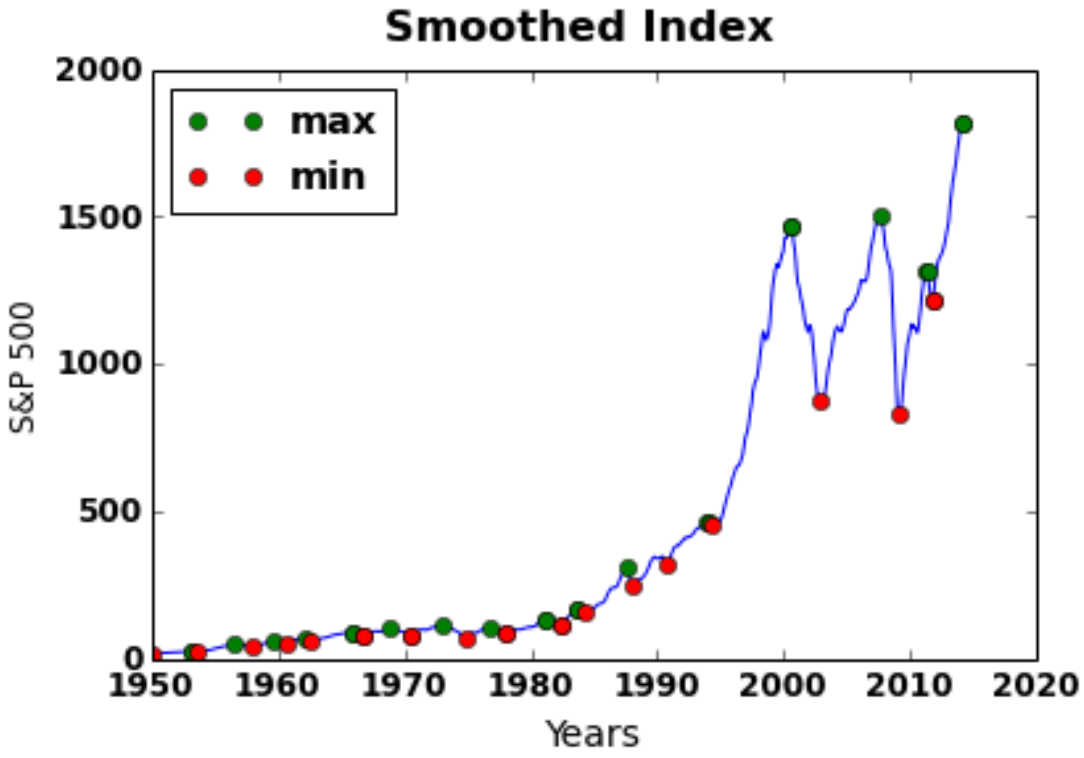


We tried to find the dates crises began and ended. We assumed that crises began when bubbles were severe. In other words, the highest value during a period of time and would be the starting date of crisis. But the problem is, the index data is so volatile that there are so many maxima. As a consequence, we rather use smoothing technique to estimate a general pattern and then pick the maxima. By k-nearest neighbor method, we estimated local maxima and chose the period we were using for further analysis. The plot shows the result. Now we can see that it has more smooth shape that we can pick maxima and minima.

2.3 Find max and min index

In [59]:
```
maxprice = range(0)
maxdate = range(0)
maxobs = range(0)
for i in range(16143):
maxi = 0
if i-180 < 0:
a = sprice[0:i-1] + sprice[i+1:i+180]
else:
if i+180 > 16142:
a = sprice[i-180:i-1] + sprice[i+1:16142]
else:
a = sprice[i-180:i-1] + sprice[i+1:i+180]
for number in a:
if number > maxi:
maxi = number
if sprice[i] > maxi:
maxprice = maxprice + [sprice[i]]
maxdate = maxdate + [numdate[i]]
maxobs = maxobs + [i]
```

```
minprice = range(0)
mindate = range(0)
minobs = range(0)
for i in range(16143):
mini = 2000
if i-180 < 0:
a = sprice[0:i-1] + sprice[i+1:i+180]
else:
if i+180 > 16142:
a = sprice[i-180:i-1] + sprice[i+1:16142]
else:
a = sprice[i-180:i-1] + sprice[i+1:i+180]
for number in a:
if number < mini:
mini = number
if sprice[i] < mini:
minprice = minprice + [sprice[i]]
mindate = mindate + [numdate[i]]
minobs = minobs + [i]

fig = plt.figure()
plt.plot(numdate,sprice)
plt.plot(maxdate, maxprice, "o", label="max")
plt.plot(mindate, minprice, "o", label="min")
plt.legend(loc=0)
fig.suptitle('Smoothed Index', fontsize=16)
plt.xlabel('Years', fontsize=14)
plt.ylabel('S&P 500', fontsize=12)
fig.savefig('test.jpg')
```

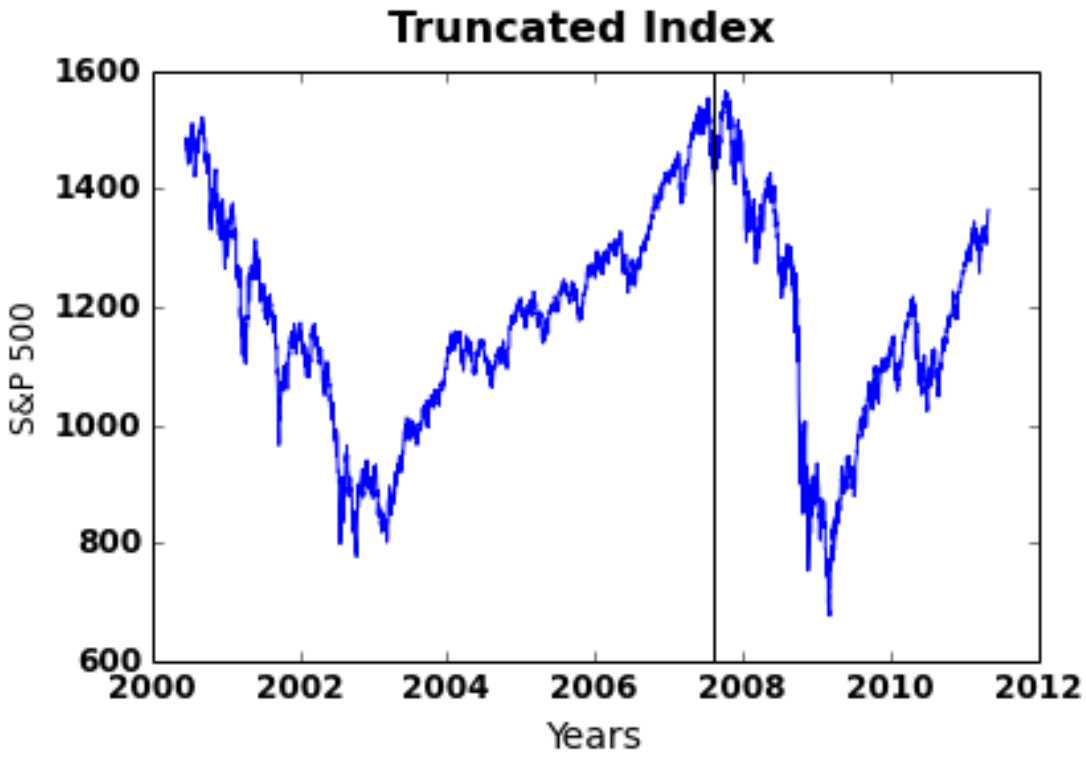


Based on dates when crises start, we cut off two different eras supposedly possessing different characteristics. The first period was from Jun 16th, 2000 to August 10th, 2007. The index during this period was closely related to so-called the Dot-com bubble. The other period started from August 11th, 2007 to May 2nd, 2011. That was related to the recent

financial crisis mainly because of the serial credit defaults in the subprime mortgages.

2.4 Movements of S&P 500 Daily Price

In [60]:
```
rreturn = range(len(price)-1)
for i in range(len(price)-1):
rreturn[i]=(price[i+1] - price[i])/price[i]

preev = price[12697:14493]
postev = price[14494:15431]

prerr = rreturn[12697:14493]
postrr = rreturn[14494:15430]

predate = numdate[12697:14493]
postdate = numdate[14494:15431]

fig = plt.figure()
plt.plot(predate+postdate,preev+postev)
plt.axvline(maxdate[len(maxdate)-5], color='k')
fig.suptitle('Truncated Index', fontsize=16)
plt.xlabel('Years', fontsize=14)
plt.ylabel('S&P 500', fontsize=12)
fig.savefig('test.jpg')
```



This plot shows the index during the two partitioned periods. Just looking the two periods does not give many intuitive ideas whether they are different in any senses, for example, the second period might be more volatile than the other. We only can claim that from this plot that the economic cycle was going to be short. In other words, the economy may encounter crisis and bubble more frequently than before. We are now using some time series data for further analysis.As we can see form the plot of stock prices, price movement doesn't have an apparent pattern. The index

really screwed up and hit the buttom during the 2008 crisis. After that, it started to go up again. I almost climbed over 1400 near the starting point but still carried with big fluctuations. The variance of the stock price seems more volatile for post-event.

2.5 Implementation of Autoregressive Integrated Moving Average Model

We want, at this step, to get a general view and better understanding of pre-event and post-event time series, to see if they are indeed of different nature. We intend to implement the Autoregressive Integrated Moving Average Model. In this way, not only could we get specific formulas describing each of the time series that serve as an accurate foundation for our argument, and it would allow us to do some predictions and simulations on each data set. To simplify, we take just one level of difference, d=1, in ARIMA Model.

In [61]:
```
import statsmodels.api as sm
from statsmodels import *
modelpre = tsa.arima_model.ARIMA(preev, [1, 0, 1],freq='M').fit()
modelpost = tsa.arima_model.ARIMA(postev, [1, 1, 1],freq='M').fit()
```

In [62]:
```
modelpre.summary() # Pre-event model summary
```

Out [62]:

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                              ARMA Model Results
==============================================================================
========
Dep. Variable:                      y   No. Observations:
1796
Model:                     ARMA(1, 1)   Log Likelihood
-7004.452
Method:                        css-mle   S.D. of innovations
11.936
Date:                Fri, 09 May 2014   AIC
14016.904
Time:                        03:19:54   BIC
14038.877
Sample:                             0   HQIC
14025.016

==============================================================================
========
                 coef    std err          z      P>|z|      [95.0%
Conf. Int.]
------------------------------------------------------------------------------
--------
const       1292.5804    132.923      9.724      0.000      1032.056
1553.105
ar.L1.y        0.9983      0.001    811.567      0.000         0.996
1.001
ma.L1.y       -0.0424      0.025     -1.691      0.091        -0.092
0.007
                                    Roots
==============================================================================
=======
```

```
                  Real           Imaginary           Modulus
Frequency
-----------------------------------------------------------------------
-------
AR.1            1.0017           +0.0000j            1.0017
0.0000
MA.1           23.5871           +0.0000j           23.5871
0.0000
-----------------------------------------------------------------------
-------
"""
```

In [63]:
```
modelpost.summary() #Post-event model summary
```

Out [63]:

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                             ARIMA Model Results
==============================================================================
========
Dep. Variable:                    D.y   No. Observations:
936
Model:                 ARIMA(1, 1, 1)   Log Likelihood
-4044.155
Method:                       css-mle   S.D. of innovations
18.206
Date:                Fri, 09 May 2014   AIC
8096.310
Time:                        03:19:56   BIC
8115.677
Sample:                             1   HQIC
8103.695

==============================================================================
========
                 coef    std err          z      P>|z|      [95.0%
Conf. Int.]
------------------------------------------------------------------------------
--------
const         -0.0903      0.473     -0.191      0.849        -1.016
0.836
ar.L1.D.y      0.2611      0.153      1.712      0.087        -0.038
0.560
ma.L1.D.y     -0.4135      0.143     -2.893      0.004        -0.694
-0.133
                                    Roots
==============================================================================
=======
                  Real           Imaginary           Modulus
Frequency
-----------------------------------------------------------------------
-------
AR.1            3.8293           +0.0000j            3.8293
0.0000
```

```
MA.1            2.4186           +0.0000j            2.4186
0.0000
-------------------------------------------------------------------------------
"""
```

Actually, python has not enough packages to analyze time series data. So we are using rpy2 to use some useful packages in R.

In [64]:
```
import rpy2 as rpy2
%load_ext rmagic
```

```
The rmagic extension is already loaded. To reload it, use:
  %reload_ext rmagic
```

In [65]:
```
%%R
install.packages("forecasting")
```

In [66]:
```
%%R
library("forecast")
```

A list with component ar and/or ma giving the AR and MA coefficients respectively. Optionally a component order can be used. An empty list gives an ARIMA(0, 0, 0) model, that is white noise. n, length of output series, before un-differencing. A strictly positive integer.

The ARIMA result tables are very detailed. The pre-event time series is fitted with ARMA(1,1) model while ARIMA(1,1,1) is chosen for the post-event time series. Each coefficient in both model has a very small standard deviation error and p-value. We can see from the criteria that the models fit pretty well. And since the resulting models are very different, we can conclude that those two are indeed not similar time series.

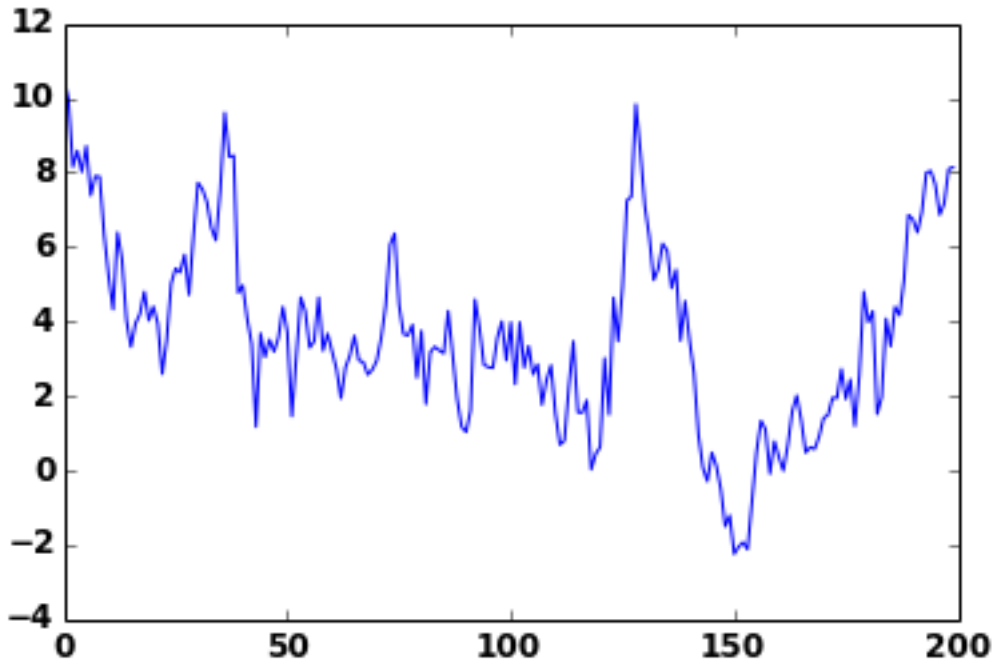
2.6 Forecast and Simulation

The data sets in pre-event and post-event are both of limited sizes. With the specific formulas obtained from the ARIMA model results, we want to use the forecast and simulation functions to get a general picture of the volatility difference between the two events in a larger background of data.

In [67]:
```
%%R -o presim
presim <- as.vector(arima.sim(list(order = c(1,0,1), ar = 0.9983, ma = -0.0424), n = 2
```

In [68]:
```
plot(range(len(presim)), presim)
```

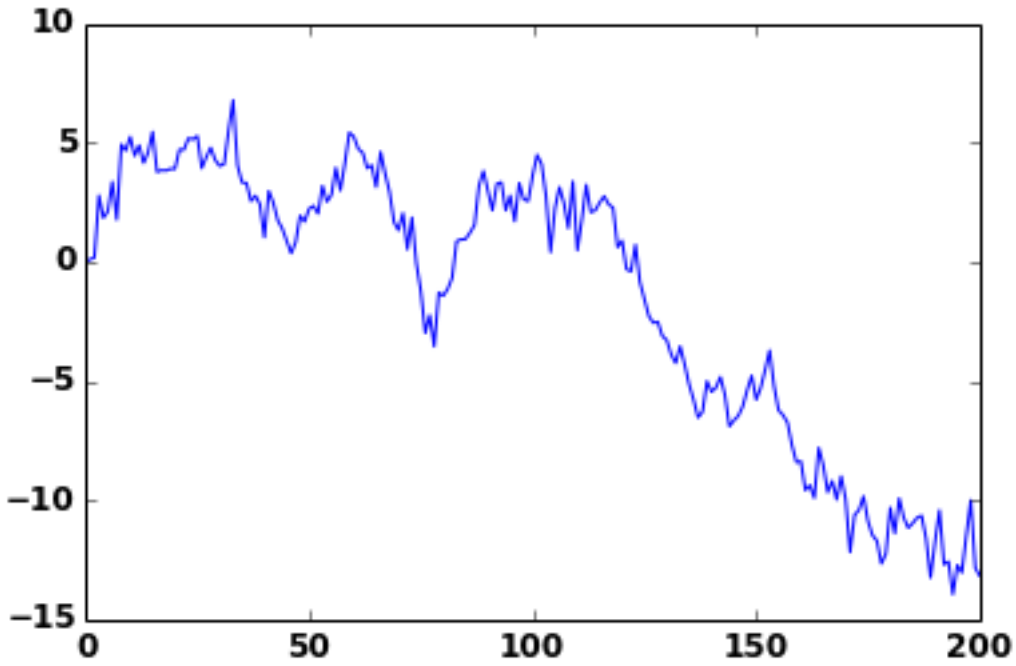
Out [68]:
```
[<matplotlib.lines.Line2D at 0x1084d3590>]
```



In [69]:
```
%%R -o postsim
postsim <- as.vector(arima.sim(list(order = c(1,1,1), ar = 0.2611, ma = -0.4135), n =
```

In [70]:
```
plot(range(len(postsim)), postsim)
```

Out [70]:
```
[<matplotlib.lines.Line2D at 0x111600dd0>]
```



ARMA model in R environment from pre event data is made into the object arimapre

In [71]:
```
%%R -i preev
library(stats)
arimapre <- arima(ts(preev),c(1,0,1))
print(arimapre)
```

```
Series: ts(preev)
ARIMA(1,0,1) with non-zero mean

Coefficients:
         ar1      ma1  intercept
      0.9985  -0.0423  1185.8297
s.e.  0.0018   0.0251   187.6526

sigma^2 estimated as 142.5:  log likelihood=-7004.82
AIC=14017.65   AICc=14017.67   BIC=14039.62
```

ARIMA model in R environment from post event data is made into the object arimapost

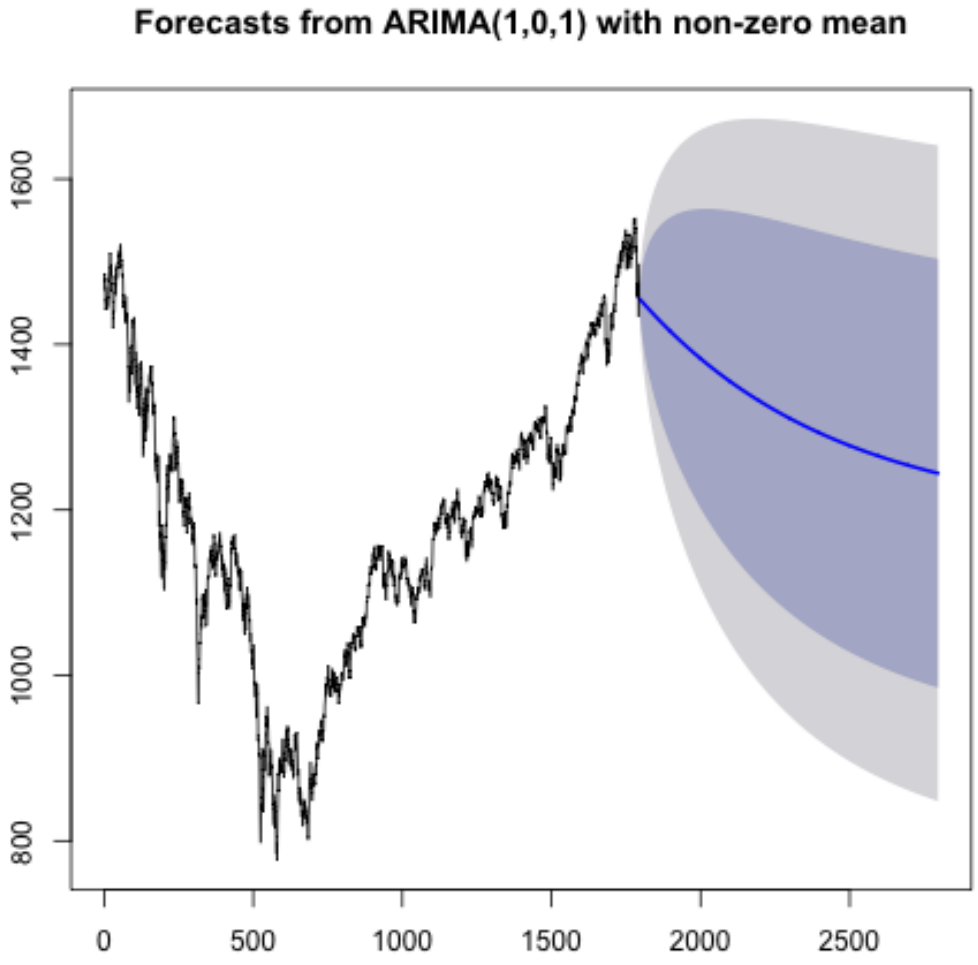
In [72]:
```
%%R -i postev
arimapost <- arima(ts(postev),c(1,1,1))
print(arimapost)
```

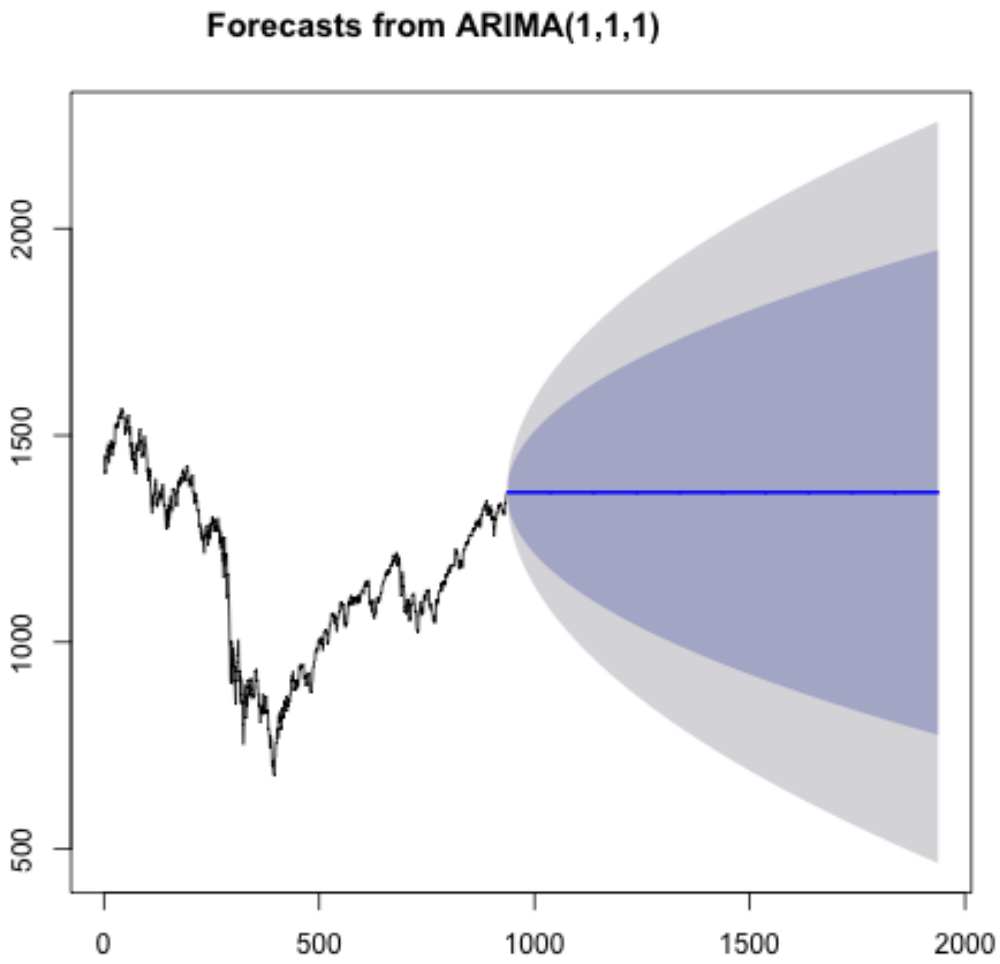
```
Series: ts(postev)
ARIMA(1,1,1)

Coefficients:
         ar1      ma1
      0.2610  -0.4133
s.e.  0.1525   0.1429
```

```
sigma^2 estimated as 331.5:  log likelihood=-4044.17
AIC=8094.35   AICc=8094.37   BIC=8108.87
```

Here are plots to forecast from the two models

In [73]:
```
%%R
library("forecast")
plot(forecast(object=arimapre, h=1000))
plot(forecast(object=arimapost, h=1000))
```





The forecast pictures clearly show an obvious difference in the standard deviation of two events. The two forecast graphs are not in the same unit. Pre-events' s standard errors is around 250 scale while post-event's standard deviation is around 500. The simulation plots also present post-event with larger volatility.

2.7 The Implementation of Generalized AutoRegressive Conditional Heteroskedasticity Model

As studies show, the asymmetric nature of information and pricing, the heteroscedasticity, suggests the use of nonlinear time series structures to model the attitude of investors toward risk and ex- pected return. For example, Bera and Higgins (1993, p.315) remarked that "a major contribution of the ARCH literature is the finding that apparent changes in the volatility of economic time series may be predictable and result from a specific type of nonlinear dependence rather than exogenous structural changes in variables." Here, in the case of financial data, large and small errors tend to occur in clusters, i.e., large returns are followed by more large returns, and small returns by more small returns. This character of volatility clustering will be show in the section of log return plots. This suggests that returns are serially correlated.

To address the previous observed dynamic pattern volatility change, we introduce the generalized autoregressive conditional heteroskedasticity model. Shocks are assumed to be random, GARCH uses a nonlinear function relating the

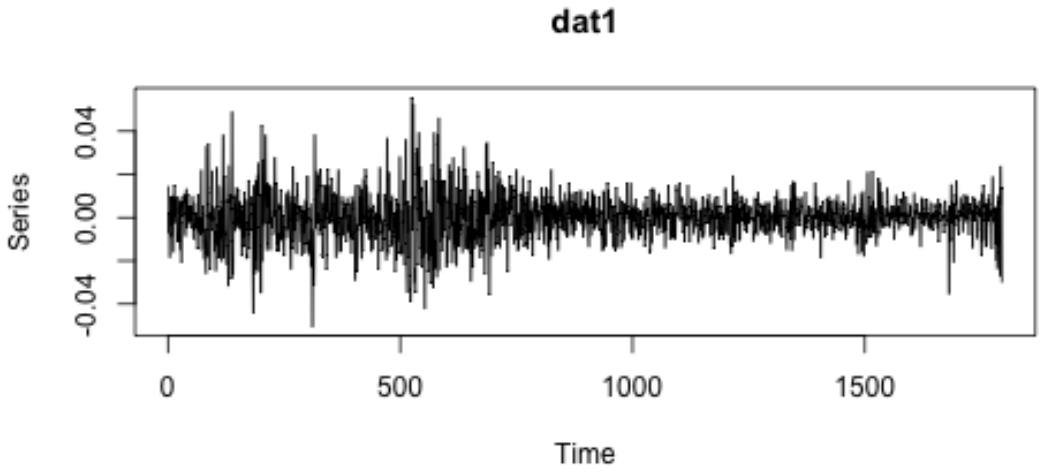
observed time series and the underlying shocks. Since in GARCH model conditional variance is dependent on its own previous lags, the volatility clustering shown by the previous plots would be characterized. The following resulting functions identify the different nature of the events' volatilities.Summary statistics from GARCH model This shows volatility from pre event data

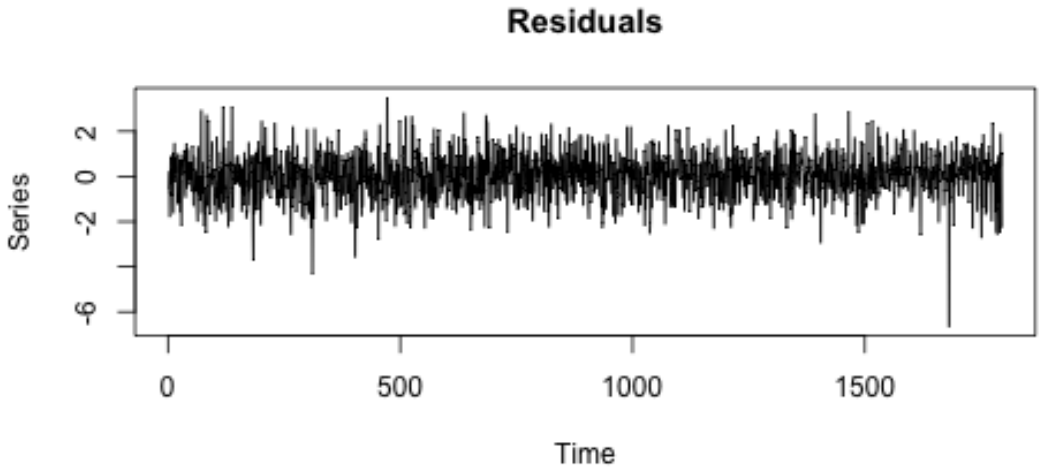
In [74]:
```
%%R
install.packages("tseries")
library("tseries")
dat1 <- diff(log(preev))
dat1 <- ts(dat1)
dat1.garch <- garch(dat1)
summary(dat1.garch)
plot(dat1.garch)
```

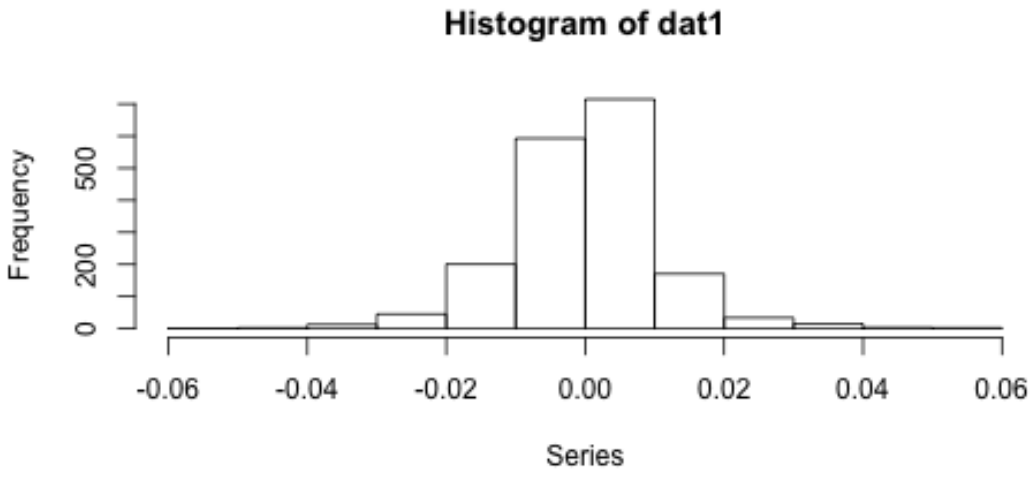
```
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:

trying URL 'http://cran.cnr.Berkeley.edu/bin/macosx/contrib/3.0/tserie
s_0.10-32.tgz'
Content type 'application/x-gzip' length 308854 bytes (301 Kb)
opened URL
==================================================
downloaded 301 Kb


The downloaded binary packages are in
        /var/folders/vc/r5p1rx6x1275xdrjkcfv2mlc0000gn/T//RtmpAMAR0T/d
ownloaded_packages

 ***** ESTIMATION WITH ANALYTICAL GRADIENT *****


     I     INITIAL X(I)        D(I)

     1     1.033614e-04     1.000e+00
     2     5.000000e-02     1.000e+00
     3     5.000000e-02     1.000e+00

    IT   NF      F         RELDF    PRELDF    RELDX   STPPAR   D*STEP
NPRELDF
     0    1 -7.270e+03
     1    7 -7.270e+03  5.41e-05  5.75e-04  1.0e-04  4.2e+10  1.0e-05
1.20e+07
     2    8 -7.271e+03  1.13e-04  1.30e-04  4.8e-05  2.0e+00  5.0e-06
1.32e+01
     3    9 -7.271e+03  1.58e-06  1.48e-06  4.9e-05  2.0e+00  5.0e-06
1.33e+01
     4   17 -7.296e+03  3.37e-03  4.95e-03  4.5e-01  2.0e+00  8.2e-02
1.33e+01
     5   20 -7.361e+03  8.89e-03  7.72e-03  7.2e-01  2.0e+00  3.3e-01
2.35e+00
     6   22 -7.383e+03  2.98e-03  2.31e-03  7.7e-02  2.0e+00  6.6e-02
4.02e+00
```

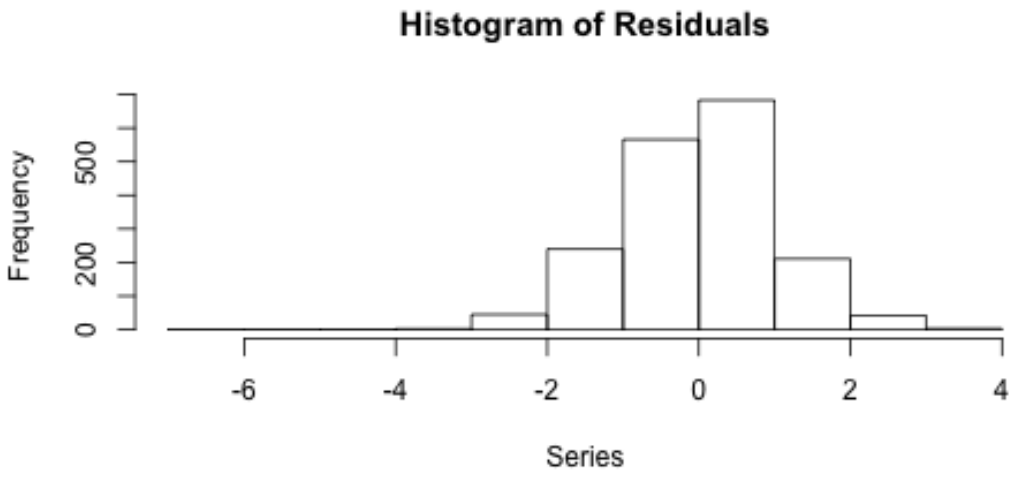
```
     7   24 -7.423e+03  5.36e-03  5.63e-03  1.3e-01  2.0e+00  1.3e-01
5.75e+01
     8   26 -7.435e+03  1.68e-03  2.19e-03  4.7e-02  2.0e+00  5.8e-02
4.07e+01
     9   27 -7.441e+03  7.64e-04  2.26e-03  4.3e-02  2.0e+00  5.8e-02
5.00e+00
    10   29 -7.443e+03  2.94e-04  9.30e-04  7.1e-03  2.0e+00  1.0e-02
5.96e-01
    11   30 -7.445e+03  2.81e-04  4.77e-04  7.1e-03  2.0e+00  1.0e-02
2.31e-03
    12   31 -7.445e+03  4.30e-05  6.41e-05  7.0e-03  2.0e+00  1.0e-02
8.77e-03
    13   33 -7.445e+03  7.87e-07  1.74e-05  1.9e-03  2.0e+00  2.7e-03
7.74e-03
    14   34 -7.446e+03  1.55e-05  1.90e-05  9.8e-04  2.0e+00  1.4e-03
1.49e-03
    15   38 -7.461e+03  2.06e-03  1.02e-03  3.9e-02  0.0e+00  7.3e-02
1.02e-03
    16   40 -7.463e+03  2.82e-04  4.63e-04  1.9e-02  1.9e+00  2.9e-02
1.49e-02
    17   43 -7.488e+03  3.29e-03  3.28e-03  5.6e-02  1.1e+00  1.2e-01
3.55e-02
    18   45 -7.490e+03  2.50e-04  8.52e-04  1.1e-02  1.9e+00  2.3e-02
1.64e-02
    19   46 -7.497e+03  9.62e-04  9.86e-04  1.1e-02  1.9e+00  2.3e-02
1.22e-02
    20   48 -7.500e+03  4.20e-04  4.78e-04  1.0e-02  1.3e+00  2.3e-02
6.38e-03
    21   49 -7.502e+03  2.42e-04  3.06e-04  1.1e-02  1.2e+00  2.3e-02
9.78e-04
    22   50 -7.503e+03  1.33e-04  2.65e-04  9.4e-03  9.4e-01  2.3e-02
4.48e-04
    23   51 -7.503e+03  6.81e-06  2.54e-05  1.4e-03  0.0e+00  3.1e-03
2.54e-05
    24   52 -7.503e+03  7.75e-06  1.53e-05  1.5e-03  6.5e-01  3.1e-03
1.60e-05
    25   53 -7.503e+03  1.57e-06  4.89e-06  7.2e-04  0.0e+00  1.4e-03
4.89e-06
    26   54 -7.503e+03  1.60e-06  1.59e-06  6.6e-04  2.7e-02  1.4e-03
1.59e-06
    27   55 -7.503e+03  4.77e-09  7.95e-09  4.0e-05  0.0e+00  9.5e-05
7.95e-09
    28   56 -7.503e+03  2.93e-10  4.73e-10  1.4e-05  0.0e+00  3.7e-05
4.73e-10
    29   57 -7.503e+03  5.21e-11  1.42e-12  9.9e-07  0.0e+00  2.2e-06
1.42e-12
    30   58 -7.503e+03 -4.37e-12  1.12e-14  6.1e-08  0.0e+00  1.3e-07
1.12e-14

 ***** RELATIVE FUNCTION CONVERGENCE *****

 FUNCTION    -7.502876e+03   RELDX        6.127e-08
 FUNC. EVALS      58         GRAD. EVALS      30
 PRELDF       1.120e-14      NPRELDF      1.120e-14
```

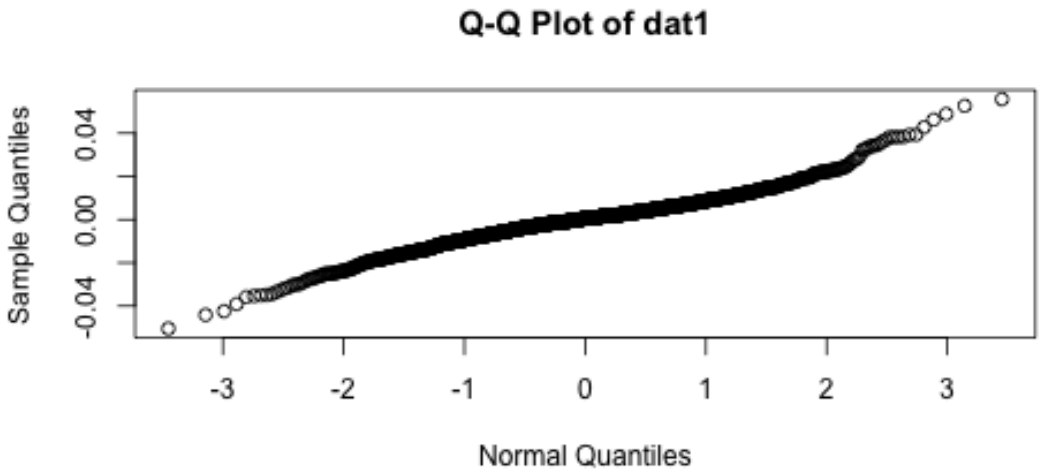
I	FINAL X(I)	D(I)	G(I)
1	1.120162e-06	1.000e+00	5.122e+01
2	6.861711e-02	1.000e+00	7.346e-04
3	9.211163e-01	1.000e+00	1.032e-03

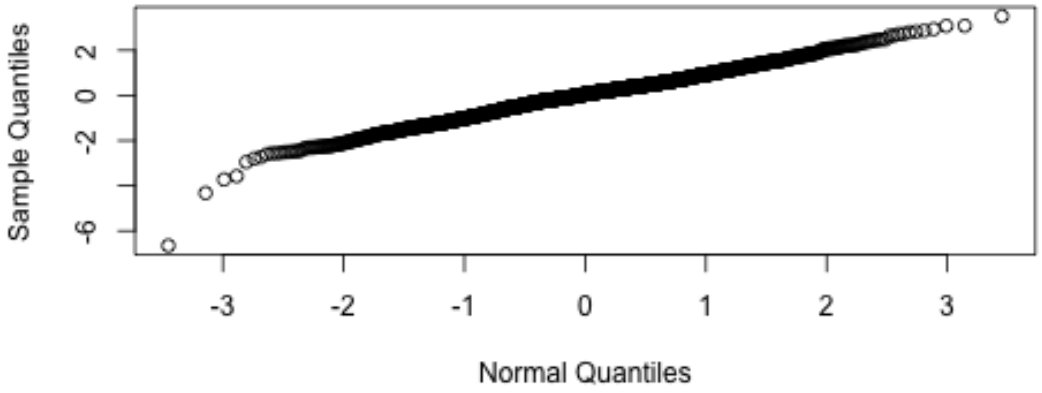


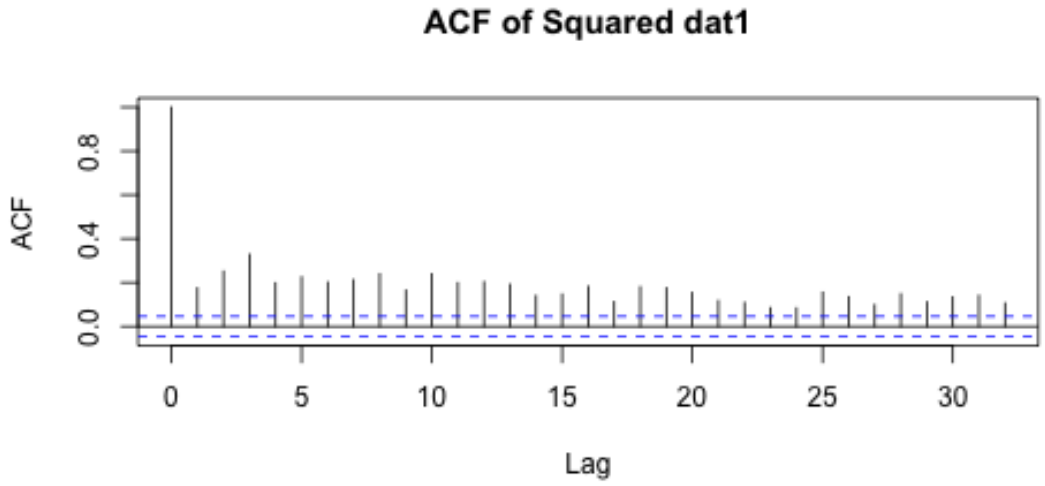


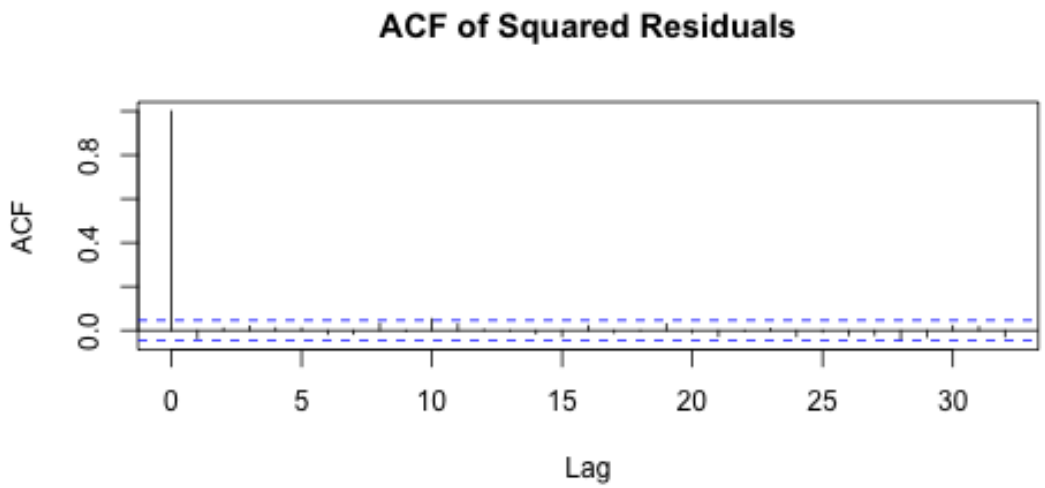

Histogram of dat1
Frequency
0
200
500
-0.06
-0.04
-0.02
0.00
0.02
0.04
0.06
Series


Histogram of Residuals
Frequency
0
200
500
-6
-4
-2
0
2
4
Series


Q-Q Plot of dat1
Sample Quantiles
-0.04
0.00
0.04
-3
-2
-1
0
1
2
3
Normal Quantiles


Q-Q Plot of Residuals
Sample Quantiles
-6
-2
0
2
-3
-2
-1
0
1
2
3
Normal Quantiles





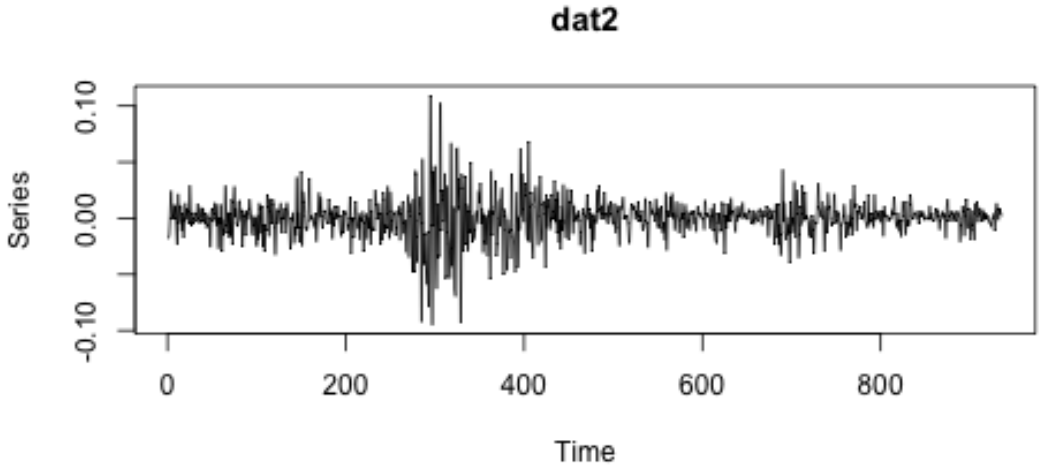
Summary statistics from GARCH model This shows volatility from pre event data

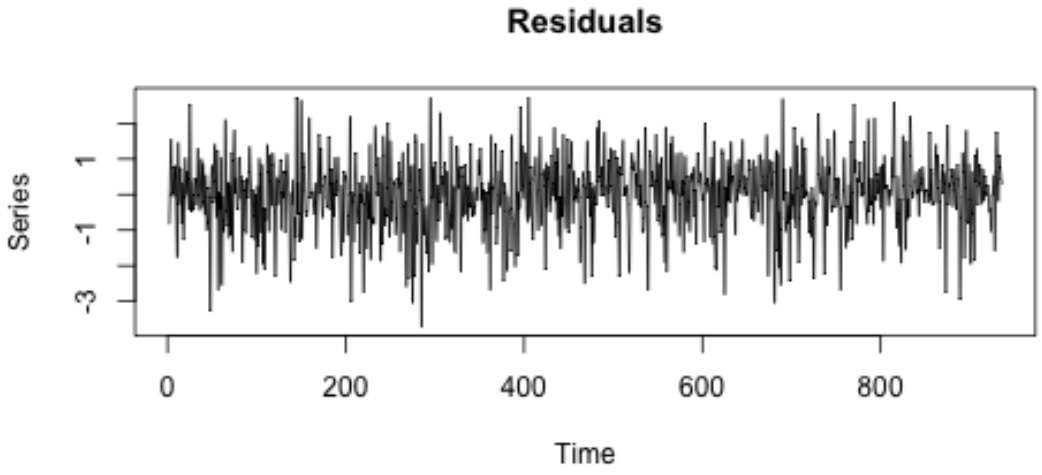
In [75]:
```
%%R
dat2 <- diff(log(postev))
dat2 <- ts(dat2)
dat2.garch <- garch(dat2)
summary(dat2.garch)
plot(dat2.garch)
```

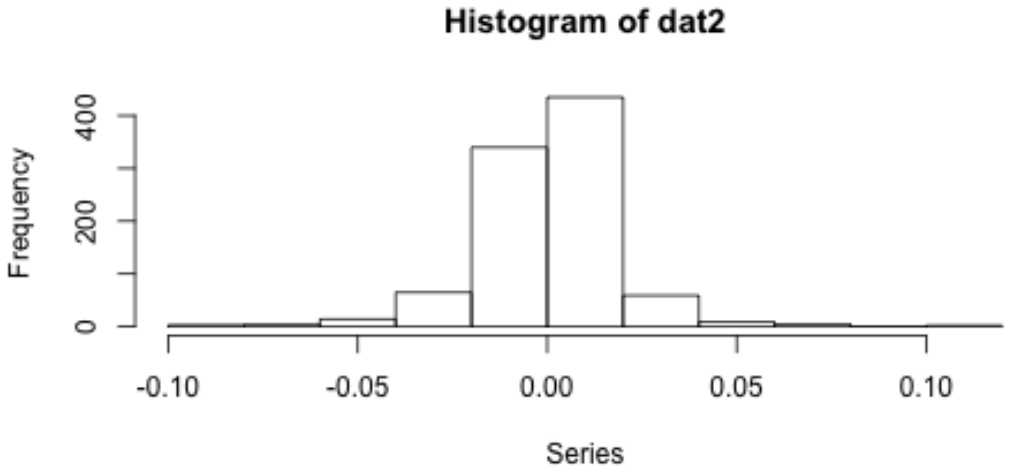
```
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:

 ***** ESTIMATION WITH ANALYTICAL GRADIENT *****


     I     INITIAL X(I)        D(I)
```

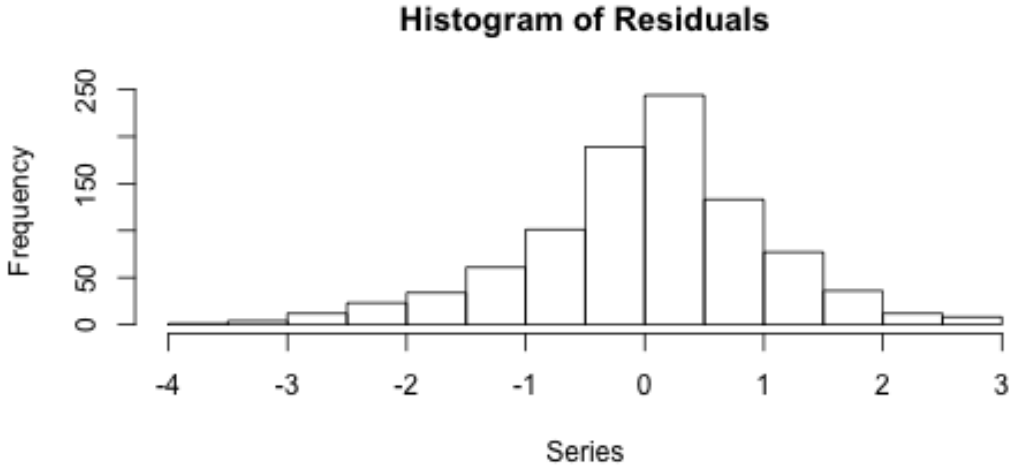
```
     1     2.840482e-04     1.000e+00
     2     5.000000e-02     1.000e+00
     3     5.000000e-02     1.000e+00

    IT   NF      F         RELDF    PRELDF    RELDX   STPPAR   D*STEP
NPRELDF
     0    1 -3.325e+03
     1    7 -3.327e+03  4.16e-04  6.03e-04  1.6e-04  7.4e+09  1.6e-05
2.25e+06
     2    8 -3.327e+03  5.05e-05  6.09e-05  1.3e-04  2.0e+00  1.6e-05
1.88e+01
     3    9 -3.327e+03  3.58e-06  3.69e-06  1.6e-04  2.0e+00  1.6e-05
1.86e+01
     4   17 -3.358e+03  9.32e-03  1.63e-02  6.0e-01  2.0e+00  1.6e-01
1.85e+01
     5   19 -3.397e+03  1.16e-02  1.16e-02  3.0e-01  2.0e+00  1.6e-01
1.04e+01
     6   21 -3.457e+03  1.72e-02  1.43e-02  3.8e-01  1.8e+00  3.1e-01
2.83e-01
     7   33 -3.468e+03  3.24e-03  1.16e-02  2.3e-05  3.3e+00  2.4e-05
4.11e+01
     8   43 -3.481e+03  3.72e-03  3.53e-03  5.0e-02  2.0e+00  5.9e-02
2.14e+01
     9   49 -3.483e+03  4.09e-04  6.81e-04  4.1e-06  1.6e+03  4.6e-06
1.14e+00
    10   50 -3.483e+03  6.95e-06  9.09e-06  3.9e-06  2.0e+00  4.6e-06
3.10e-02
    11   51 -3.483e+03  7.00e-07  6.99e-07  4.0e-06  2.0e+00  4.6e-06
3.05e-02
    12   59 -3.494e+03  3.20e-03  4.14e-03  6.1e-02  1.6e+00  7.6e-02
3.04e-02
    13   60 -3.502e+03  2.27e-03  2.06e-03  5.2e-02  3.2e-01  7.6e-02
2.18e-03
    14   62 -3.514e+03  3.52e-03  3.41e-03  8.9e-02  0.0e+00  1.6e-01
3.41e-03
    15   64 -3.543e+03  8.20e-03  5.90e-03  8.4e-02  0.0e+00  1.6e-01
1.12e-02
    16   67 -3.543e+03  8.06e-05  9.10e-04  5.6e-03  2.0e+00  1.1e-02
4.43e-01
    17   68 -3.546e+03  5.71e-04  8.23e-04  2.8e-03  2.0e+00  5.7e-03
9.04e-02
    18   70 -3.547e+03  2.89e-04  3.76e-04  8.1e-03  2.0e+00  1.7e-02
6.47e-02
    19   72 -3.547e+03  2.47e-05  3.57e-05  1.6e-03  1.8e+00  3.1e-03
4.33e-04
    20   73 -3.547e+03  2.95e-05  6.25e-05  1.7e-03  1.5e+00  3.1e-03
2.35e-04
    21   74 -3.547e+03  6.21e-06  1.09e-05  8.0e-04  0.0e+00  1.4e-03
1.09e-05
    22   75 -3.547e+03  1.63e-06  2.02e-06  5.7e-04  1.1e+00  1.4e-03
3.78e-06
    23   76 -3.547e+03  5.60e-07  9.16e-07  3.5e-04  0.0e+00  6.3e-04
9.16e-07
    24   77 -3.547e+03  9.14e-08  7.30e-08  8.9e-05  0.0e+00  1.6e-04
```

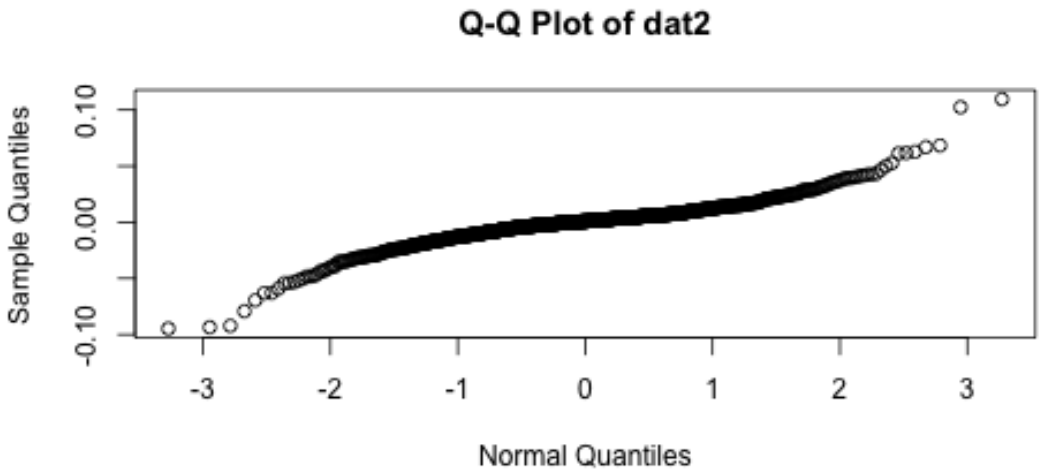
```
7.30e-08
    25   78 -3.547e+03  2.30e-10  2.94e-10  1.2e-05  0.0e+00  2.9e-05
2.94e-10
    26   79 -3.547e+03 -9.18e-11  1.64e-12  9.2e-07  0.0e+00  2.3e-06
1.64e-12

 ***** RELATIVE FUNCTION CONVERGENCE *****

 FUNCTION    -3.546765e+03   RELDX        9.204e-07
 FUNC. EVALS      79         GRAD. EVALS      26
 PRELDF       1.637e-12      NPRELDF      1.637e-12

      I      FINAL X(I)        D(I)          G(I)

      1    2.373868e-06     1.000e+00     4.157e+01
      2    9.723640e-02     1.000e+00     4.273e-03
      3    8.941176e-01     1.000e+00    -1.297e-03
```

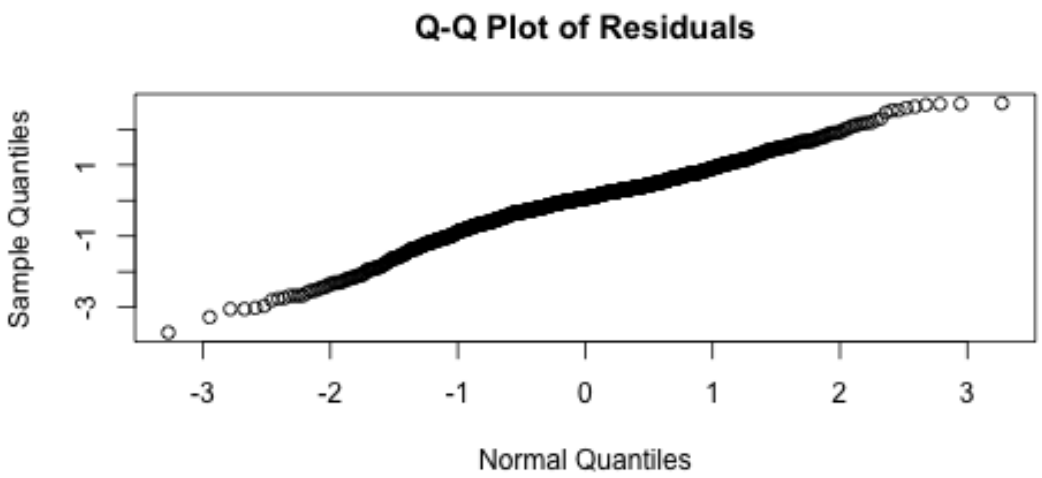


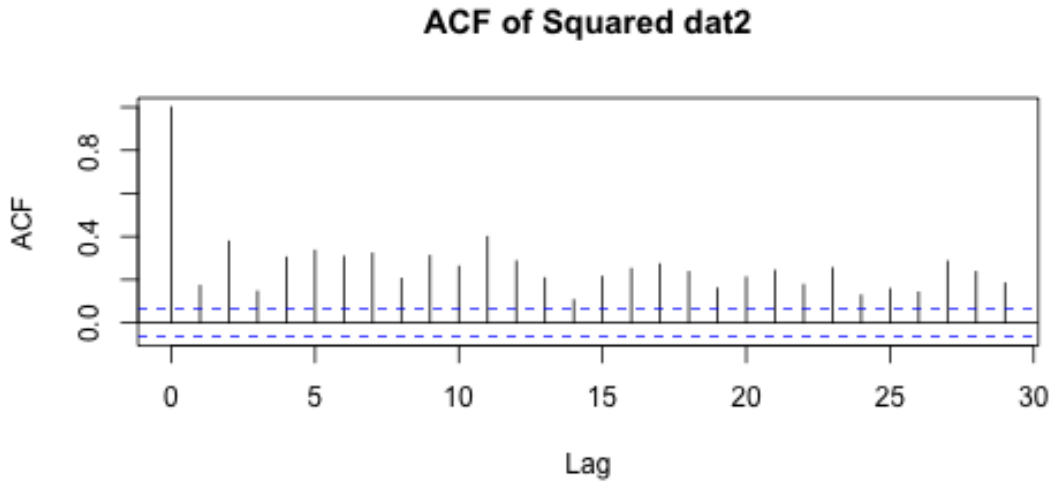


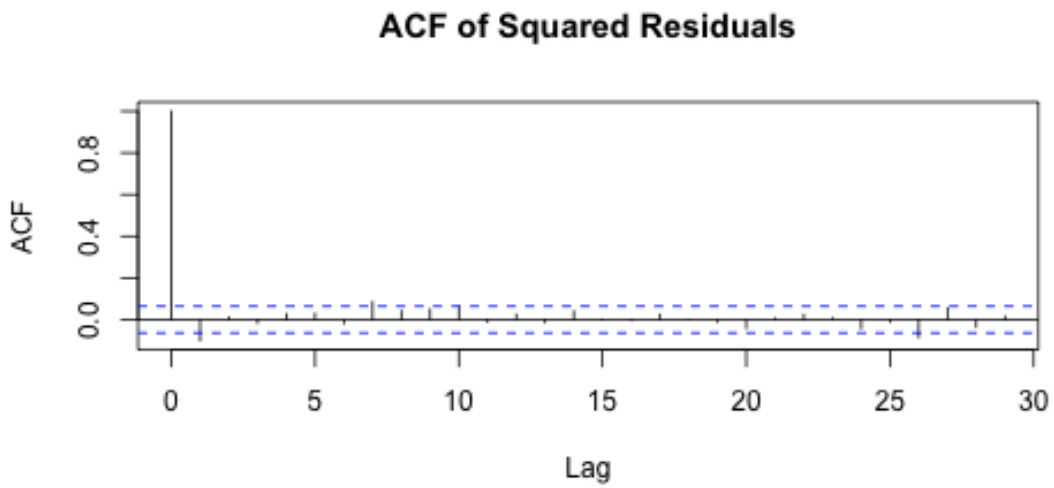

Histogram of dat2
Frequency
0
200
400
-0.10
-0.05
0.00
0.05
0.10
Series


Histogram of Residuals
Frequency
0
50
150
250
-4
-3
-2
-1
0
1
2
3
Series


Q-Q Plot of dat2
Sample Quantiles
-0.10
0.00
0.10
-3
-2
-1
0
1
2
3
Normal Quantiles


Q-Q Plot of Residuals
Sample Quantiles
-3
-1
1
-3
-2
-1
0
1
2
3
Normal Quantiles





2.8 Log Returns and Residuals

Since there're no observable mathematical patterns in the S&P500 daily price data, we use the first degree of difference on the price. i.e. returns. And then transform it into log returns to distinguish between series values being uncorrelated and series values being independent. Because if series values are truly independent from each other, then nonlinear instantaneous transformations such as taking logarithms preserves independence. However, those kind of transformation could alter the characteristic of correlation, as correlation, by its definition, is only a measure of linear dependence.

The first figures in pre-event and post-event illustrate how log returns and residuals of S&P500 daily price evolve through time. As we can see from the plot log returns give us a more stabilized and a clearer view of the variance over the time since the plot generally moves around a horizontal line.

Now we take a closer look to observe the volatility character of the shown plots. The post event plots exhibit larger fluctuations than the pre event ones. Also, we observe from the plots of both event that volatility clustering exists, i.e. large changes tend to be followed by large charges and small changes tend to be followed by small changes, of either sign. This is also a "stylized fact" of many economic and financial variables. In other words, volatility clustering

means there are times stock price is much more volatile than other times. Thus, the conditional variance of the time series varies over time, that is, there exists a dynamical pattern in the volatility of a time series.

2.9 Normality of the Data

Since the distributions of log returns and squared residuals of the two-time periods, as shown in the histogram in the second part of the figures, are all bell-shaped, the transformed data could be normally distributed. To test the normality of transformed data. We get the quatitiles of normal distribution plot through python.

2.10 The Quantiles of Normal Distribution Plot

The third part of the figures presents the quantitiles of normal distribution plots of the data. If the data set is perfectly normal the points should all fall on the 45 degree y=x line.

The linearity of the pouts suggests the log returns and residuals from both event are normally distributed. The QQ plot of log returns suggests that its distribution of post event have a thicker tail than that of a normal distribution and may be somewhat skewed to the right.

2.11 Autocorrelation Function Plot

Perform the autocorrelation function on squared log returns. Compare the plot of autocorrelation function from log returns of pre and post event, we can see that the ACF values of both event keep falling far out of the range of two standard deviations away even after long lags, that is the ACF values are not tailing off gradually. So the time series fit AR Model. And almost all the ACF values from post event are around or larger than 0.2 while more than half of ACF values from pre event are smaller than 0.2.

As for autocorrelation function plot from residuals of pre and post event, the ACF values of both event fall into the range of two standard deviations gradually as the lag becomes large, that is the ACF values tail off gradually. So the time series fit MA Model. And ACF values from post event are also generally larger than pre event.

The time series from the post event are more autocorrelated than pre event. The ACF results also collaborate with our point that the post-event are more volatile than the previous one, i.e. the effect of financial crisis is persistent

3 Weekend Effect

3.1 Introduction

The Weekend Effect is defined as, “A phenomenon in financial markets in which stock returns on Mondays are often significantly lower than those of the immediate preceding Fridays.” Some theories that explain the effect attribute the tendency for companies to release bad news on Friday after the markets close to depressed stock prices on Monday. However, the others do not acknowledge such weekly tendencies and stated that it may be a pure stochastic event.

In this module, we intend to employ a series of conventional statistical methods to analyze whether or not the stock market, resembled by the SP500 Index, exhibited a tendency against the weekend evenings. Our intention is to change minds of the traders who believe in the Weekend Effect.

To begin, we set the workspace, as well as retrive data locally stored in the work space. Immediately next, we want to split the historical SP500 data by the days of the week. Hence, append a new column in the SP500 dataframe, and denote it “Day.”

In [76]:
```
import os
print os.getcwd()
os.chdir("/Users/edwsurewin/Dropbox/Berkeley MA/222/STAT222-SP500")
# or elsewhere at which "daily.csv" is stored
```

```
/Users/edwsurewin/Dropbox/Berkeley MA/222/STAT222-SP500
```

In [77]:
```
import pandas as pd

SP500 = pd.read_csv("daily.csv")
SP500['Day'] = None # create a Day of the Week column
print(SP500.head())
```

```
         Date     Open     High      Low    Close      Volume  Adj
Close   Day
0  2014-02-28  1855.12  1867.92  1847.67  1859.45  3917450000
1859.45  None
1  2014-02-27  1844.90  1854.53  1841.13  1854.29  3547460000
1854.29  None
2  2014-02-26  1845.79  1852.65  1840.66  1845.16  3716730000
1845.16  None
3  2014-02-25  1847.66  1852.91  1840.19  1845.12  3515560000
1845.12  None
4  2014-02-24  1836.78  1858.71  1836.78  1847.61  4014530000
1847.61  None

[5 rows x 8 columns]
```

3.2 Quantity of interest

First of all, we proceed by defining a quantity of interest. Since we are interested in the overnight changes, we are interested in the overnight returns by days of the week. Denote overnight return $R_{today} = \frac{Next-Trading-Day-Opening}{Today's-Adjusted-Close}$.Therefore, during any given ordinary week without holidays (i.e. non-trading days), we have five overnight returns. Next, we defined a function that automates computing the rate of return for any two consecutive trading days. Then, we employed a built-in package in python, which calculates the "day of the week" indicator for each row observation.

Once we had the returns computed together with the indicators, we construct five dynamic lists to store Monday, Tuesday, Wednesday, Thursday, and Friday overnight returns, respectively.

In [78]:
```
# print(SP500.tail()) = 0 to 16146
# len(SP500['Day']) = 16147`

rate = [None]*(len(SP500['Day'])-1)

#create an array and fill in
def compute_rate(SP500, rate):
    for i in range(0, len(SP500['Day'])-1):
        rate[i] = SP500['Open'][i]/SP500['Adj Close'][i+1]
    return rate

rate = compute_rate(SP500, rate)

# first 5 items in open/close rate list
print(rate[0:5])
```

```
[1.0004476106757842, 0.99985909081055302, 1.0003631200138745,
1.0000270619881904, 1.0002886317222601]
```

In [79]:
```
import datetime

# fill out the days of the weeks
# 1: Monday, 2: Tuesday, ..., 5: Friday

for i in range(0, len(SP500['Day'])):
    SP500['Day'][i] = datetime.date(int(SP500['Date'][i][0:4]), int(SP500['Date'][i][5

print(SP500.head())
```

```
         Date     Open     High      Low    Close      Volume  Adj
Close Day
0  2014-02-28  1855.12  1867.92  1847.67  1859.45  3917450000
1859.45   5
1  2014-02-27  1844.90  1854.53  1841.13  1854.29  3547460000
1854.29   4
2  2014-02-26  1845.79  1852.65  1840.66  1845.16  3716730000
1845.16   3
3  2014-02-25  1847.66  1852.91  1840.19  1845.12  3515560000
1845.12   2
4  2014-02-24  1836.78  1858.71  1836.78  1847.61  4014530000
1847.61   1

[5 rows x 8 columns]
```

In [80]:
```
# create dynamic lists to store overnight returns of corresponding days of the week
# control group  Friday, whereas experiment group  any other day(s) of the week

# e.g. a Monday's overnight is defined to be
#      (the quotient of) Tuesday Opening / Monday's Adj Close

Monday = []
Tuesday = []
Wednesday = []
Thursday = []
Friday = []

for i in range(1, len(SP500['Day'])):
    if SP500['Day'][i] == 1:
        Monday.append(rate[i-1])
    else:
        if SP500['Day'][i] == 2:
            Tuesday.append(rate[i-1])
        else:
            if SP500['Day'][i] == 3:
                Wednesday.append(rate[i-1])
            else:
                if SP500['Day'][i] == 4:
                    Thursday.append(rate[i-1])
                else:
                    if SP500['Day'][i] == 5:
                        Friday.append(rate[i-1])
# recent 10 years
Monday = Monday[0:520]
Tuesday = Tuesday[0:520]
Wednesday = Wednesday[0:520]
```

```
Thursday = Thursday[0:520]
Friday = Friday[0:520]

# for exmaple, Friday vs. NonFriday
NonFri = Monday + Tuesday + Wednesday + Thursday
```

3.3 Quantity of Interest Plots

Before moving on, we subset the first 520 elements in each list; therefore, we have roughly the recent 10-year data to analyze upon. The reason of subsetting is that the fundamentals of stock market might have changed substantially over the last decades, and historical data from the past could tell little about the current dynamics in today's market. We want to rule out the noises due to a different financial regulation settings happenend in the past, and want to solely pinpoint what's happened recently.

With the data cleaned and sort, we would like to have a visual impression how the data look like, and whether we could draw any immediate preliminary conclusion from the pictures. We plotted the following 4 pictures, with each of the Friday group and another day of the week. Notice that the Friday group is the control group, and 4 other groups are the treatment groups; in other words, we apply another day of the week and try to observe possiblity of significance improvements.

On the x-axis, we have "chronological number of weeks until March 6th, 2014", when we lastly accessed the public data from Yahoo Faince. On the y-axis, we have the quantitiy of interest being plotted, i.e. the overnight rates of return. The red dots are of the Friday group, the black dots are of the respective treatement groups. We attempt to answer these questions:

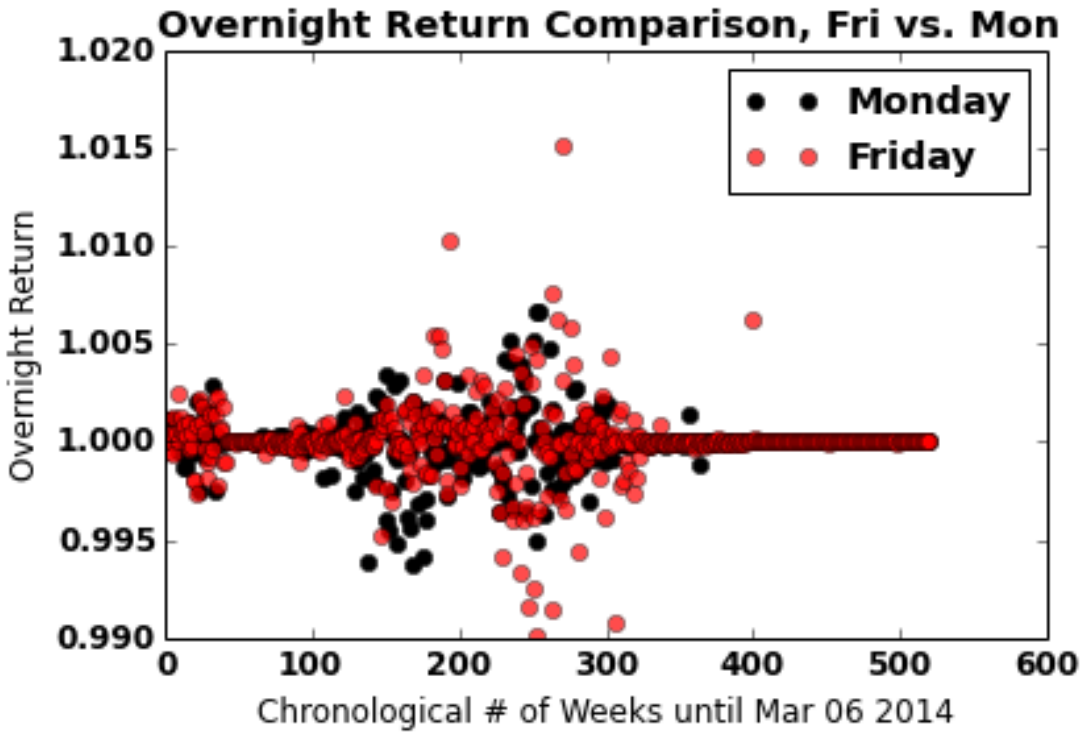
1) Are all the red dots systematically lower than the black dots?
2) If so, which picture shows the most apparent difference?
3) Any patterns?

To answer these questions, we take a closer look at each picture. It seems that the answers for the first two questions are negative, while we can say something about the 3). It appears that we experienced a period of volatilities during Week 100 to Week 300; if we convert the dates back to calendar, we will notice that that was the period we were in the financial meltdown commercing 2007. Hence, the pictures nevertheless showed we caputured the data correctly.

Yet, we are not able to make any statements with regards to the research objective.

In [81]:
```
import numpy
import matplotlib

# overnight return: Monday vs. Friday
# X-axis is timeline in chronological order

plt.plot(Monday, 'ko', label='Monday')
plt.hold(True)
plt.plot(Friday, 'ro', label='Friday', alpha=0.7)
plt.legend(loc=0)
plt.title('Overnight Return Comparison, Fri vs. Mon')
plt.ylabel('Overnight Return')
plt.xlabel('Chronological # of Weeks until Mar 06 2014')
```

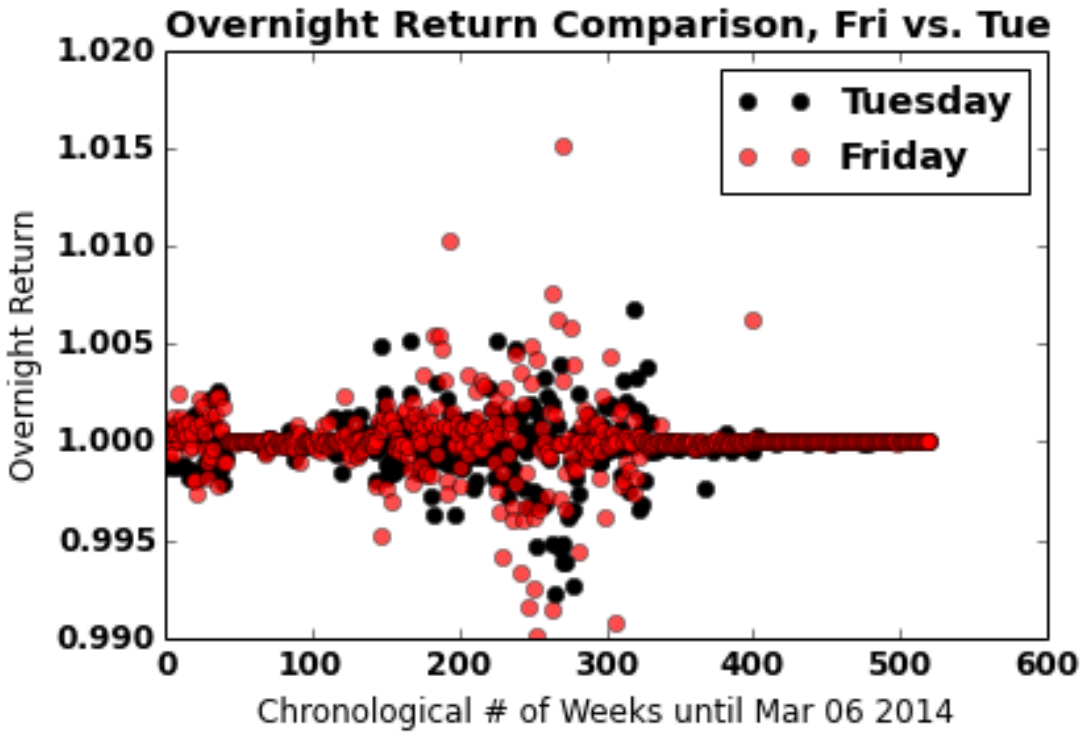
Out [81]:
```
<matplotlib.text.Text at 0x1125562d0>
```



In [82]:
```
# overnight return: Tuesday vs. Friday
# X-axis is timeline in chronological order

plt.plot(Tuesday, 'ko', label='Tuesday')
plt.hold(True)
plt.plot(Friday, 'ro', label='Friday', alpha=0.7)
plt.title('Overnight Return Comparison, Fri vs. Tue')
plt.ylabel('Overnight Return')
plt.xlabel('Chronological # of Weeks until Mar 06 2014')
plt.legend(loc=0)
```

Out [82]:

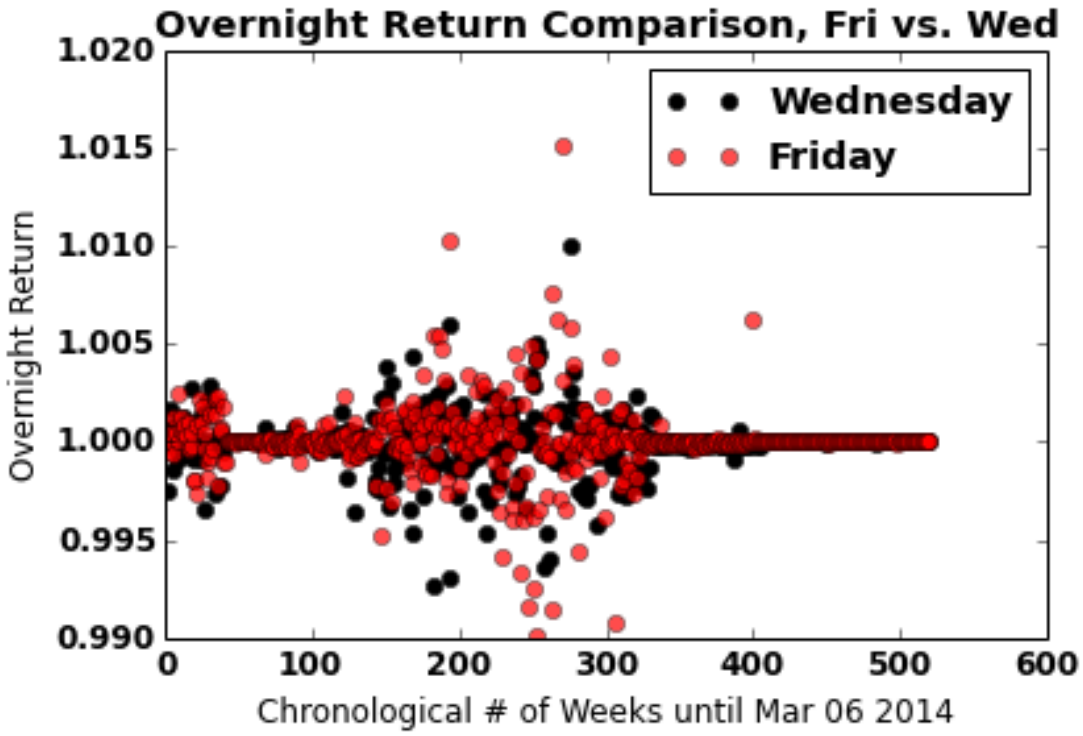
```
<matplotlib.legend.Legend at 0x10e0c3f10>
```



In [83]:
```
# overnight return: Wednesday vs. Friday
# X-axis is timeline in chronological order

plt.plot(Wednesday, 'ko', label='Wednesday')
plt.hold(True)
plt.plot(Friday, 'ro', label='Friday', alpha=0.7)
plt.title('Overnight Return Comparison, Fri vs. Wed')
plt.ylabel('Overnight Return')
plt.xlabel('Chronological # of Weeks until Mar 06 2014')
plt.legend(loc=0)
```

Out [83]:

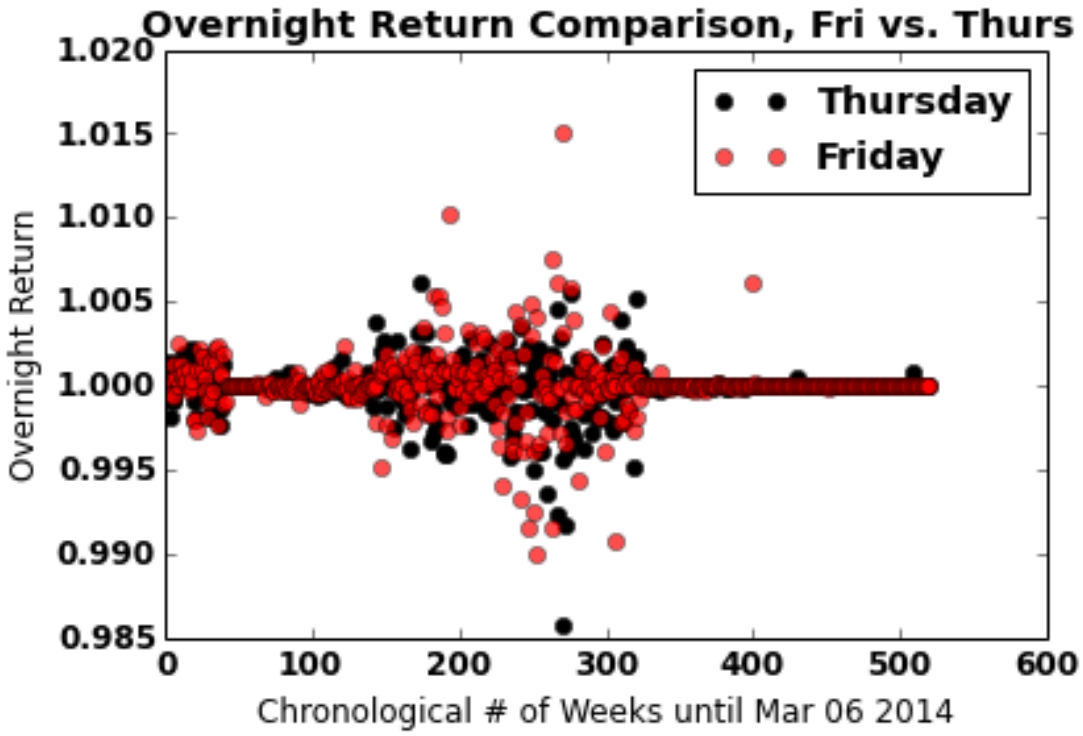
```
<matplotlib.legend.Legend at 0x10e0fd3d0>
```



In [84]:
```
# overnight return: Thursday vs. Friday
# X-axis is timeline in chronological order

plt.plot(Thursday, 'ko', label='Thursday')
plt.hold(True)
plt.plot(Friday, 'ro', label='Friday', alpha=0.7)
plt.title('Overnight Return Comparison, Fri vs. Thurs')
plt.ylabel('Overnight Return')
plt.xlabel('Chronological # of Weeks until Mar 06 2014')
plt.legend(loc=0)
```

Out [84]:

```
<matplotlib.legend.Legend at 0x10e0b2910>
```



Naturally, we are going to empower our research by performing a series of statistical tests, wherever applicable. Notice that most tests have a distinct set of requirements under which the data have to suffice. For the time being, we are willing to assume all required, although that could lead us to an aggressive concludsion.

3.4 Mann–Whitney U Test

More efficient than the T test in non-normal distribution, this test is said to embrace a very general formulation which assumes that:

1. All the observations from both groups are independent of each other,
2. The responses are ordinal (i.e. one can at least say, of any two observations, which is the greater),
3. The distributions of both groups are equal under the null hypothesis, so that the probability of an observation from one population (X) exceeding an observation from the second population (Y) equals the probability of an observation from Y exceeding an observation from X. That is, there is a symmetry between populations with respect to probability of random drawing of a larger observation.
4. Under the alternative hypothesis, the probability of an observation from one population (X) exceeding an observation from the second population (Y) (after exclusion of ties) is not equal to 0.5. The alternative may also be stated in terms of a one-sided test, for example: P(X > Y) + 0.5 P(X = Y) is greater than 0.5.

Thus, null is mean(Friday) = mean(nonFriday), and inequality otherwise.

In [85]:
```
import scipy.stats as stats
u, MW_p_value = stats.mannwhitneyu(Friday, NonFri)
print("two-sample wilcoxon-test p-value is", MW_p_value)

# p_value < 0.01 => alternative hypothesis:
# they don't have the same mean at the 1% significance level
```

```
('two-sample wilcoxon-test p-value is', 0.23750501845496358)
```

3.5 Paired T-test

The test assumes that the differences between pairs are normally distributed; one can use the histogram spreadsheet on that page to check the normality. The null hypothesis is that the mean difference between paired observations is zero. In other words,

Null: mean(Friday) - mean(nonFriday) = 0, and Alternative: Otherwise.

In [86]:
```
AvgNon = [None]*len(Friday)
for i in range(0, len(AvgNon)):
    AvgNon[i] = (Monday[i]+Tuesday[i]+Wednesday[i]+Thursday[i])/4

PT_diff = [None]*len(Friday)
for i in range(0, len(AvgNon)):
    PT_diff[i] = Friday[i] - AvgNon[i]

PT_tstats=((mean(Friday)-mean(AvgNon))-0)/(std(PT_diff)*sqrt(len(Friday)-1))

print("Matched Pair T Test yields t-stat", PT_tstats)
```

```
('Matched Pair T Test yields t-stat', 0.0021826026699113838)
```

In [87]:
```
t_statistic, TSTAT_p_value = stats.ttest_ind(Friday, NonFri)
print("similarly in two-sample t-test, p-value is", TSTAT_p_value)
```

```
('similarly in two-sample t-test, p-value is', 0.16235584887494622)
```

In [88]:
```
print("Mean of Fridays is", mean(Friday), "and mean of NonFris is", mean(NonFri))
```

```
('Mean of Fridays is', 1.0000457911663738, 'and mean of NonFris is',
0.99994432697442182)
```

3.6 The Kruskal-Wallis H-test

This test exams the null hypothesis that the population median of all of the groups are equal. It is a non-parametric version of ANOVA. Therefore,

Null: medium(Friday) - medium(nonFriday) = 0, and Alternative: Otherwise.

In [89]:
```
u, ANOVA_p_value = stats.mstats.kruskalwallis(Friday, NonFri)
# Chi-square degree of freedom 1
print("one-way ANOVA Kruskal Wallis test p-value is", ANOVA_p_value)
```

```
('one-way ANOVA Kruskal Wallis test p-value is', 0.4749895027320612)
```

With a p value smaller than 0.01, we are able to claim that the alternative hypothesis: The groups do not have the same median at the 1% significance level. As a hindsight remark, the Mann–Whitney U Test and the Kruskal-Wallis H-test suffer from a limitation that, the Friday grouop might not be independent from the Non-friday group, especially given the data is time-series.

3.7 The Friedman Test

This test exames the null hypothesis that repeated measurements of the same individuals have the same distribution. However, it additionally requires the groups have are paired and thus have the same size, which our data satisfies.

In [90]:
```
# In order to apply Friedman test, measurements must have equal length, hence,
# ASSUME that all the missing days are of no information

#FridayFM = Friday + [NaN]*63
#MondayFM = Monday + [NaN]*187
#TuesdayFM = Tuesday + [NaN]*1
#WednesdayFM = Wednesday
#ThursdayFM = Thursday + [NaN]*43

FridayFM = Friday
MondayFM = Monday
TuesdayFM = Tuesday
WednesdayFM = Wednesday
ThursdayFM = Thursday

print(len(FridayFM), len(MondayFM), len(TuesdayFM), len(WednesdayFM), len(ThursdayFM))
```

```
(520, 520, 520, 520, 520)
```

In [91]:
```
u, fridman_p_value=stats.mstats.friedmanchisquare(FridayFM, MondayFM, TuesdayFM, Wedne
print("five-measure Friedman test p-value is", fridman_p_value)

# p_value < 0.01 => alternative hypothesis:
# some of the measurements were taken differently at the 1% significance level
```

```
('five-measure Friedman test p-value is', 6.6210477396613121e-07)
```

In [92]:
```
u, p_value = stats.mstats.friedmanchisquare(MondayFM, TuesdayFM, WednesdayFM, Thursday
print("four-measure (excluding Friday) Friedman test p-value is", p_value)

# p_value > 0.01 => null hypothesis:
# null is not rejected at the 1% significance level, measurements consistant
```

```
('four-measure (excluding Friday) Friedman test p-value is',
3.0375578037409941e-07)
```

The limitation of Friedman Test is that, due to the assumption that the test statistic has a chi squared distribution, the p value is only reliable for n greater than 10 and more than 6 repeated measurements. However, we only as well as could at most have 4 repeated measurements.

3.8 Statistical Tests Result Table

In [93]:
```
# put results in a table

data = {'Test Name': ['Wilcoxon rank', 'Matched Pair T', 'Kruskal-Wallis', 'T Statisti
        'Requirements': ['independent, dist equal under null', 'paired up, normal', 'o
        'P Value': [MW_p_value, PT_tstats, ANOVA_p_value, TSTAT_p_value, fridman_p_val
        'Signif': [">.05", "<.01",">.05", ">.05", "<.01"]}
football = pd.DataFrame(data, columns=['Test Name', 'Requirements', 'P Value', 'Signif
print (football)
```

```
         Test Name                          Requirements   P Value Signif
0   Wilcoxon rank  independent, dist equal under null  0.237505   >.05
1  Matched Pair T                   paired up, normal  0.002183   <.01
2  Kruskal-Wallis          one-way ANOVA, iid, normal  0.474990   >.05
3     T Statistic         student"s t-tests, dof > 30  0.162356   >.05
4         Fridman         repeated measure same dist?  0.000001   <.01

[5 rows x 4 columns]
```

3.9 Histogram

Among all the tests above, only two tests came up postivie at the 5% signficance level, indicating some confidence that the Friday returns might have a different distribution pattern comparing to the rest of the groups. Again, this observation is drawn under the assumptions that all the required premises were met, unique to each individual test.

To better visually assess the how Friday group behaves in comparison to the other groups, we plotted the histogram below. From which, we are able to make reference on mean, medium and mode of each group. The historgrams below are standardized, between the Friday group and the non-friday group.

Looking at the histogram of Friday, we would attempt to think that Friday in general has a smaller expected return among the days of a week. However, when we explicitly computed out the mean of Friday group, and compared that value of the non-friday group, we were bewildered that Friday group actually had a larger expected return - or mean value - however, the histogram suggested that the mode of Friday is smaller.

Then, we were confident to think that the Friday group is relatively skewed to the left, with a heavier tail at the high-end values. That perhaps was the reason that Friday group had a larger mean. We wish we could reproduce the Friday groups many times as desired, and anlyze its behaviors in a larger scale.

Hence, we consider Bootstrapping methods thereafter.

In [94]:
```
# By CLT, mean is asymptotically normal with var = /sigma^{2} over n
z = (mean(Friday)-1)/sqrt(var(Friday)/len(Friday))

# Null:    mean(Friday) = 1
# Alter:   mean(Friday) != 1

print("z-score is", z)
```

```
('z-score is', 0.57707808184650711)
```

In [95]:
```
# By CLT, mean is asymptotically normal with var = /sigma^{2} over n
z= (mean(Friday+NonFri)-1)/sqrt(var(Friday+NonFri)/len(Friday+NonFri))

# Null:    mean(weekdays) = 1
# Alter:   mean(weekdays) != 1

print("z-score is", z)
```

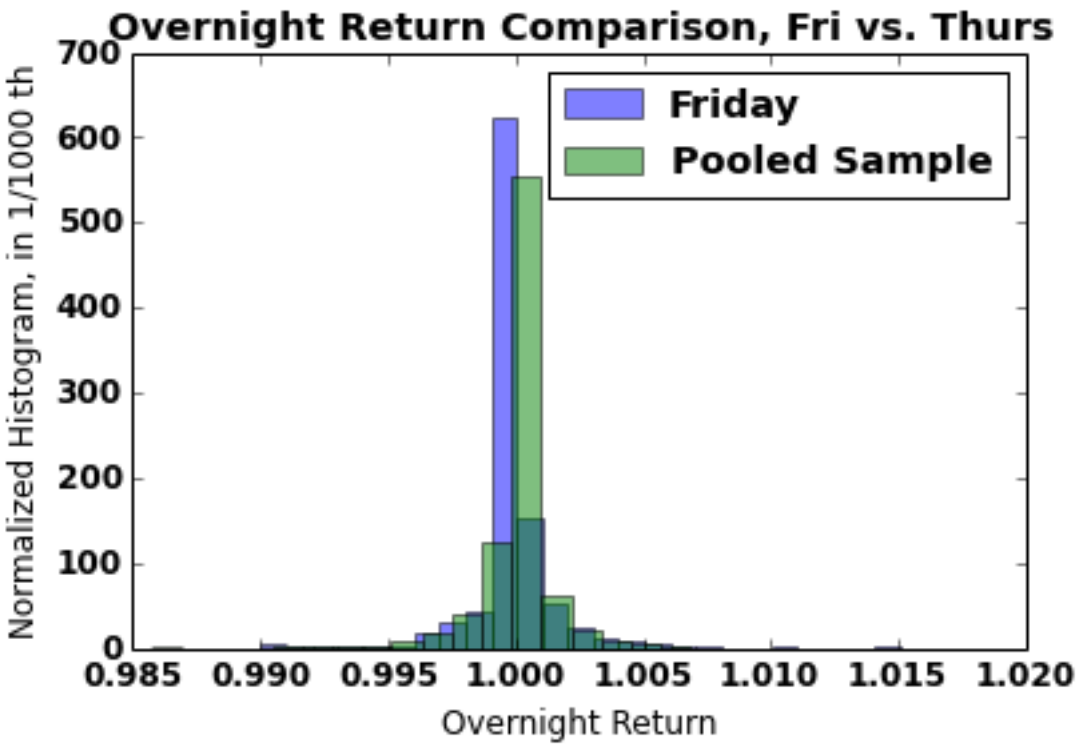
```
('z-score is', -1.2183461996897837)
```

In [96]:
```
# multi normalized histogram of Friday and NonFriday

#setbins = [-0.01, -0.008, -0.006, -0.004, -0.002, 0, 0.002, 0.004, 0.006, 0.008, 0.01
plt.hist(Friday, bins=25, label="Friday",  normed=True,  alpha=.5)
plt.hist(NonFri+Friday, bins=25, label="Pooled Sample",  normed=True, alpha=.5)
```

```
#plt.xlim((.9995,1.0005))
plt.title('Overnight Return Comparison, Fri vs. Thurs')
plt.ylabel('Normalized Histogram, in 1/1000 th')
plt.xlabel('Overnight Return')
plt.legend(loc=0)
```

Out [96]:

```
<matplotlib.legend.Legend at 0x110cdf3d0>
```



In [97]:

```
print("Mean of Fridays is", mean(Friday), "and Mean of Pooled is", mean(NonFri+Friday))
```

```
('Mean of Fridays is', 1.0000457911663738, 'and Mean of Pooled is',
0.99996461981281226)
```

In [98]:

```
print("Mean of Pooled is", mean(NonFri+Friday))
```

```
('Mean of Pooled is', 0.99996461981281226)
```

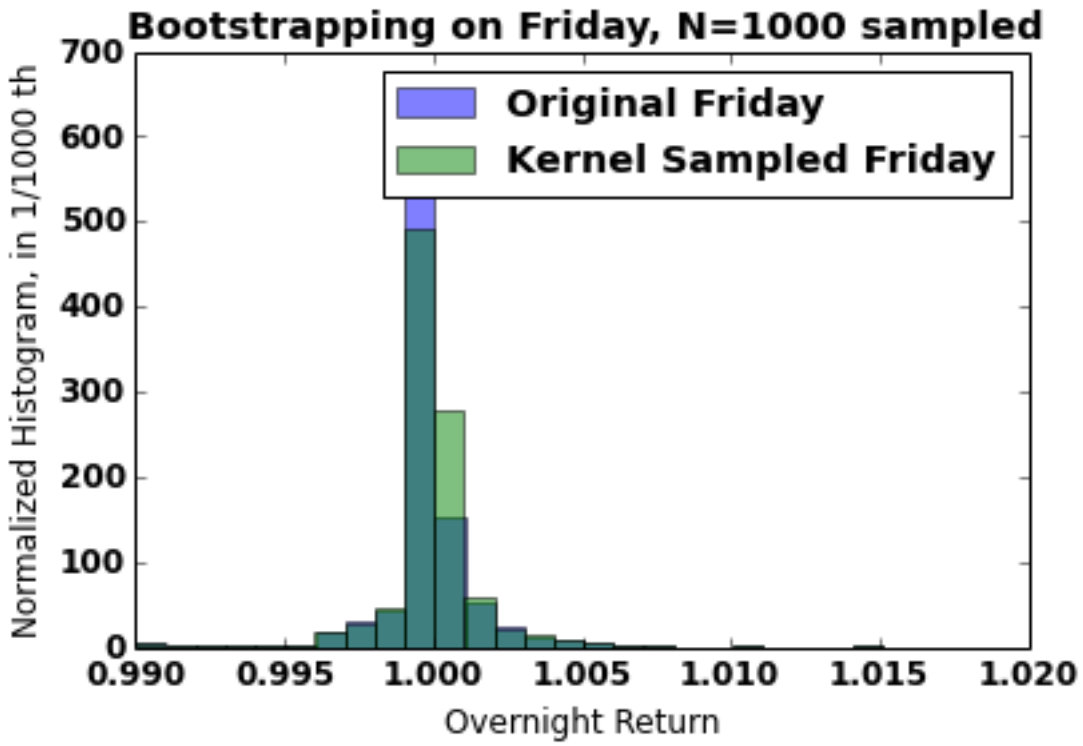
3.10 Kernel Estimation

Before bootstrapping, we would need to train our kernels such that they can catch the raw data's distributions closely. In other words, we want them to be able to reproduce the raw data to the best capacity. Given the underlying distributions were highly concentrated around 1, we were not able to choose a Gaussian kernel but had to try a spike-like "tophat" kernel.

We start off by simulating the raw data set, and compare-and-constrat if the reproduced data collide with the original data well. We scratched new histograms, and observed that the shapes of two histograms almost perfectly resembled each other, as well as the colors blended.

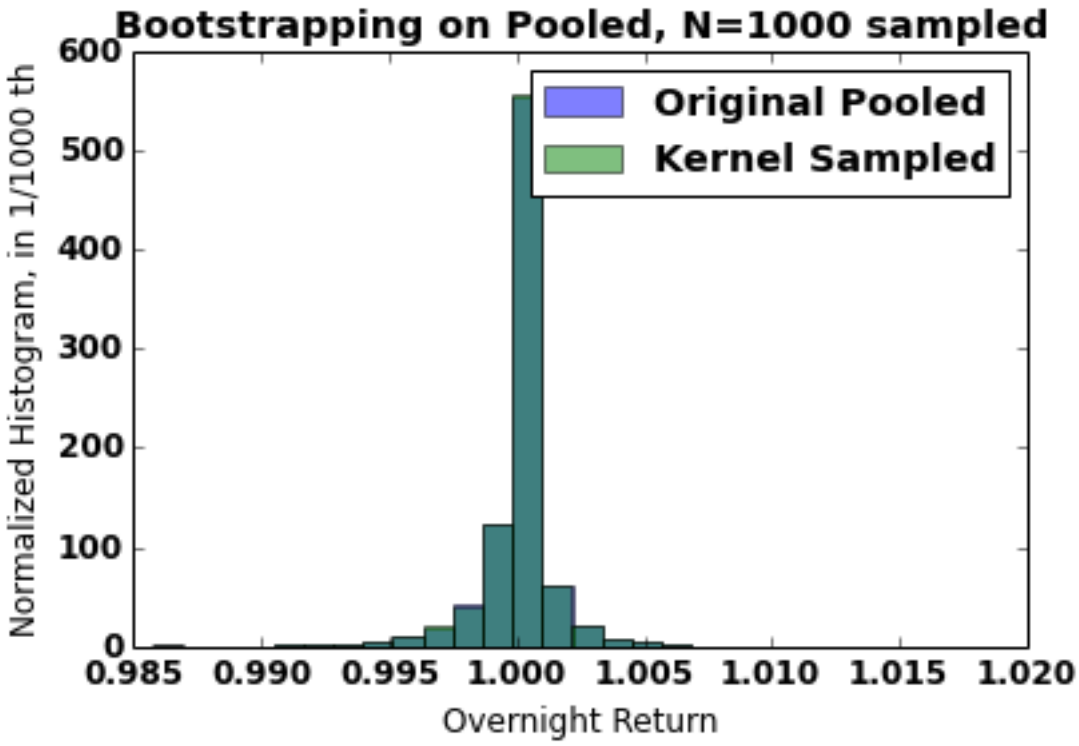
In [99]:
```
from sklearn.neighbors import KernelDensity
kde = KernelDensity(bandwidth=.001, algorithm='auto', kernel='tophat', metric='euclide
# kernel estimation with tophat as spike-like histogram

# train the model
kde.fit(Friday)

# 100 sample
ker100 = kde.sample(n_samples=2000, random_state=None)

sample = []
for i in range(0, len(ker100)):
    sample = sample + ker100.tolist()[0]

plt.hist(Friday, bins=25, label="Original Friday", normed=True, alpha=.5)
plt.hist(sample, bins=25, normed=True, label="Kernel Sampled Friday", alpha=.5)
# settled down to Friday's

plt.title('Bootstrapping on Friday, N=1000 sampled')
plt.ylabel('Normalized Histogram, in 1/1000 th')
plt.xlabel('Overnight Return')
plt.legend(loc=0)

# kernel estiamted collapse w/ original, so estimation worked
```

Out [99]:
```
<matplotlib.legend.Legend at 0x10e08e350>
```



In [100]:
```
from sklearn.neighbors import KernelDensity
kde = KernelDensity(bandwidth=.001, algorithm='auto', kernel='tophat', metric='euclide
# kernel estimation with tophat as spike-like histogram

# train the model
kde.fit(Friday+NonFri)

# 100 sample
ker100 = kde.sample(n_samples=1000, random_state=None)

sample = []
for i in range(0, len(ker100)):
    sample = sample + ker100.tolist()[0]

plt.hist(Friday+NonFri, bins=25, label="Original Pooled", normed=True, alpha=.5)
plt.hist(sample, bins=25, normed=True, label="Kernel Sampled", alpha=.5)   # settled do
plt.title('Bootstrapping on Pooled, N=1000 sampled')
plt.ylabel('Normalized Histogram, in 1/1000 th')
plt.xlabel('Overnight Return')
plt.legend(loc=0)

# kernel estiamted collapse w/ original, so estimation worked
```

Out [100]:
```
<matplotlib.legend.Legend at 0x10e09f350>
```



3.11 Bootstrap

Then, we are able to bootstrap. We defined the bootstrap function to be the sample mean. Therefore, for each boot, we compute and store the mean of bootstrapped sample. We first plotted the bootstrapped sample means of the Friday group and the population group.

The Central Limit Theorem dictates the histogram of boostrapped means to be normal, which they are.

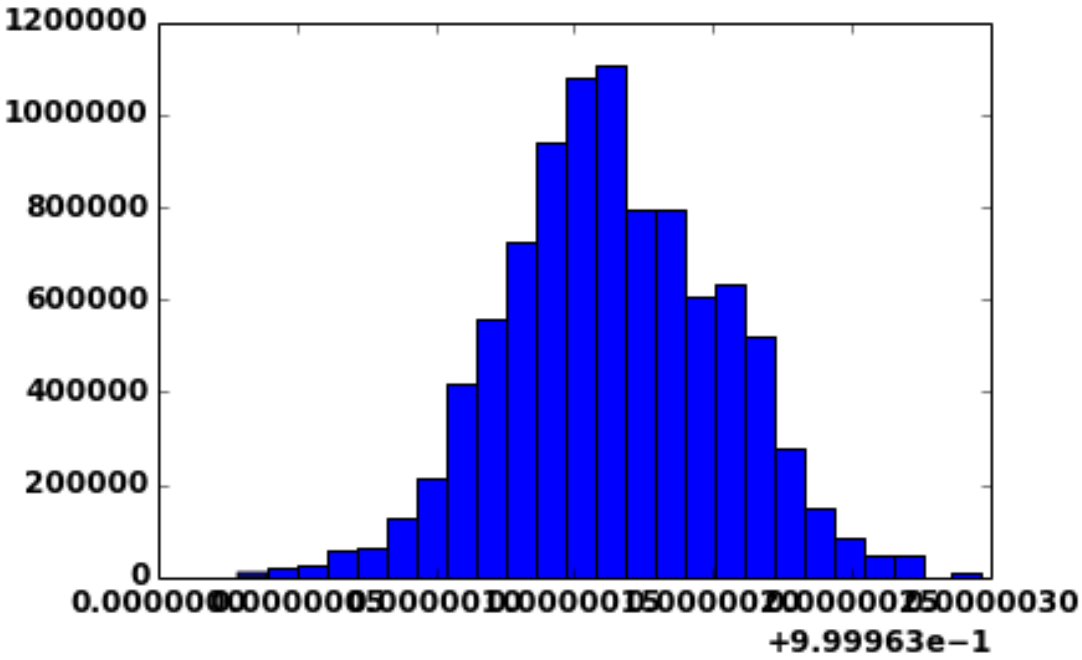
In [101]:
```
MonteCarloBoot = kde.sample(n_samples=1000, random_state=None)
```

In [102]:
```
BootMean = [None]*len(MonteCarloBoot)
def bootMean(MonteCarloBoot, BootMean):
    for i in range(0, len(MonteCarloBoot)):
        BootMean[i] = MonteCarloBoot[i].mean()
    return(BootMean)

BootMean = bootMean(MonteCarloBoot, BootMean)

# Bootstrapping with T = mean(obs)
# v_boot = 1/B * B_sum(T - mean(T))^2

Boot_var = std(BootMean)

print("Monte Carlo Bootstrapping with mean", mean(BootMean), "variance", Boot_var)
```

```
('Monte Carlo Bootstrapping with mean', 0.99996463454836415,
'variance', 3.9317659582286139e-07)
```

In [103]:
```
plt.hist(BootMean, bins=25,normed=True)
```

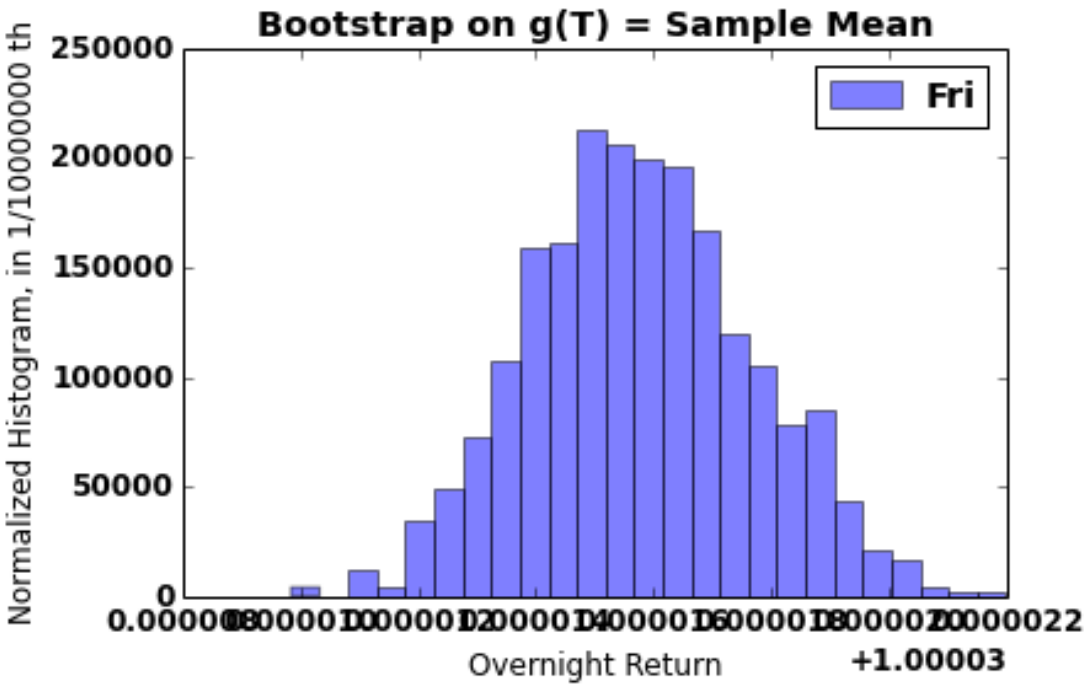
Out [103]:
```
(array([    9319.26375747,    18638.52751494,    27957.79127242,
          55915.58254483,    65234.8463698 ,   130469.69260461,
         214343.06642187,   419366.86908626,   559155.82544835,
         726902.57308285,   941245.63950472,  1081034.5958668 ,
        1108992.38828663,   792137.41938516,   792137.41938516,
         605752.14423571,   633709.93550813,   521878.77041846,
         279577.91272417,   149108.22011956,    83873.37390403,
          46596.31878736,    46596.31878736,        0.        ,
           9319.26375747]),
 array([ 0.99996329,  0.9999634 ,  0.9999635 ,  0.99996361,
0.99996372,
        0.99996382,  0.99996393,  0.99996404,  0.99996415,
0.99996425,
        0.99996436,  0.99996447,  0.99996458,  0.99996468,
0.99996479,
        0.9999649 ,  0.999965  ,  0.99996511,  0.99996522,
0.99996533,
        0.99996543,  0.99996554,  0.99996565,  0.99996576,
0.99996586,
        0.99996597]),
 <a list of 25 Patch objects>)
```



In [104]:
```
from sklearn.neighbors import KernelDensity
# kernel estimation with tophat as spike-like histogram

############################ Friday ############################

kdeF = KernelDensity(bandwidth=.001, algorithm='auto', kernel='tophat', metric='euclid

kdeF.fit(Friday)
FriBoot = kdeF.sample(n_samples=1000, random_state=None)

FriBootMean = [None]*len(FriBoot)
FriBootMean = bootMean(FriBoot, FriBootMean)

############################ Thursday ############################

kdeR = KernelDensity(bandwidth=.001, algorithm='auto', kernel='tophat', metric='euclid

kdeR.fit(Thursday)
ThrBoot = kdeR.sample(n_samples=1000, random_state=None)

ThrBootMean = [None]*len(ThrBoot)
ThrBootMean = bootMean(ThrBoot, ThrBootMean)


############################ Wednesday ############################


kdeW = KernelDensity(bandwidth=.001, algorithm='auto', kernel='tophat', metric='euclid

kdeW.fit(Wednesday)
WedBoot = kdeW.sample(n_samples=1000, random_state=None)

WedBootMean = [None]*len(WedBoot)
WedBootMean = bootMean(WedBoot, WedBootMean)

############################ Tuesday ############################
```

```
kdeT = KernelDensity(bandwidth=.001, algorithm='auto', kernel='tophat', metric='euclid

kdeT.fit(Wednesday)
TueBoot = kdeT.sample(n_samples=1000, random_state=None)

TueBootMean = [None]*len(TueBoot)
TueBootMean = bootMean(TueBoot, TueBootMean)

############################ Monday ############################


kdeM = KernelDensity(bandwidth=.001, algorithm='auto', kernel='tophat', metric='euclid

kdeM.fit(Monday)
MonBoot = kdeM.sample(n_samples=1000, random_state=None)

MonBootMean = [None]*len(MonBoot)
MonBootMean = bootMean(MonBoot, MonBootMean)


# plt.hist(FriBootMean, bins=25,normed=True)
```

In [105]:
```
plt.hist(FriBootMean, 25, label="Fri", alpha=.5, normed=True)
#plt.hist(ThrBootMean, 25, label="Thr", alpha=.5, normed=True)
#plt.hist(WedBootMean, 25, label="Wed", alpha=.5, normed=True)
#plt.hist(TueBootMean, 25, label="Tue", alpha=.5, normed=True)
#plt.hist(MonBootMean, 25, label="Mon", alpha=.5,normed=True)
#plt.hist(BootMean, 25, label="Pooled", alpha=.5, normed=True)
plt.legend(loc=0)
plt.ticklabel_format(style='plain', axis='x', scilimits=(0,0))
plt.title('Bootstrap on g(T) = Sample Mean')
plt.ylabel('Normalized Histogram, in 1/10000000 th')
plt.xlabel('Overnight Return')
#plt.ylim((0,300000))
show()
```



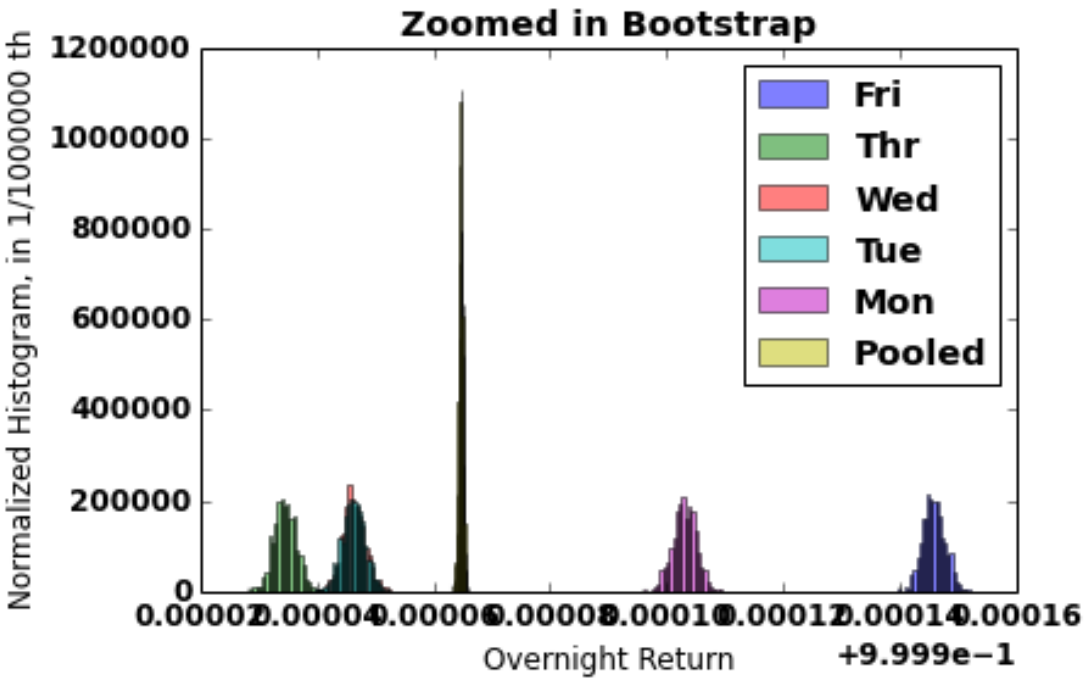
3.12 Bootstrap Results

Monte Carlo Simulation indicates that Friday has different distribution pattern from other days of the week combined. Surprisingly, the Friday group came up the most positive.

To better understand the following plots, notice that

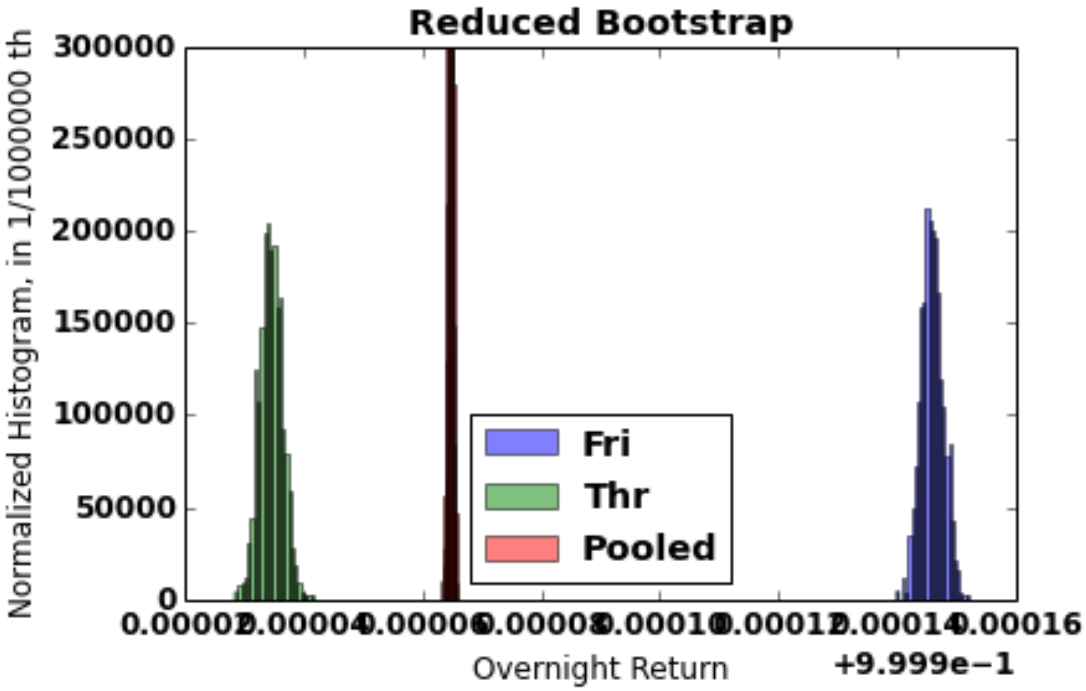
1) There are six histograms being plotted on the same picture, but two of which overlapped.
2) The tallest and skinest one is the boot mean of the population. As it had five times as many size of the others, its standard error shrinks to by one over square root of five. Hence, it had a much smaller variability that results a different shape.
3) Among five "day of the week" individual boots, Friday had the highest boot means. In other words, Fridays over the last ten years were probably doing better than the rest weekdays.
4) To better see the comparison, notice a cleaner plot underneath the summary plot.

In [106]:
```
plt.hist(FriBootMean, 25, label="Fri", alpha=.5, normed=True)
plt.hist(ThrBootMean, 25, label="Thr",  alpha=.5, normed=True)
plt.hist(WedBootMean, 25, label="Wed",  alpha=.5, normed=True)
plt.hist(TueBootMean, 25, label="Tue",  alpha=.5, normed=True)
plt.hist(MonBootMean, 25, label="Mon",  alpha=0.5,normed=True)
plt.hist(BootMean, 25, label="Pooled",  alpha=.5, normed=True)
plt.title('Zoomed in Bootstrap')
plt.ylabel('Normalized Histogram, in 1/1000000 th')
plt.xlabel('Overnight Return')
plt.legend(loc=0)
#plt.ylim((0,300000))
show()
```



In [107]:
```
plt.hist(FriBootMean, 25, label="Fri", alpha=.5, normed=True)
plt.hist(ThrBootMean, 25, label="Thr", alpha=.5, normed=True)
#plt.hist(WedBootMean, 25, label="Wed", alpha=.5, normed=True)
#plt.hist(TueBootMean, 25, label="Tue", alpha=.5, normed=True)
#plt.hist(MonBootMean, 25, label="Mon", alpha=0.5,normed=True)
```

```
plt.hist(BootMean, 25, label="Pooled", alpha=.5, normed=True)
plt.title('Reduced Bootstrap')
plt.ylabel('Normalized Histogram, in 1/1000000 th')
plt.xlabel('Overnight Return')
plt.legend(loc=0)
plt.ylim((0,300000))
show()
```



4 Conclusions and Limitations

By now via conventional statistical methods, our group has shown that Friday evenings over the past decade were not necessarily resulting a signficiant lower return. However, we acknowledge there are many aggressive assumptions - and perhaps, loopholes - in our research.

The main difficulty of the research was that we utilized conventional statistical methods on a time-series finanical data. Oftentimes, as seen eariler, the methods require each comparison group to possess certain properties, such as normality, independence among each observations, IID, and pairwise movements. However, we perhaps shall assume none of those.

The bootstrapping method was distribution free, meaning that if the data were correctly accessed, the bootstrap summary yet predicts some valuable information of the data structure, regardless correlation within the data set. It is our hope that our audience, at this point, are somewhat convinced that the Weekend Effect might be just a myth.