Assignment 1 - Financial Econometrics

Diogo Franquinho and Max Meijer

February 21, 2021

1 Assignment 1: GARCH modelling

Deadline: Sunday 21 February, 23.59.

Your notebook should run withouth errors when executed with `Run All`. Please submit your answers via Canvas.

Name	Student ID	Email

****Hand in the following:**** * Your notebook. N.B. **click on** `Kernel`**, then** `Restart & Run All` before submitting, see notes. * A (printed) pdf version of your notebook. Tip: you can use `nbconvert` (user guide) for this, or simply print the webpage to pdf.

****NOTES****: * The assignment is a partial stand-in for a final examination, so the usual rules regarding plagiarism and fraud apply, with all attendant consequences. Code found on the internet or elsewhere is not acceptable as a solution. * Before submitting your work, **click on** `Kernel`**, then** `Restart & Run All` and verify that your notebook produces the desired results and does not error. * If your function uses random numbers, then set the seed to 0 before calling it. This makes it much easier to grade the assignments (at least as long as the answer is correct).

Declaration of Originality:

By submitting these answers, we declare that 1. We have read and understood the notes above. 2. These solutions are solely our own work. 3. We have not made these solutions available to any other student.

1.1 Introduction

This is an assignment about the selection, estimation, testing and Monte Carlo simulation of GARCH models for daily stock index returns.

[1]:
```
import numpy as np
import pandas as pd
```

```
import pandas_datareader.data as web
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
import statsmodels.graphics.tsaplots as tsaplots
from statsmodels.tsa.arima_model import ARMA
from statsmodels.compat import lzip
from scipy import stats
import seaborn as sns
!pip install git+git://github.com/khrapovs/skewstudent
from skewstudent import SkewStudent
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm
```

```
Collecting git+git://github.com/khrapovs/skewstudent
  Cloning git://github.com/khrapovs/skewstudent to /tmp/pip-req-build-nt6f9k8u
  Running command git clone -q git://github.com/khrapovs/skewstudent /tmp/pip-
req-build-nt6f9k8u
Requirement already satisfied (use --upgrade to upgrade): skewstudent==1.0 from
git+git://github.com/khrapovs/skewstudent in /usr/local/lib/python3.6/dist-
packages
Building wheels for collected packages: skewstudent
  Building wheel for skewstudent (setup.py) ... done
  Created wheel for skewstudent: filename=skewstudent-1.0-cp36-none-any.whl
size=5903
sha256=e51916bd5dfc8ae134a6de8f6e5df503e712ab649ced2ac5de4768d6e03d6659
  Stored in directory: /tmp/pip-ephem-wheel-cache-
o_unk2qk/wheels/9f/64/de/0392851dcd9a00a7651659831866c7909499094f813c4224e4
Successfully built skewstudent
```

[2]:
```
import pandas_datareader.data as web
```

FTSE data First download data on the FTSE 100 index for the period January 1, 1998 – January 29, 2021, from Yahoo Finance using `pandas-datareader` (example and function reference).

Hint: use `'%5EFTSE%3FP%3DFTSE'` as ticker symbol. Using `'^FTSE'` doesn't work, likely this is because of some labeling/referencing issue within Yahoo Finance.

[3]:
```
df = pd.DataFrame()
start = pd.datetime(1998,1,1)
end = pd.datetime(2021,1,29)
FTSE100 = web.DataReader('%5EFTSE%3FP%3DFTSE','yahoo',start,end)
# We use the closing price of each day
df['ftse'] = FTSE100['Adj Close']
```

```python
df['log_ftse'] = np.log(FTSE100['Adj Close'])
df.dropna(inplace=True)
df['perc_log_return'] = (df.log_ftse - df.log_ftse.shift(1))*100
df
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: FutureWarning:
The pandas.datetime class is deprecated and will be removed from pandas in a
future version. Import from datetime instead.

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3: FutureWarning:
The pandas.datetime class is deprecated and will be removed from pandas in a
future version. Import from datetime instead.
  This is separate from the ipykernel package so we can avoid doing imports
until
```

[3]:

```
                   ftse  log_ftse  perc_log_return
Date
1998-01-02  5193.500000  8.555163              NaN
1998-01-05  5262.500000  8.568361         1.319836
1998-01-06  5264.399902  8.568722         0.036096
1998-01-07  5224.100098  8.561038        -0.768461
1998-01-08  5237.100098  8.563523         0.248538
...                 ...       ...              ...
2021-01-25  6638.899902  8.800702        -0.842966
2021-01-26  6654.000000  8.802973         0.227191
2021-01-27  6567.399902  8.789873        -1.310018
2021-01-28  6526.200195  8.783580        -0.629313
2021-01-29  6407.500000  8.765224        -1.835570

[5816 rows x 3 columns]
```

1.1.1 GARCH in Python

Make sure that the arch package is installed before importing it. It holds functionality to estimate GARCH models.

Uncomment the next line to install. Note: ! executes shell commands.

[4]:

```
!pip install arch
```

```
Requirement already satisfied: arch in /usr/local/lib/python3.6/dist-packages
(4.16.1)
Requirement already satisfied: numpy>=1.14 in /usr/local/lib/python3.6/dist-
packages (from arch) (1.19.5)
Requirement already satisfied: property-cached>=1.6.4 in
/usr/local/lib/python3.6/dist-packages (from arch) (1.6.4)
Requirement already satisfied: pandas>=0.23 in /usr/local/lib/python3.6/dist-
```

```
packages (from arch) (1.1.5)
Requirement already satisfied: scipy>=1.2.3 in /usr/local/lib/python3.6/dist-
packages (from arch) (1.4.1)
Requirement already satisfied: statsmodels>=0.10 in
/usr/local/lib/python3.6/dist-packages (from arch) (0.10.2)
Requirement already satisfied: cython>=0.29.14 in /usr/local/lib/python3.6/dist-
packages (from arch) (0.29.21)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-
packages (from pandas>=0.23->arch) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in
/usr/local/lib/python3.6/dist-packages (from pandas>=0.23->arch) (2.8.1)
Requirement already satisfied: patsy>=0.4.0 in /usr/local/lib/python3.6/dist-
packages (from statsmodels>=0.10->arch) (0.5.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-
packages (from python-dateutil>=2.7.3->pandas>=0.23->arch) (1.15.0)
```

[5]:
```
from arch import arch_model
```

1.2 Questions

1. Use the theory explained in the book and the lecture notes to select, estimate and test an empirical ARMA-GARCH model for the daily log-returns (in percentages: $r_t = 100 \cdot \Delta \log P_t$). Report on your findings, paying attention to the following elements:
    1. Testing for autocorrelation in the returns: is there any need for ARMA terms, and if so, what would be useful order p and q to start with?
    2. Testing for volatility clustering: what type of ARCH or GARCH model would be suitable?
    3. Estimation and testing of a selected ARMA-GARCH model. Do the standardised residuals behave as homoskedastic white noise, according to the available tests? Have you taken appropriate account of possible asymmetry in the news impact curve? Is the standard normal distribution appropriate for the standardised residuals, or would it be better to use another distribution?
    4. If any of the tests under 1.C indicate room for improvement, then adapt or extend the model, and check wether the revised model passes the tests.
    5. Make a plot of the estimated volatility from your final model, and also make a graph of the estimated news impact curve.
2. Use the model estimated under 1, and the resulting residual $\hat{a}_T$ and estimated volatility $\hat{\sigma}_T$ for **January 29, 2021**, to simulate the conditional distribution of the index return over the following 21 trading days (about a month). You will have to simulate the daily returns $r_{T+1}, \ldots, r_{T+21}$, to obtain a simulation of total monthly return $r[21]_{T+21} = \sum_{t=1}^{21} r_{T+t}$. The function `simulation` provides a starting point for such an analysis, but you will have to complete the program with the information from your empirical analysis in the first part.

After completing the program, analyse the outcomes and report on your findings, paying attention to the following:

1. What is the standard deviation of the monthly return? Is this what you would expect from the average daily standard deviation of the returns over the last 21 years? If not, can you give an explanation for the difference? [*Note*: an approximation of the n-period (average) volatility is $\sqrt{n}$ times the 1-period (average) volatility; this approximation is based on the assumption of uncorrelated returns.]

2. What is the shape of the distribution of the monthly returns? Does it display skewness and/or excess kurtosis? Can you explain these findings from the model you have used for the simulations?

3. It may be of interest to experiment a little with the effect of different parameter values on the outcomes under A. and B.; for example, you could compare the results with and without asymmetric (leverage) effects. Also you could choose another month (corresponding to other r_T, $\hat{a}_T$ and $\hat{\sigma}_T$ to start the simulation), and compare the result.

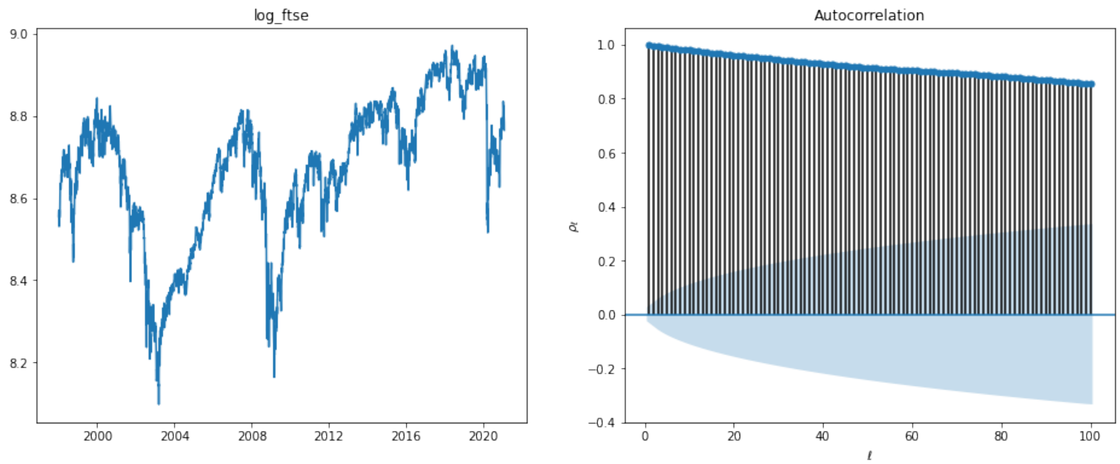
1.2.1 1.a)

[6]:
```
df.describe()
```

[6]:

	ftse	log_ftse	perc_log_return
count	5816.000000	5816.000000	5815.000000
mean	5907.710418	8.669603	0.003612
std	967.816598	0.173251	1.204978
min	3287.000000	8.097731	-11.511706
25%	5274.650024	8.570668	-0.557872
50%	5980.250000	8.696218	0.042179
75%	6578.174805	8.791513	0.605554
max	7877.500000	8.971766	9.384244

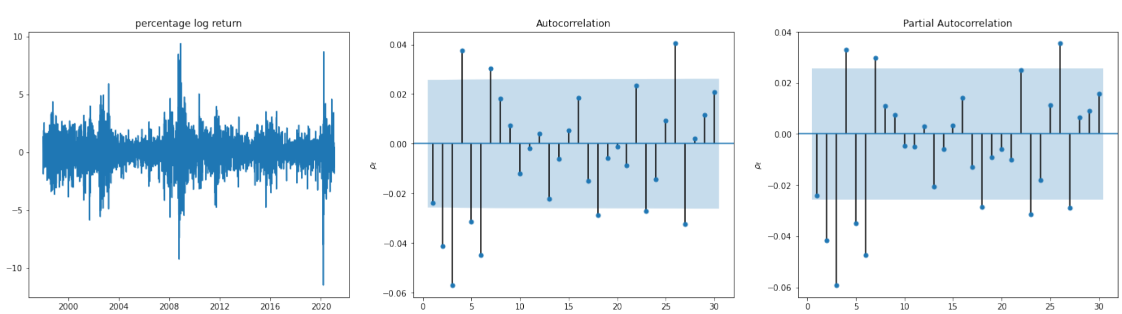
[7]:
```
fig,ax = plt.subplots(nrows=1, ncols=2, figsize=(16,6));
ax[0].plot(df.log_ftse);
ax[0].set_title('log_ftse')
tsaplots.plot_acf(df.log_ftse, lags=100, ax = ax[1], zero=False)
ax[1].set_ylabel(r'$\rho_\ell$')
ax[1].set_xlabel(r'$\ell$');
```



In the graph of the log of the FTSE we see that it stays away from its mean for long periods of time (several years), so the process is not stationary.

If we look at the autocorrelation function then we see that it decays very slowly. This is indicative of it being a nonstationary process. We can apply differencing in the hope of obtaining a stationary process that way (this will work if the process from the graph above is indeed I(1)).

[8]:
```
fig,ax = plt.subplots(nrows=1, ncols=3, figsize=(24,6));
ax[0].plot(df.perc_log_return);
ax[0].set_title('percentage log return')
tsaplots.plot_acf(df.perc_log_return.dropna(), lags=30, ax = ax[1], zero=False)
ax[1].set_ylabel(r'$\rho_\ell$')
ax[1].set_xlabel(r'$\ell$');
tsaplots.plot_pacf(df.perc_log_return.dropna(), lags=30, ax = ax[2], zero=False)
ax[2].set_ylabel(r'$\rho_\ell$')
ax[2].set_xlabel(r'$\ell$');
```



The graph of the percentage log return seems like that of a stationary process (possibly with volatility clustering) because it seems to reverts back to its mean. Both the autocorrelation function and the partial autocorrelation function do not seem to start being zero from some time on, so an AR(p) model or an MA(q) model is unlikely to suffice. Therefore, we look into using an ARMA(p,q) model. For the parameters p and q, it makes sense to first look at small values of these parameters so that we can avoid any overfitting that would result from adding too many parameters. Therefore, setting p = 1 and q = 1 is a good way to start.

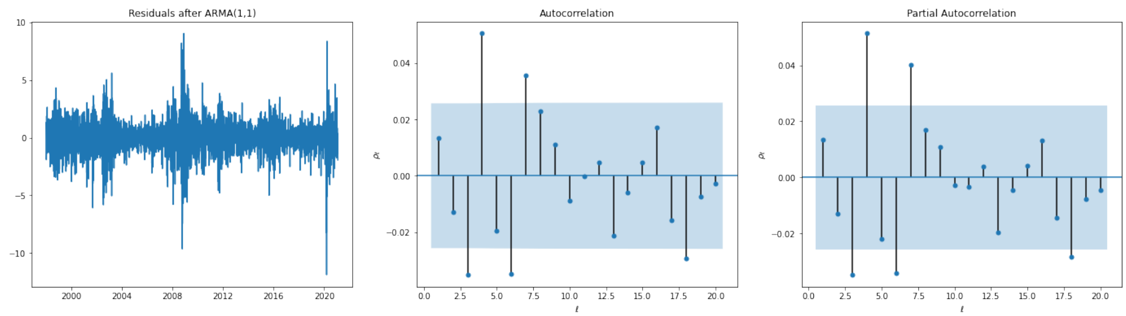
[9]:
```
p = 1
q = 1
arma11 = ARMA(df.perc_log_return.dropna(), order=(p,q)) # Give the (p,q) order␣
↪of the ARMA(p,q) model
results = arma11.fit()
print(results.summary2())
results.params
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

```
                            Results: ARMA
=====================================================================
Model:              ARMA             BIC:                 18683.3797
Dependent Variable: perc_log_return  Log-Likelihood:      -9324.4
Date:               2021-02-21 21:56 Scale:               1.0000
No. Observations:   5815             Method:              css-mle
Df Model:           3                Sample:              0
Df Residuals:       5812                                  5
Converged:          1.0000           S.D. of innovations: 1.203
No. Iterations:     16.0000          HQIC:                18665.984
AIC:                18656.7069
---------------------------------------------------------------------
                       Coef.  Std.Err.    t    P>|t|   [0.025  0.975]
---------------------------------------------------------------------
const                  0.0037   0.0133  0.2767 0.7820 -0.0225  0.0298
ar.L1.perc_log_return  0.7342   0.0933  7.8663 0.0000  0.5512  0.9171
ma.L1.perc_log_return -0.7751   0.0869 -8.9247 0.0000 -0.9453 -0.6049
-------------------------------------------------------------------------------
                Real            Imaginary           Modulus           Frequency
-------------------------------------------------------------------------------
AR.1           1.3621              0.0000            1.3621              0.0000
MA.1           1.2901              0.0000            1.2901              0.0000
=====================================================================
```

[9]:
```
const                    0.003692
ar.L1.perc_log_return    0.734171
ma.L1.perc_log_return   -0.775121
dtype: float64
```

[10]:
```
fig,ax = plt.subplots(nrows=1, ncols=3, figsize=(24,6));
ax[0].plot(results.resid);
ax[0].set_title('Residuals after ARMA(1,1)')
tsaplots.plot_acf(results.resid, lags=20, ax = ax[1], zero=False)
ax[1].set_ylabel(r'$\rho_\ell$')
ax[1].set_xlabel(r'$\ell$');
tsaplots.plot_pacf(results.resid, lags=20, ax = ax[2], zero=False)
ax[2].set_ylabel(r'$\rho_\ell$')
ax[2].set_xlabel(r'$\ell$');
```



The PACF is indicative of the AR part of the process and the ACF is indicative of the MA part of the model. From this graph we can see that the ARMA(1,1) doesn't seem to suffice. If we look at both the ACF and the PACF we can see that the cutoff of both function seems to be at lag 6, so we will try to fit the ARMA(6,6) model.

[11]:
```
p = 6
q = 6
arma11 = ARMA(df.perc_log_return.dropna(), order=(p,q)) # Give the (p,q) order␣
↪of the ARMA(p,q) model
results = arma11.fit()
print(results.summary2())
results.params
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)

                              Results: ARMA
=====================================================================
Model:                ARMA              BIC:                 18722.5478
Dependent Variable:   perc_log_return   Log-Likelihood:      -9300.6
Date:                 2021-02-21 21:57  Scale:               1.0000
No. Observations:     5815              Method:              css-mle
Df Model:             13                Sample:              0
Df Residuals:         5802                                   5
```

```
Converged:           1.0000         S.D. of innovations: 1.198
No. Iterations:      74.0000        HQIC:                18661.664
AIC:                 18629.1930
--------------------------------------------------------------------
                       Coef.  Std.Err.    t    P>|t|   [0.025  0.975]
--------------------------------------------------------------------
const                  0.0037   0.0140  0.2617 0.7935 -0.0238  0.0311
ar.L1.perc_log_return -0.2527   0.2344 -1.0778 0.2812 -0.7121  0.2068
ar.L2.perc_log_return -0.3411   0.1773 -1.9240 0.0544 -0.6886  0.0064
ar.L3.perc_log_return -0.1633   0.2364 -0.6907 0.4898 -0.6265  0.3000
ar.L4.perc_log_return -0.1966   0.1842 -1.0674 0.2858 -0.5577  0.1644
ar.L5.perc_log_return -0.0572   0.1713 -0.3341 0.7383 -0.3930  0.2785
ar.L6.perc_log_return -0.2856   0.1238 -2.3063 0.0211 -0.5284 -0.0429
ma.L1.perc_log_return  0.2286   0.2359  0.9692 0.3325 -0.2337  0.6909
ma.L2.perc_log_return  0.2949   0.1787  1.6498 0.0990 -0.0554  0.6452
ma.L3.perc_log_return  0.0854   0.2345  0.3639 0.7159 -0.3743  0.5451
ma.L4.perc_log_return  0.2004   0.1871  1.0706 0.2844 -0.1664  0.5672
ma.L5.perc_log_return  0.0047   0.1702  0.0277 0.9779 -0.3288  0.3382
ma.L6.perc_log_return  0.2308   0.1297  1.7789 0.0753 -0.0235  0.4851
-----------------------------------------------------------------------------
                  Real           Imaginary          Modulus          Frequency
-----------------------------------------------------------------------------
AR.1            -1.0148           -0.6595            1.2102           -0.4083
AR.2            -1.0148            0.6595            1.2102            0.4083
AR.3             0.9560           -0.8333            1.2682           -0.1141
AR.4             0.9560            0.8333            1.2682            0.1141
AR.5            -0.0414           -1.2184            1.2191           -0.2554
AR.6            -0.0414            1.2184            1.2191            0.2554
MA.1            -0.9934           -0.6987            1.2145           -0.4024
MA.2            -0.9934            0.6987            1.2145            0.4024
MA.3             1.0150           -0.8516            1.3249           -0.1111
MA.4             1.0150            0.8516            1.3249            0.1111
MA.5            -0.0318           -1.2932            1.2936           -0.2539
MA.6            -0.0318            1.2932            1.2936            0.2539
====================================================================
```

[11]:
```
const                     0.003660
ar.L1.perc_log_return    -0.252656
ar.L2.perc_log_return    -0.341124
ar.L3.perc_log_return    -0.163255
ar.L4.perc_log_return    -0.196637
ar.L5.perc_log_return    -0.057230
ar.L6.perc_log_return    -0.285627
ma.L1.perc_log_return     0.228600
ma.L2.perc_log_return     0.294897
ma.L3.perc_log_return     0.085360
```

```
ma.L4.perc_log_return    0.200360
ma.L5.perc_log_return    0.004717
ma.L6.perc_log_return    0.230797
dtype: float64
```

[12]:

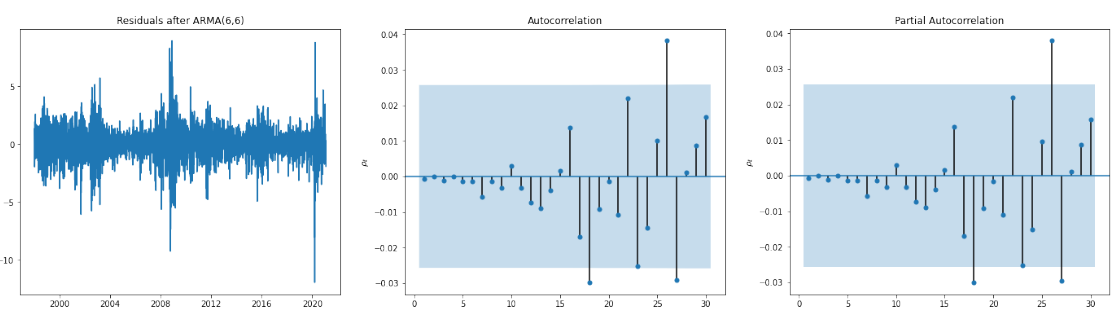
```
def print_acf(y, nlags=30):
    '''Prints the autocorrelations, partial autocorrelations, Q-statistics and␣
↪p-values.'''

    # Compute (partial) autocorrelations
    acf, qstat, pvalues = sm.tsa.stattools.acf(y, nlags=nlags, qstat=True)
    pacf = sm.tsa.stattools.pacf(y, nlags=nlags)
    T = len(y)
    ci = 2/np.sqrt(T) # Asymptotic 95% significance conf. int.

    # Print in table
    print('%3s | %8s | %8s | %8s | %8s' %('lag','AC', 'PAC','Q-stat','p-value') )
    for l in range(len(qstat)):
        if abs(acf[l+1]) > ci:
            print('%3i | *%7.3f | %8.3f | %8.3f | %8.3f'␣
↪%(l+1,acf[l+1],pacf[l+1],qstat[l],pvalues[l]) )
        else:
            print('%3i | %8.3f | %8.3f | %8.3f | %8.3f'␣
↪%(l+1,acf[l+1],pacf[l+1],qstat[l],pvalues[l]) )
    print('Autocorrelation, partial autocorrelation, Q-statistic and p-value.')
    print('Stars denote that the autocorrelation is outside the 95%% confidence␣
↪bounds (+/-%.3f) under i.i.d. assumption.' %ci)

    return acf, pacf, qstat, pvalues
```

[13]:

```
print_acf(results.resid)
fig,ax = plt.subplots(nrows=1, ncols=3, figsize=(24,6));
ax[0].plot(results.resid);
ax[0].set_title('Residuals after ARMA(6,6)')
tsaplots.plot_acf(results.resid, lags=30, ax = ax[1], zero=False)
ax[1].set_ylabel(r'$\rho_\ell$')
ax[1].set_xlabel(r'$\ell$');
tsaplots.plot_pacf(results.resid, lags=30, ax = ax[2], zero=False)
ax[2].set_ylabel(r'$\rho_\ell$')
ax[2].set_xlabel(r'$\ell$');
```

```
lag |       AC |      PAC |   Q-stat |  p-value
  1 |   -0.001 |   -0.001 |    0.003 |    0.954
  2 |   -0.000 |   -0.000 |    0.003 |    0.998
  3 |   -0.001 |   -0.001 |    0.010 |    1.000
  4 |   -0.000 |   -0.000 |    0.010 |    1.000
  5 |   -0.001 |   -0.001 |    0.020 |    1.000
```

```
  6 |    -0.001 |    -0.001 |     0.031 |     1.000
  7 |    -0.006 |    -0.006 |     0.222 |     1.000
  8 |    -0.001 |    -0.001 |     0.231 |     1.000
  9 |    -0.003 |    -0.003 |     0.291 |     1.000
 10 |     0.003 |     0.003 |     0.341 |     1.000
 11 |    -0.003 |    -0.003 |     0.401 |     1.000
 12 |    -0.007 |    -0.007 |     0.712 |     1.000
 13 |    -0.009 |    -0.009 |     1.181 |     1.000
 14 |    -0.004 |    -0.004 |     1.269 |     1.000
 15 |     0.002 |     0.001 |     1.282 |     1.000
 16 |     0.014 |     0.014 |     2.382 |     1.000
 17 |    -0.017 |    -0.017 |     4.049 |     0.999
 18 |  * -0.030 |    -0.030 |     9.233 |     0.954
 19 |    -0.009 |    -0.009 |     9.714 |     0.960
 20 |    -0.001 |    -0.002 |     9.725 |     0.973
 21 |    -0.011 |    -0.011 |    10.406 |     0.973
 22 |     0.022 |     0.022 |    13.219 |     0.927
 23 |    -0.025 |    -0.025 |    16.919 |     0.813
 24 |    -0.015 |    -0.015 |    18.152 |     0.796
 25 |     0.010 |     0.010 |    18.739 |     0.809
 26 |  *  0.038 |     0.038 |    27.245 |     0.397
 27 |  * -0.029 |    -0.029 |    32.219 |     0.224
 28 |     0.001 |     0.001 |    32.226 |     0.265
 29 |     0.009 |     0.009 |    32.660 |     0.292
 30 |     0.017 |     0.016 |    34.284 |     0.270
Autocorrelation, partial autocorrelation, Q-statistic and p-value.
Stars denote that the autocorrelation is outside the 95% confidence bounds
(+/-0.026) under i.i.d. assumption.

/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/stattools.py:541:
FutureWarning: fft=True will become the default in a future version of
statsmodels. To suppress this warning, explicitly set fft=False.
  warnings.warn(msg, FutureWarning)
```



Here we look at the Ljung–Box statistic (Q-statistic) of the residuals, the p-value even at 30 lagas is of 0.27, so we don't reject the null hyphotesis that this data can be from a white noise process.

So far the ARMA(6,6) model seems like a good fit.

This can also seen visualy as the (P)ACF plots as the values stay mostly within the 95% confidence interval around 0.

1.2.2 1.b)

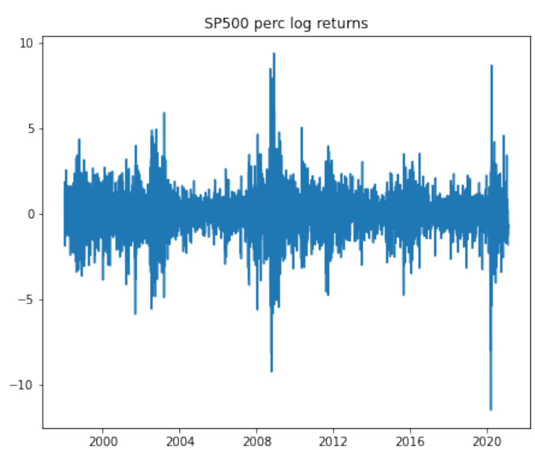
[14]:
```
print('Kurtosis', df.perc_log_return.kurt())
```

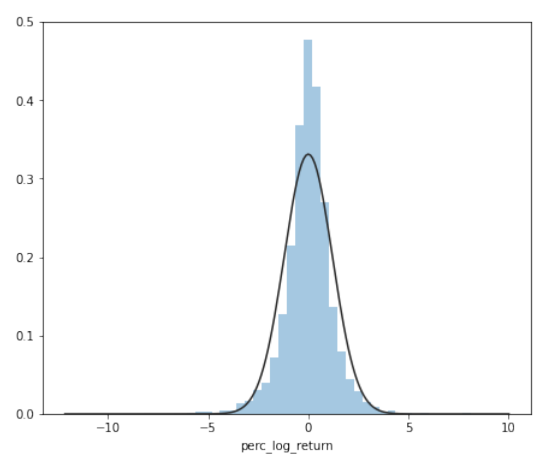
```
Kurtosis 7.097488137498708
```

The excess kurtosis of the percentage log return is approximately 7, so it is not a homoskedastic stationary process with Gaussian white noise. Combined with the fact that we see some clustering of the positive and negative peaks, we can conclude that the data seems to indicate that there is volatility clustering involved. For modeling this type of thing, GARCH-like models are often used.

[15]:
```
plt.figure(figsize=(16,6))
plt.subplot(121)
plt.plot(df.perc_log_return.dropna())
plt.title('SP500 perc log returns')
plt.subplot(122)
sns.distplot(df.perc_log_return.dropna(), kde=False, fit=stats.norm);
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



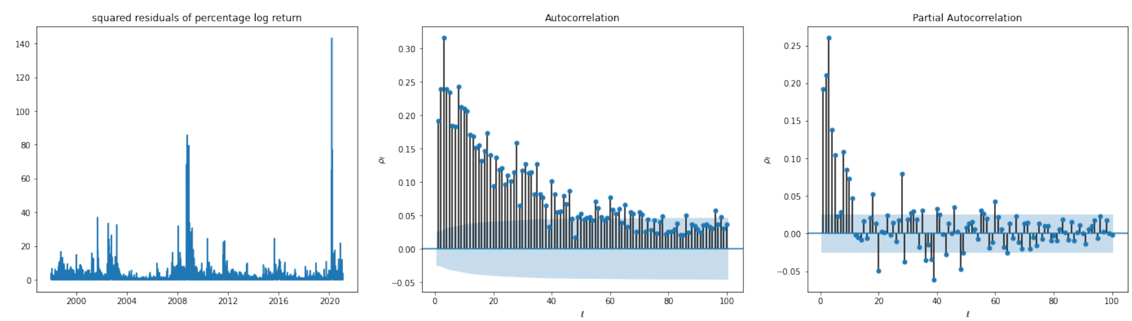


[16]:
```
print('Skewness', df.perc_log_return.skew())
```

```
Skewness -0.2953748423158725
```

If we look look at the distribution of the log returns plotted above we can see the impact of the high kurtosis and the slightly negative skewness of the returns.

[17]:
```python
fig,ax = plt.subplots(nrows=1, ncols=3, figsize=(24,6));
ax[0].plot(results.resid**2);
ax[0].set_title('squared residuals of percentage log return')
tsaplots.plot_acf(results.resid**2, lags=100, ax = ax[1], zero=False)
ax[1].set_ylabel(r'$\rho_\ell$')
ax[1].set_xlabel(r'$\ell$');
tsaplots.plot_pacf(results.resid**2, lags=100, ax = ax[2], zero=False)
ax[2].set_ylabel(r'$\rho_\ell$')
ax[2].set_xlabel(r'$\ell$');
```



Here we plotted the ACF and the PACF of the squared residuals to see if these were stationary and to see if the process has heteroskedasticity. We see both significant autocorrelation and siginificant values of the partial autocorrelation function which suggests that we can make a better model by incorporating volatiltiy clustering.

[18]:
```python
print_acf(np.abs(df.perc_log_return.dropna()))
```

```
lag |       AC |      PAC |   Q-stat |  p-value
  1 | *  0.262 |    0.262 |  398.183 |    0.000
  2 | *  0.300 |    0.249 |  923.274 |    0.000
  3 | *  0.319 |    0.223 | 1514.087 |    0.000
  4 | *  0.281 |    0.141 | 1975.351 |    0.000
  5 | *  0.292 |    0.132 | 2473.302 |    0.000
  6 | *  0.279 |    0.101 | 2927.507 |    0.000
  7 | *  0.246 |    0.050 | 3278.982 |    0.000
  8 | *  0.277 |    0.087 | 3725.051 |    0.000
  9 | *  0.255 |    0.058 | 4102.508 |    0.000
 10 | *  0.255 |    0.057 | 4480.187 |    0.000
 11 | *  0.248 |    0.044 | 4839.094 |    0.000
 12 | *  0.230 |    0.024 | 5147.300 |    0.000
 13 | *  0.242 |    0.040 | 5488.072 |    0.000
 14 | *  0.204 |   -0.006 | 5731.170 |    0.000
```

```
 15 | *  0.203 |     0.001 | 5972.202 |     0.000
 16 | *  0.213 |     0.020 | 6237.244 |     0.000
 17 | *  0.222 |     0.042 | 6524.619 |     0.000
 18 | *  0.213 |     0.031 | 6790.376 |     0.000
 19 | *  0.207 |     0.021 | 7039.462 |     0.000
 20 | *  0.185 |    -0.005 | 7238.941 |     0.000
 21 | *  0.217 |     0.038 | 7513.734 |     0.000
 22 | *  0.208 |     0.031 | 7766.097 |     0.000
 23 | *  0.199 |     0.020 | 7997.121 |     0.000
 24 | *  0.190 |     0.006 | 8206.944 |     0.000
 25 | *  0.177 |    -0.006 | 8389.813 |     0.000
 26 | *  0.174 |    -0.007 | 8567.055 |     0.000
 27 | *  0.176 |     0.002 | 8748.736 |     0.000
 28 | *  0.186 |     0.024 | 8950.522 |     0.000
 29 | *  0.147 |    -0.026 | 9077.710 |     0.000
 30 | *  0.170 |     0.008 | 9246.923 |     0.000
Autocorrelation, partial autocorrelation, Q-statistic and p-value.
Stars denote that the autocorrelation is outside the 95% confidence bounds
(+/-0.026) under i.i.d. assumption.
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/stattools.py:541:
FutureWarning: fft=True will become the default in a future version of
statsmodels. To suppress this warning, explicitly set fft=False.
  warnings.warn(msg, FutureWarning)
```

[18]:
```
(array([1.        , 0.26160999, 0.30039521, 0.31861319, 0.28149827,
        0.29245347, 0.27928812, 0.24566095, 0.27672756, 0.25453535,
        0.25458817, 0.24815882, 0.22994435, 0.24176681, 0.20418195,
        0.20329492, 0.2131621 , 0.22194172, 0.21341235, 0.20659256,
        0.18486362, 0.21695381, 0.20789323, 0.19889238, 0.18953059,
        0.17692326, 0.17416473, 0.17631726, 0.18580117, 0.14749829,
        0.17011569]),
 array([ 1.        ,  0.26165499,  0.24908867,  0.22267567,  0.14061904,
         0.13243564,  0.10131202,  0.05036501,  0.08699305,  0.0579938 ,
         0.05682778,  0.04441196,  0.02390018,  0.04018799, -0.0060968 ,
         0.0011406 ,  0.02029917,  0.04226778,  0.0307626 ,  0.02072621,
        -0.0049098 ,  0.03836896,  0.03138444,  0.02034438,  0.0062599 ,
        -0.00642512, -0.00683368,  0.00194728,  0.02352331, -0.02644807,
         0.00805167]),
 array([ 398.18271647,  923.27357985, 1514.08743902, 1975.35124353,
        2473.30178462, 2927.50721696, 3278.98219703, 3725.05094712,
        4102.50835402, 4480.18749302, 4839.09356879, 5147.2999959 ,
        5488.07240771, 5731.16998157, 5972.20150422, 6237.24407542,
        6524.61874312, 6790.37565186, 7039.46190772, 7238.9413891 ,
        7513.73388834, 7766.09703434, 7997.12072629, 8206.94409539,
        8389.81309313, 8567.0546852 , 8748.73584622, 8950.52237385,
        9077.70984769, 9246.92301455]),
 array([1.36941997e-088, 3.26354415e-201, 0.00000000e+000, 0.00000000e+000,
```

```
0.00000000e+000, 0.00000000e+000, 0.00000000e+000, 0.00000000e+000,
0.00000000e+000, 0.00000000e+000, 0.00000000e+000, 0.00000000e+000,
0.00000000e+000, 0.00000000e+000, 0.00000000e+000, 0.00000000e+000,
0.00000000e+000, 0.00000000e+000, 0.00000000e+000, 0.00000000e+000,
0.00000000e+000, 0.00000000e+000, 0.00000000e+000, 0.00000000e+000,
0.00000000e+000, 0.00000000e+000, 0.00000000e+000, 0.00000000e+000,
0.00000000e+000, 0.00000000e+000]))
```

Here we look again at the Ljung–Box statistic (Q-statistic) but this time of the absolute value of the residuals. Since all the p-values equal 0, we clearly reject the null hyphotesis and conclude that it exhibits serial correlation. (This is due to volatility clustering)

Next we will resume by doing the ARCH-LM Test.

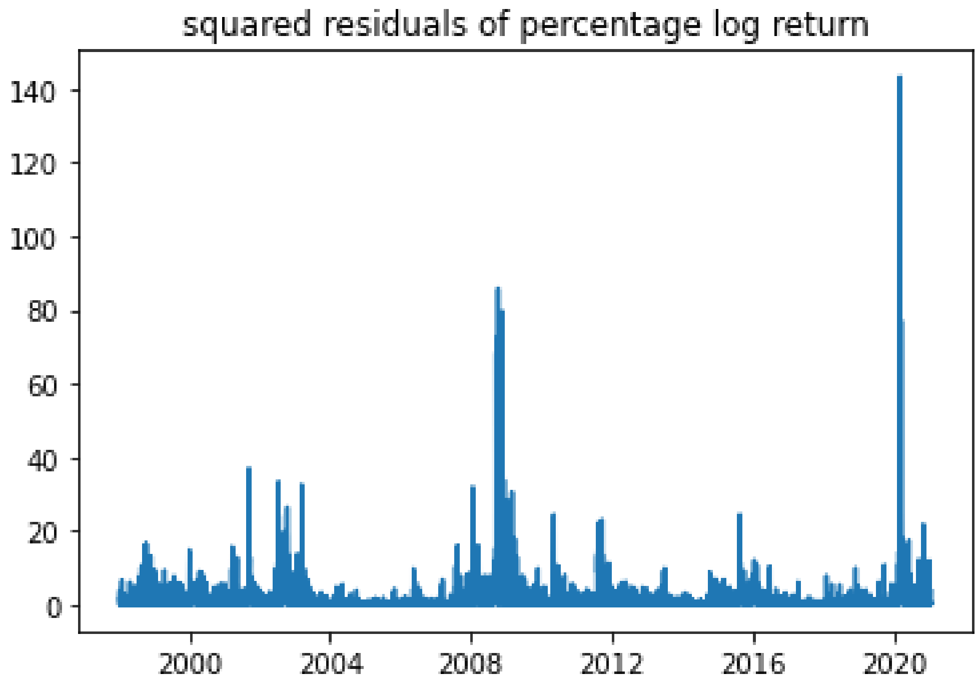
[19]:
```
name = ['LM','p-value (LM)','F','p-value (F)']
test = sm.stats.diagnostic.het_arch(np.abs(df.perc_log_return.dropna()),␣
 ↪store=False, ddof=p+q)
lzip(name, test)
```

[19]:
```
[('LM', 1306.1743145578864),
 ('p-value (LM)', 1.251772262380477e-252),
 ('F', 49.33006885126817),
 ('p-value (F)', 1.415341587994757e-288)]
```

This is a test of the conditional heteroscedasticy or the squared returns. From the results we can see that the p-value of the Lagrange Multiplier and of the F-statistic are extremely small, so we reject the Null-hyphotesis that the values of the coefficients in an ARCH-model setting should be all equal to zero.

[20]:
```
print("Skewness" ,results.resid.skew())
plt.plot(results.resid**2);
plt.title('squared residuals of percentage log return')
plt.show()
```

```
Skewness -0.3987515575262098
```



For modeling the heteroskedasticity of the data, GARCH-like models are very suitable. Although ARCH might work, the GARCH model provides more flexibility and is thus more likely to work.

If we look at the graph above we can also see that the periods where volatility are higher are usually during periods of market stress. This indicates that negative shocks are expected to increse the volatility more than larger positive shocks. This idicates that the EGARCH model might be a good solutions as it accounts for the impact of negative shocks while the ARCH and the GARCH model don't. This skewness of the returns (being negative), also points to this assumption of negative shocks have a bigger impact on volatility.

1.2.3 1.c)

GARCH/EGARCH MODEL

[21]:
```
# parameters to check
vols = ['EGARCH', "GARCH"]
ps = [1,2]
os = [0,1,2]
qs = [0,1,2]
ls = [1,3,5,6,7,9]
dists = ['normal', 'skewt']

# Variables to store the best results
best_aic = 20000
best_params = None
```

```
# Iteration number for progress printing
i = 0
for vol in vols:
  for p in ps:
    for o in os:
      for q in qs:
        for l in ls:
          for dist in dists:
            # Print progress
            i = i+1
            print("Iteration", i, "out of",␣
 ↪len(vols)*len(ps)*len(os)*len(qs)*len(ls)*len(dists))
            # Fit an AR(l)-vol(p,o,q) model with dist as distribution for the␣
 ↪innovation sequence
            am = arch_model(df.perc_log_return.dropna(), mean='AR', lags=l,␣
 ↪vol=vol, p=p ,o=o, q=q, dist=dist)
            res = am.fit()
            # Take the model with the best Akaike Information Coefficient
            if res.aic < best_aic:
              best_aic = res.aic
              best_params = (vol, p, o, q, dist, l)
```

```
Streaming output truncated to the last 0 lines.
```

In the above, we tried multiple model (hyper)parameters and used the one which had the highest AIC. While getting a better (log) likelihood is better, it should be kept in mind that introducing additional parameters could lead to overfitting. The AIC makes a trade off between the log likelihood of the model and the number of parameters used.

[22]:
```
import math
print(best_aic, best_params)
#best_params=('EGARCH',1,1,1, 'skewt', 6)
am = arch_model(df.perc_log_return.dropna(), mean='AR', lags=best_params[5],␣
 ↪vol=best_params[0], p=best_params[1],o=best_params[2],q=best_params[3],␣
 ↪dist=best_params[4])
res = am.fit()  # Fit model
print(res.summary())
#res.arch_lm_test()
```

```
15977.141842329896 ('EGARCH', 2, 2, 1, 'skewt', 9)
Iteration:      1,   Func. Count:     20,   Neg. LLF: 8053.205439006811
Iteration:      2,   Func. Count:     44,   Neg. LLF: 8042.74294951837
Iteration:      3,   Func. Count:     69,   Neg. LLF: 8028.579099354643
Iteration:      4,   Func. Count:     94,   Neg. LLF: 8025.639896394477
Iteration:      5,   Func. Count:    118,   Neg. LLF: 8015.351217084506
Iteration:      6,   Func. Count:    142,   Neg. LLF: 8013.445169279821
Iteration:      7,   Func. Count:    166,   Neg. LLF: 8013.093455068951
Iteration:      8,   Func. Count:    190,   Neg. LLF: 8012.955413795511
```

```
Iteration:      9,   Func. Count:    215,   Neg. LLF: 8012.901004381526
Iteration:     10,   Func. Count:    238,   Neg. LLF: 8011.960908472208
Iteration:     11,   Func. Count:    262,   Neg. LLF: 8011.794374254707
Iteration:     12,   Func. Count:    284,   Neg. LLF: 8007.939531800574
Iteration:     13,   Func. Count:    306,   Neg. LLF: 7986.753880629478
Iteration:     14,   Func. Count:    330,   Neg. LLF: 7981.469725228598
Iteration:     15,   Func. Count:    353,   Neg. LLF: 7981.362689728026
Iteration:     16,   Func. Count:    374,   Neg. LLF: 7978.306954073537
Iteration:     17,   Func. Count:    396,   Neg. LLF: 7976.2775797020095
Iteration:     18,   Func. Count:    420,   Neg. LLF: 7976.250796235638
Iteration:     19,   Func. Count:    442,   Neg. LLF: 7975.80850805251
Iteration:     20,   Func. Count:    463,   Neg. LLF: 7972.858185663295
Iteration:     21,   Func. Count:    485,   Neg. LLF: 7971.68009085382
Iteration:     22,   Func. Count:    507,   Neg. LLF: 7971.61458280483
Iteration:     23,   Func. Count:    528,   Neg. LLF: 7970.665504221666
Iteration:     24,   Func. Count:    548,   Neg. LLF: 7970.579380345675
Iteration:     25,   Func. Count:    568,   Neg. LLF: 7970.571196362975
Iteration:     26,   Func. Count:    588,   Neg. LLF: 7970.570943917165
Iteration:     27,   Func. Count:    608,   Neg. LLF: 7970.570921854405
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 7970.570921164948
            Iterations: 27
            Function evaluations: 609
            Gradient evaluations: 27
                                AR - EGARCH Model Results
================================================================================
=========
Dep. Variable:                  perc_log_return   R-squared:
0.004
Mean Model:                                  AR   Adj. R-squared:
0.002
Vol Model:                               EGARCH   Log-Likelihood:
-7970.57
Distribution:      Standardized Skew Student's t   AIC:
15977.1
Method:                      Maximum Likelihood   BIC:
16097.1
                                                  No. Observations:
5806
Date:                          Sun, Feb 21 2021   Df Residuals:
5796
Time:                                  22:00:47   Df Model:
10
                                    Mean Model
================================================================================
==
                     coef    std err          t      P>|t|       95.0% Conf.
Int.
```

```
-----------------------------------------------------------------------------
--
Const           2.1003e-03  1.149e-02      0.183      0.855
[-2.042e-02,2.462e-02]
perc...urn[1]     -0.0209  1.303e-02     -1.601      0.109
[-4.641e-02,4.683e-03]
perc...urn[2]     -0.0299  1.365e-02     -2.190  2.852e-02
[-5.665e-02,-3.142e-03]
perc...urn[3]     -0.0200  1.373e-02     -1.459      0.145
[-4.695e-02,6.883e-03]
perc...urn[4]     -0.0215  1.337e-02     -1.607      0.108
[-4.771e-02,4.713e-03]
perc...urn[5]     -0.0106  1.336e-02     -0.796      0.426
[-3.682e-02,1.555e-02]
perc...urn[6] -6.2838e-03  1.339e-02     -0.469      0.639
[-3.254e-02,1.997e-02]
perc...urn[7]  3.1311e-03  1.265e-02      0.248      0.804
[-2.166e-02,2.792e-02]
perc...urn[8]  7.4246e-03  1.324e-02      0.561      0.575
[-1.852e-02,3.337e-02]
perc...urn[9]  3.3793e-03  1.265e-02      0.267      0.789
[-2.141e-02,2.817e-02]
                               Volatility Model
============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
----------------------------------------------------------------------------
omega      1.2930e-03  2.059e-03      0.628      0.530 [-2.742e-03,5.328e-03]
alpha[1]       0.0591  3.262e-02      1.811  7.020e-02   [-4.871e-03,  0.123]
alpha[2]       0.0649  3.314e-02      1.959  5.006e-02   [-1.790e-05,  0.130]
gamma[1]      -0.1881  2.122e-02     -8.864  7.724e-19     [ -0.230, -0.147]
gamma[2]       0.0808  2.292e-02      3.525  4.234e-04    [3.587e-02,  0.126]
beta[1]        0.9856  2.708e-03    363.973      0.000     [  0.980,  0.991]
                                Distribution
==========================================================================
                 coef    std err          t      P>|t|     95.0% Conf. Int.
--------------------------------------------------------------------------
nu            11.3903      1.537      7.408  1.278e-13    [  8.377, 14.404]
lambda        -0.1365  1.890e-02     -7.221  5.148e-13 [ -0.174,-9.946e-02]
==========================================================================

Covariance estimator: robust
```

From simulation we can confirm what he previously assumed that the EGARCH model obatains better results than the GARCH model. From the AIC we see that the best model to fit is the EGARCH(2,1), with an AIC of 15977.1. For simplicity sake, and because the AIC doesn't seem to drop that much, we will resume our report using the EGARCH(1,1) model, that obtains an AIC of 15994.6.

[23]:
```python
import math
#print(best_aic, best_params)
best_params=('EGARCH',1,1,1, 'skewt', 6)
am = arch_model(df.perc_log_return.dropna(), mean='AR', lags=best_params[5],␣
 ↪vol=best_params[0], p=best_params[1],o=best_params[2],q=best_params[3],␣
 ↪dist=best_params[4])
res = am.fit()  # Fit model
print(res.summary())
#res.arch_lm_test()
```

```
Iteration:      1,   Func. Count:     15,   Neg. LLF: 8045.042466633063
Iteration:      2,   Func. Count:     36,   Neg. LLF: 8022.7514790383675
Iteration:      3,   Func. Count:     55,   Neg. LLF: 8015.621331756515
Iteration:      4,   Func. Count:     74,   Neg. LLF: 8013.138877921244
Iteration:      5,   Func. Count:     93,   Neg. LLF: 8012.210440564211
Iteration:      6,   Func. Count:    112,   Neg. LLF: 8012.047505217472
Iteration:      7,   Func. Count:    131,   Neg. LLF: 8011.929663866598
Iteration:      8,   Func. Count:    148,   Neg. LLF: 8009.497834845549
Iteration:      9,   Func. Count:    164,   Neg. LLF: 7986.890852110325
Iteration:     10,   Func. Count:    182,   Neg. LLF: 7985.89970776809
Iteration:     11,   Func. Count:    201,   Neg. LLF: 7985.891352976838
Iteration:     12,   Func. Count:    219,   Neg. LLF: 7985.873153863778
Iteration:     13,   Func. Count:    236,   Neg. LLF: 7985.626395151281
Iteration:     14,   Func. Count:    252,   Neg. LLF: 7985.013754782052
Iteration:     15,   Func. Count:    269,   Neg. LLF: 7984.420115862295
Iteration:     16,   Func. Count:    285,   Neg. LLF: 7984.401984945364
Iteration:     17,   Func. Count:    300,   Neg. LLF: 7984.31375176232
Iteration:     18,   Func. Count:    315,   Neg. LLF: 7984.291944666487
Iteration:     19,   Func. Count:    330,   Neg. LLF: 7984.289364208852
Iteration:     20,   Func. Count:    345,   Neg. LLF: 7984.288931861026
Iteration:     21,   Func. Count:    360,   Neg. LLF: 7984.288680342083
Iteration:     22,   Func. Count:    377,   Neg. LLF: 7984.288676472373
Iteration:     23,   Func. Count:    393,   Neg. LLF: 7984.288667703696
Iteration:     24,   Func. Count:    409,   Neg. LLF: 7984.288665665798
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 7984.288665368191
            Iterations: 24
            Function evaluations: 411
            Gradient evaluations: 24
                                AR - EGARCH Model Results
================================================================================
=========
Dep. Variable:                 perc_log_return   R-squared:
0.003
Mean Model:                                 AR   Adj. R-squared:
0.002
Vol Model:                              EGARCH   Log-Likelihood:
```

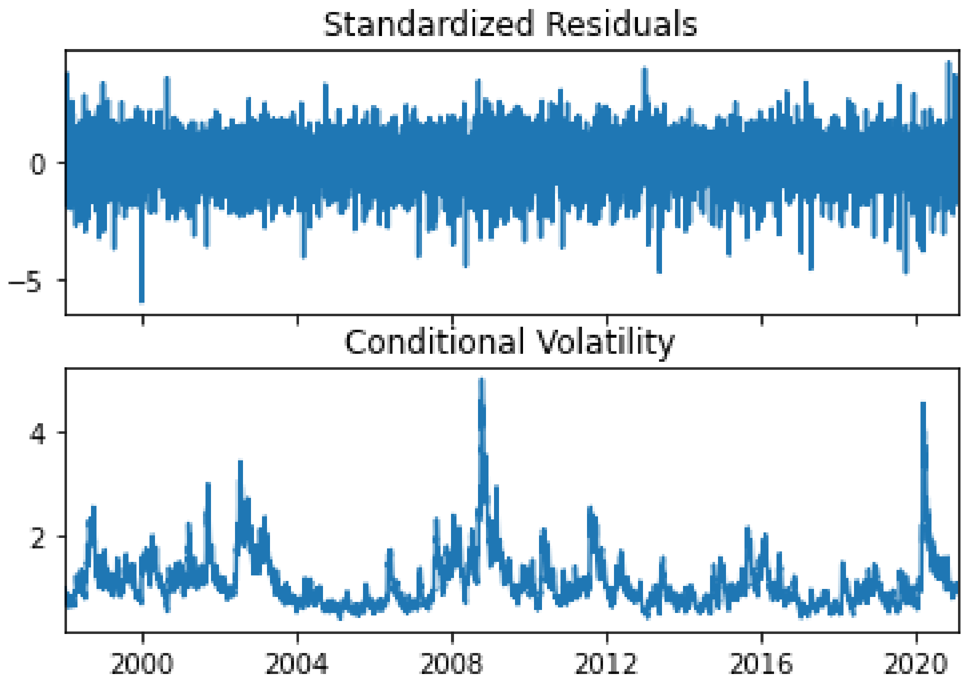
```
-7984.29
Distribution:      Standardized Skew Student's t   AIC:
15994.6
Method:                       Maximum Likelihood   BIC:
16081.3
                                                   No. Observations:
5809
Date:                           Sun, Feb 21 2021   Df Residuals:
5802
Time:                                   22:00:48   Df Model:
7
                                    Mean Model
================================================================================
==
                     coef    std err          t      P>|t|        95.0% Conf.
Int.
--------------------------------------------------------------------------------
--
Const           1.0724e-03  6.350e-03      0.169      0.866
[-1.137e-02,1.352e-02]
perc...urn[1]      -0.0213  8.469e-03     -2.516  1.187e-02
[-3.791e-02,-4.708e-03]
perc...urn[2]      -0.0256  9.142e-03     -2.800  5.111e-03
[-4.351e-02,-7.679e-03]
perc...urn[3]      -0.0161  7.729e-03     -2.087  3.690e-02
[-3.128e-02,-9.812e-04]
perc...urn[4]      -0.0197  1.046e-02     -1.882  5.982e-02
[-4.019e-02,8.143e-04]
perc...urn[5] -7.0119e-03  1.033e-02     -0.679      0.497
[-2.726e-02,1.324e-02]
perc...urn[6] -3.1589e-03  8.785e-03     -0.360      0.719
[-2.038e-02,1.406e-02]
                               Volatility Model
=============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
-----------------------------------------------------------------------------
omega      1.4029e-03  1.710e-03      0.820      0.412 [-1.949e-03,4.755e-03]
alpha[1]       0.1253  1.375e-02      9.117  7.747e-20    [9.839e-02,  0.152]
gamma[1]      -0.1140  9.582e-03    -11.895  1.262e-32  [ -0.133,-9.519e-02]
beta[1]        0.9844  2.631e-03    374.173      0.000      [  0.979,  0.990]
                              Distribution
========================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------
nu            11.6994      1.608      7.275  3.475e-13 [  8.547, 14.852]
lambda        -0.1369  1.834e-02     -7.467  8.198e-14 [ -0.173, -0.101]
========================================================================
```

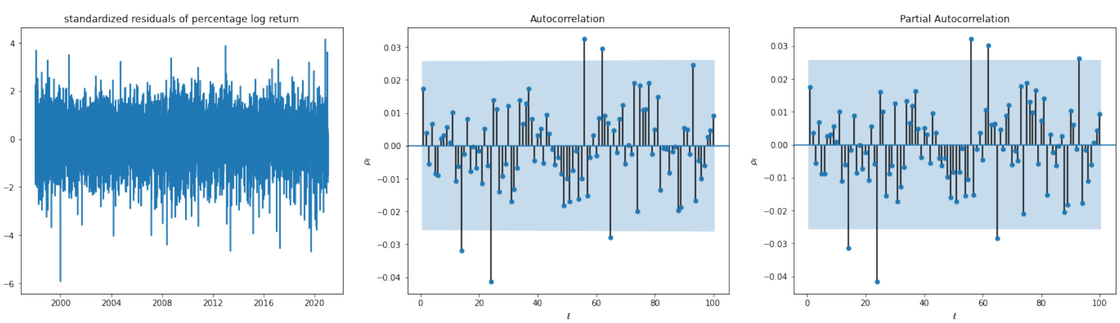
```
Covariance estimator: robust
```

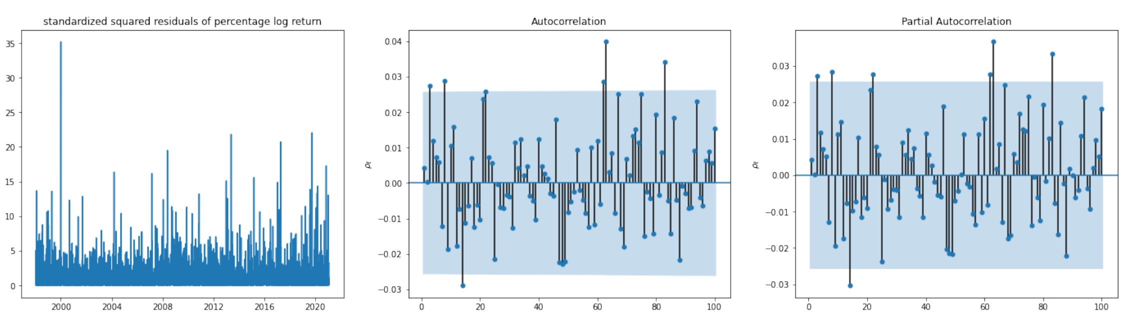
[24]:
```python
print('p-value for homoskedasticity (ARCH LM test)', sm.stats.diagnostic.
 ↪het_arch(res.std_resid.dropna(), store=False, ddof=p+q)[1])
print('p-value for white noise (Q-statistic)', sm.tsa.stattools.acf(res.
 ↪std_resid.dropna(), nlags=30, qstat=True)[2][29])
plt.close()
res.plot()
plt.show()
plt.close()

fig,ax = plt.subplots(nrows=1, ncols=3, figsize=(24,6));
ax[0].plot(res.std_resid.dropna());
ax[0].set_title('standardized residuals of percentage log return')
tsaplots.plot_acf(res.std_resid.dropna(), lags=100, ax = ax[1], zero=False)
ax[1].set_ylabel(r'$\rho_\ell$')
ax[1].set_xlabel(r'$\ell$');
tsaplots.plot_pacf(res.std_resid.dropna(), lags=100, ax = ax[2], zero=False)
ax[2].set_ylabel(r'$\rho_\ell$')
ax[2].set_xlabel(r'$\ell$');

fig,ax = plt.subplots(nrows=1, ncols=3, figsize=(24,6));
ax[0].plot(res.std_resid.dropna()**2);
ax[0].set_title('standardized squared residuals of percentage log return')
tsaplots.plot_acf(res.std_resid.dropna()**2, lags=100, ax = ax[1], zero=False)
ax[1].set_ylabel(r'$\rho_\ell$')
ax[1].set_xlabel(r'$\ell$');
tsaplots.plot_pacf(res.std_resid.dropna()**2, lags=100, ax = ax[2], zero=False)
ax[2].set_ylabel(r'$\rho_\ell$')
ax[2].set_xlabel(r'$\ell$');
```

```
p-value for homoskedasticity (ARCH LM test) 0.2686404036890153
p-value for white noise (Q-statistic) 0.5897707641362849
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/stattools.py:541:
FutureWarning: fft=True will become the default in a future version of
statsmodels. To suppress this warning, explicitly set fft=False.
  warnings.warn(msg, FutureWarning)
```







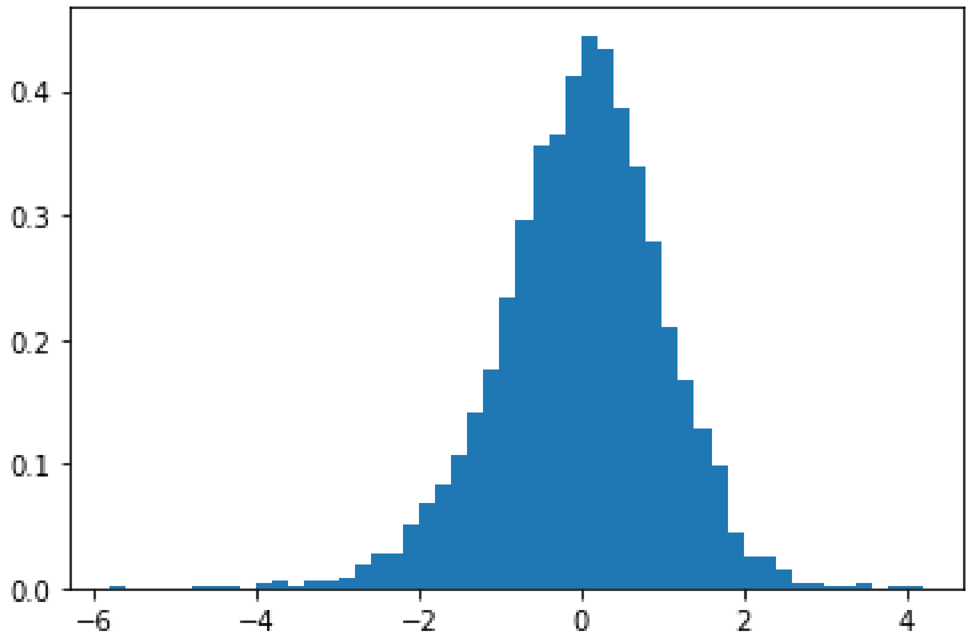
We have run the Lagrange Multiplier test on the squared standardized residuals. This results in a value of around 0.27 which means that we do not reject the null hypothesis of the standardized residuals being homoskedastic. Furthermore we see that the standardized residuals behave as white noise according to the Ljung-Box statistic with a p-value of 0.59 at lag 30. So we conclude from these statistics that the standardized residuals behave as homoskedastic white noise.

In terms of the news-impact curve (NIC) we also took that into account when selecting the EGARCH model. Because the EGARCH model accounts for assymetry in the NIC which is more realistic than the ARRCH and GARCH model as previously mentioned.

[25]:
```
import math
print(best_aic, best_params)
params=('EGARCH',1,1,1, 'normal', 6)
am = arch_model(df.perc_log_return.dropna(), mean='AR', lags=params[5],␣
 ↪vol=params[0], p=params[1],o=params[2],q=params[3], dist=params[4])
res = am.fit()  # Fit model
plt.hist(res.std_resid.dropna(), density=True, bins=50)
print('p-value for normality', stats.jarque_bera(res.std_resid.dropna())[1])
print("Skewness", res.std_resid.skew())
params=('EGARCH',1,1,1, 'skewt', 6)
am = arch_model(df.perc_log_return.dropna(), mean='AR', lags=params[5],␣
 ↪vol=params[0], p=params[1],o=params[2],q=params[3], dist=params[4])
res = am.fit()  # Fit model
```

```
15977.141842329896 ('EGARCH', 1, 1, 1, 'skewt', 6)
Iteration:      1,   Func. Count:     13,   Neg. LLF: 8087.71691855004
Iteration:      2,   Func. Count:     30,   Neg. LLF: 8068.598401467927
Iteration:      3,   Func. Count:     47,   Neg. LLF: 8066.096836670223
Iteration:      4,   Func. Count:     64,   Neg. LLF: 8064.687467840405
Iteration:      5,   Func. Count:     81,   Neg. LLF: 8063.766394204421
Iteration:      6,   Func. Count:     98,   Neg. LLF: 8063.544167889919
Iteration:      7,   Func. Count:    115,   Neg. LLF: 8063.53162027044
Iteration:      8,   Func. Count:    131,   Neg. LLF: 8063.524750280933
Iteration:      9,   Func. Count:    146,   Neg. LLF: 8062.551983435808
Iteration:     10,   Func. Count:    162,   Neg. LLF: 8057.2276801242515
Iteration:     11,   Func. Count:    179,   Neg. LLF: 8057.210310564896
Iteration:     12,   Func. Count:    195,   Neg. LLF: 8057.207685706937
Iteration:     13,   Func. Count:    209,   Neg. LLF: 8057.012455453565
Iteration:     14,   Func. Count:    225,   Neg. LLF: 8056.95429109327
Iteration:     15,   Func. Count:    238,   Neg. LLF: 8056.923041042986
Iteration:     16,   Func. Count:    251,   Neg. LLF: 8056.921495498993
Iteration:     17,   Func. Count:    264,   Neg. LLF: 8056.920008822536
Iteration:     18,   Func. Count:    278,   Neg. LLF: 8056.91968412343
Iteration:     19,   Func. Count:    291,   Neg. LLF: 8056.91939247088
Iteration:     20,   Func. Count:    304,   Neg. LLF: 8056.919306254933
Iteration:     21,   Func. Count:    318,   Neg. LLF: 8056.919291373074
Iteration:     22,   Func. Count:    331,   Neg. LLF: 8056.9192749415115
Iteration:     23,   Func. Count:    344,   Neg. LLF: 8056.91927268463‌9
Iteration:     24,   Func. Count:    358,   Neg. LLF: 8056.919267793559
```

```
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 8056.919267167436
            Iterations: 24
            Function evaluations: 361
            Gradient evaluations: 24
p-value for normality 0.0
Skewness -0.33683775872719884
Iteration:      1,   Func. Count:     15,   Neg. LLF: 8045.042466633063
Iteration:      2,   Func. Count:     36,   Neg. LLF: 8022.7514790383675
Iteration:      3,   Func. Count:     55,   Neg. LLF: 8015.621331756515
Iteration:      4,   Func. Count:     74,   Neg. LLF: 8013.138877921244
Iteration:      5,   Func. Count:     93,   Neg. LLF: 8012.210440564211
Iteration:      6,   Func. Count:    112,   Neg. LLF: 8012.047505217472
Iteration:      7,   Func. Count:    131,   Neg. LLF: 8011.929663866598
Iteration:      8,   Func. Count:    148,   Neg. LLF: 8009.497834845549
Iteration:      9,   Func. Count:    164,   Neg. LLF: 7986.890852110325
Iteration:     10,   Func. Count:    182,   Neg. LLF: 7985.89970776809
Iteration:     11,   Func. Count:    201,   Neg. LLF: 7985.891352976838
Iteration:     12,   Func. Count:    219,   Neg. LLF: 7985.873153863778
Iteration:     13,   Func. Count:    236,   Neg. LLF: 7985.626395151281
Iteration:     14,   Func. Count:    252,   Neg. LLF: 7985.013754782052
Iteration:     15,   Func. Count:    269,   Neg. LLF: 7984.420115862295
Iteration:     16,   Func. Count:    285,   Neg. LLF: 7984.401984945364
Iteration:     17,   Func. Count:    300,   Neg. LLF: 7984.31375176232
Iteration:     18,   Func. Count:    315,   Neg. LLF: 7984.291944666487
Iteration:     19,   Func. Count:    330,   Neg. LLF: 7984.289364208852
Iteration:     20,   Func. Count:    345,   Neg. LLF: 7984.288931861026
Iteration:     21,   Func. Count:    360,   Neg. LLF: 7984.288680342083
Iteration:     22,   Func. Count:    377,   Neg. LLF: 7984.288676472373
Iteration:     23,   Func. Count:    393,   Neg. LLF: 7984.288667703696
Iteration:     24,   Func. Count:    409,   Neg. LLF: 7984.288665665798
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 7984.288665368191
            Iterations: 24
            Function evaluations: 411
            Gradient evaluations: 24
```



We also tried running the model with a normal distribution for the residuals but in this case we actually end up with standardized residuals that do not look normal according to the graph and for which the Jargue Bera statistic rejects normality with a p-value of 0.

Visualy we can also see that the graph looks skewd and by printing the value of the skewnness of -0.33.

We decided in the end to go with a skew T distribution. This also results in a higher AIC of the model.

1.2.4 1.d)

As described above, we tried many different model parameters algorithmically. In the end we are satisfied with the models included above since they pass the diagnostic tests.

1.2.5 1.e)

The formula for the EGARCH model that is used by the arch_model function is given by:

$$\ln \sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i \left(|\epsilon_{t-i}| - \sqrt{2/\pi} \right) + \sum_{j=1}^{o} \gamma_j \epsilon_{t-j} + \sum_{k=1}^{q} \beta_k \ln \sigma_{t-k}^2$$

where $\epsilon_t = a_t / \sigma_t$.
In our case we have the EGARCH(1,1) model, therefore we have that:

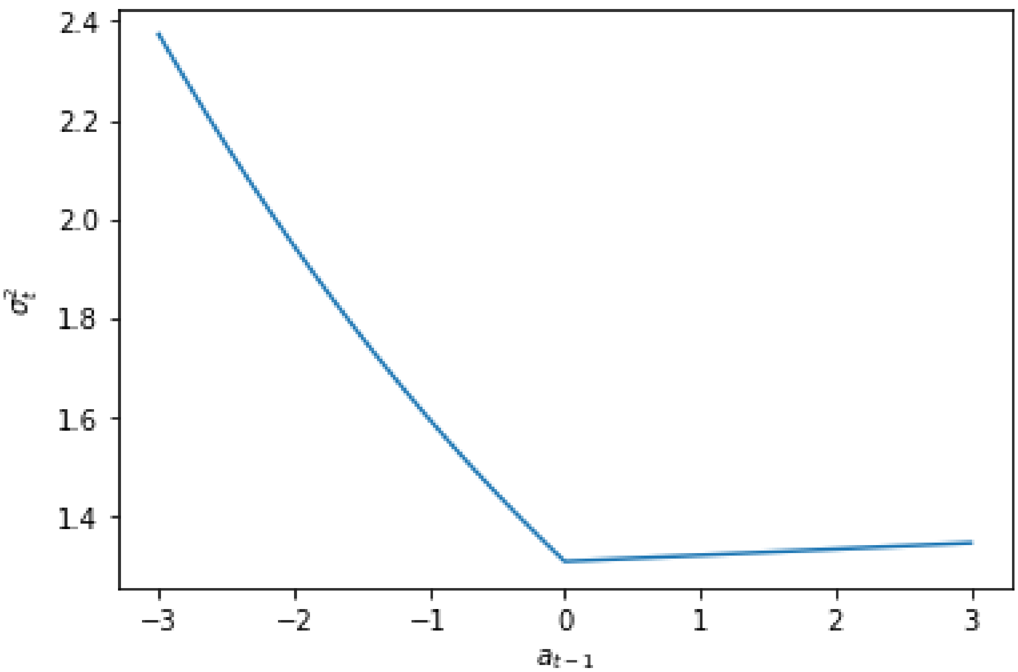
$$\ln \sigma_t^2 = \omega + \alpha \left(|e_{t-1}| - \sqrt{2/\pi} \right) + \gamma e_{t-1} + \beta \ln \sigma_{t-1}^2$$

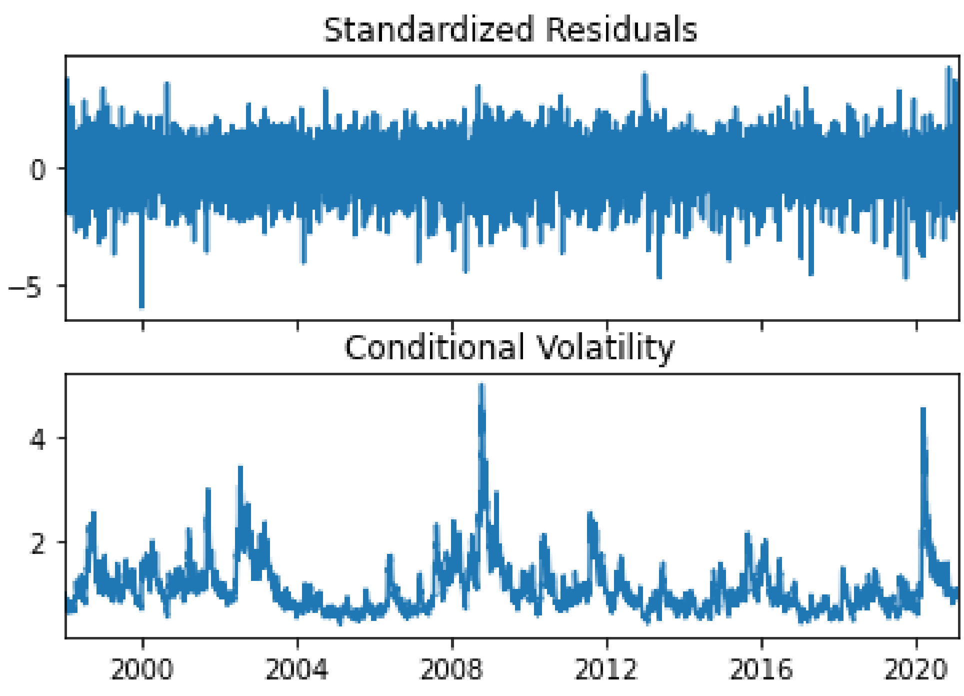
Where $\omega = 1.4025e{-}03$, $\alpha = 0.1253$, $\gamma = -0.1140$, $\beta = 0.9844$. (This results are taken from the res.summary() function) So to calculate the news impact we want to see what the impact of previous values of the process of future volatility. Remark: The EGARCH model is asymmetric because, positive return shocks generate less volatility then negative return shocks, all else being equal. The news Impact curve is given by:

$$\sigma_t^2 = A\exp[(\gamma+\alpha)\epsilon_{t-1}], \text{for } \epsilon_{t-1} > 0, \text{ and}$$
$$\sigma_t^2 = A\exp[(\gamma-\alpha)\epsilon_{t-1}], \text{for } \epsilon_{t-1} < 0,$$

where $A \equiv \sigma^{2\beta} \cdot \exp[\omega - \alpha \cdot \sqrt{\frac{2}{\pi}}]$

[26]:
```
#News Impact Curve
es = np.linspace(-3,3,1000)
var = np.var(df.perc_log_return.dropna())
A = var ** res.params['beta[1]'] * math.exp(res.params['omega'] - res.
 ↪params['alpha[1]'] * math.sqrt(2/math.pi))
def computeNIC(a):
  if a > 0:
    return A* np.exp((res.params['gamma[1]']+res.params['alpha[1]'])/math.
 ↪sqrt(var)*a)
  else:
    return A* np.exp((res.params['gamma[1]']-res.params['alpha[1]'])/math.
 ↪sqrt(var)*a)
plt.plot(es, list(map(computeNIC, es)))
plt.xlabel('$a_{t-1}$')
plt.ylabel('$\sigma_t^2$')
plt.show()
plt.close()
res.plot()
plt.show()
```





Here we ploted the news impact curve of the EGARCH model news impact curve, which is very similar to the theoritical one seen in class.

We also ploted estimated volatility from our model, here we can definatevely see how negative shocks have a bigger impact on the volatility in the EGARCH as wee look at the 2008 peak (Financial Crisis) and the 2020 peak (Covid Crisis).

1.2.6 2.a)

In this simulation we ran the ARMA(1,1) model because running an ARMA(p,q) of higher order would take making significant alterations to the code provided for running the simulation. So for simplicity sake we will use the ARMA(1,1)-EGARCH(1,1).

[27]:
```
p = 1
q = 1
arma11 = ARMA(df.perc_log_return.dropna(), order=(p,q)) # Give the (p,q) order␣
 ↪of the ARMA(p,q) model
results = arma11.fit()
results.params
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

[27]:
```
const                    0.003692
ar.L1.perc_log_return    0.734171
ma.L1.perc_log_return   -0.775121
dtype: float64
```

[28]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
%matplotlib inline
import matplotlib.pyplot as plt
# ARMA(1,1)-EGARCH(1,1,1, 'skewt')

def computeConditionalVolatility(prev_epsilon, prev_sigma2):
  A = prev_sigma2 ** res.params['beta[1]'] * math.exp(res.params['omega'] - res.
 ↪params['alpha[1]'] * math.sqrt(2/math.pi))
  if prev_epsilon > 0:
    return A* np.exp((res.params['gamma[1]']+res.
 ↪params['alpha[1]'])*prev_epsilon)
  else:
    return A* np.exp((res.params['gamma[1]']-res.
 ↪params['alpha[1]'])*prev_epsilon)

def simulation(params, rT, aT, sT, R=2000, m=22):
    '''
```

```
    Simulating the distribution of a monthly return from a daily ARMA-GARCH␣
↪model.

    Notes:
    * Need to fill in an expression for s2 and r depending on the model for the␣
↪conditional volatility and mean.
    * The program is applicable if only one lag / starting value is needed, e.g.␣
↪ARMA(1,1)-GARCH(1,1);
      for higher-order models the program needs adjustment.

    INPUT
    -----
    params: parameter vector (both mean and volatility parameters) from␣
↪ARMAResults

    R: number of replications
    m: number of trading days in a month (~21 days), plus one starting value␣
↪(change this if more starting values are needed)

    Starting values (if more lags are involved, then more values from January␣
↪2020 are needed)
    rT: the daily return at time of forecasting
    aT: the residual at time of forecasting
    sT: the estimated volatility at time of forecasting


    OUTPUT
    ------
    monthreturn: array with R simulated monthly returns

    '''
    np.random.seed(0)
    # Define variables
    r = np.zeros(m+1)  # daily returns
    a = np.zeros(m+1)  # disturbances
    s2 = np.zeros(m+1)  # conditional variance
    epsilon = np.zeros(m+1)  # standardized disturbances
    monthreturn = np.zeros(R)  # monthly return

    # Initialising various variables at the relevant starting values
    r[0] = rT
    a[0] = aT
    s2[0] = sT**2
    epsilon[0] = aT / sT

    # Draw R times the monthly return according to the ARMA-GARCH specification
```

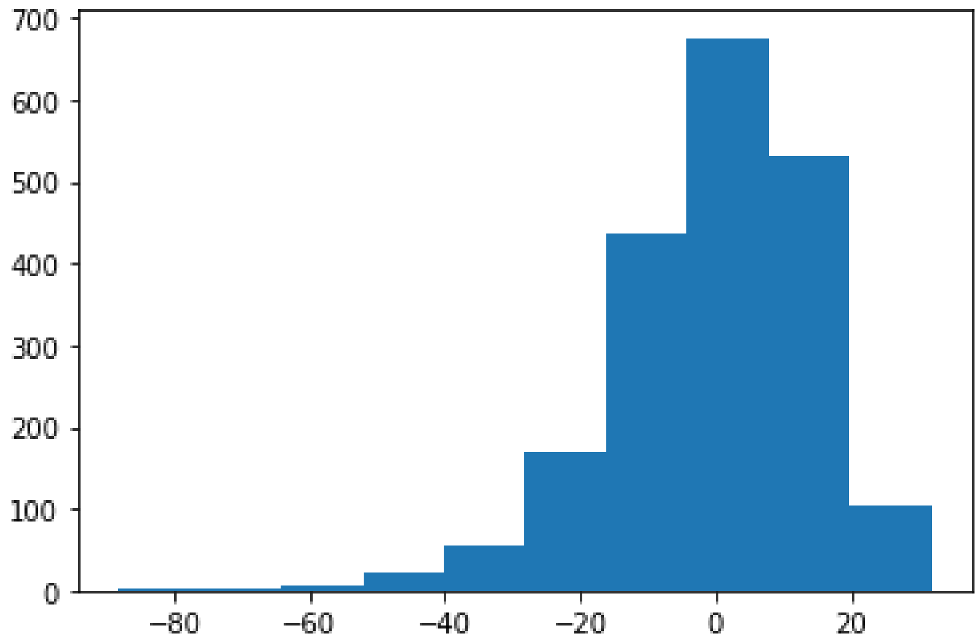
```
    for rep in range(R):
        # use random number generator from other distribution for epsilon if␣
↪necessary
        if 'dist' in params and params['dist'] == 'normal':
          epsilon[1:] = np.random.randn(m)
        else:
          epsilon[1:] = SkewStudent(lam=res.params['lambda'], eta=res.
↪params['nu']).rvs(size=m)

        # use estimated GARCH equation, expressing conditional variance s2 in␣
↪terms of s2[h-1] and epsilon[h-1]
        for h in range(1,m):
            # EGARCH(1,1)
            s2[h] = computeConditionalVolatility(epsilon[h-1], s2[h-1])


        a[1:] = np.sqrt(s2[1:])*epsilon[1:]

        # use estimated ARMA equation, expressing r in terms of a and possibly␣
↪r[h-1] and/or a[h-1]
        for h in range(1,m):
            r[h] = results.params['const'] + a[h]
            for i in range(p):
              r[h] += results.params['ar.L' + str(i+1) + '.perc_log_return'] *␣
↪r[h-i]
            for i in range(q):
              r[h] -= results.params['ma.L' + str(i+1) + '.perc_log_return'] *␣
↪a[h-i]
        monthreturn[rep] = sum(r[1:])

    return monthreturn
```

[29]:
```
p = 1#best_params[0]
q = 1#best_params[1]
arma11 = ARMA(df.perc_log_return.dropna(), order=(p,q)) # Give the (p,q) order␣
 ↪of the ARMA(p,q) model
results = arma11.fit()
print(results.summary2())

sims = simulation([], df.perc_log_return[-1], res.resid[-1], res.
 ↪conditional_volatility[-1])
plt.hist(sims)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

```
                           Results: ARMA
====================================================================
Model:              ARMA             BIC:                 18683.3797
Dependent Variable: perc_log_return  Log-Likelihood:      -9324.4
Date:               2021-02-21 22:00 Scale:               1.0000
No. Observations:   5815             Method:              css-mle
Df Model:           3                Sample:              0
Df Residuals:       5812                                  5
Converged:          1.0000           S.D. of innovations: 1.203
No. Iterations:     16.0000          HQIC:                18665.984
AIC:                18656.7069
--------------------------------------------------------------------
                       Coef.  Std.Err.    t    P>|t|   [0.025  0.975]
--------------------------------------------------------------------
const                  0.0037   0.0133  0.2767 0.7820 -0.0225  0.0298
ar.L1.perc_log_return  0.7342   0.0933  7.8663 0.0000  0.5512  0.9171
ma.L1.perc_log_return -0.7751   0.0869 -8.9247 0.0000 -0.9453 -0.6049
------------------------------------------------------------------------------
                Real           Imaginary          Modulus          Frequency
------------------------------------------------------------------------------
AR.1           1.3621             0.0000           1.3621             0.0000
MA.1           1.2901             0.0000           1.2901             0.0000
====================================================================
```

[29]:
```
(array([  2.,   1.,   7.,  21.,  54., 170., 436., 676., 530., 103.]),
 array([-88.11141681, -76.12205439, -64.13269196, -52.14332954,
        -40.15396711, -28.16460469, -16.17524226,  -4.18587984,
          7.80348259,  19.79284501,  31.78220744]),
 <a list of 10 Patch objects>)
```



[30]:
```
print(pd.DataFrame(sims).describe())
```

```
                 0
count  2000.000000
mean     -0.330002
std      14.578346
min     -88.111417
25%      -8.526895
50%       1.436921
75%       9.809259
max      31.782207
```

From here we can see that the standard deviation of the monthly return is 14.578346

[31]:
```
np.std(df.perc_log_return)*math.sqrt(22)
```

[31]: 5.651360198990237

Based on the standard deviation of the daily average log returns, we have computed (under the assumption of independence and multivariate normality / sum-stability) that the standard deviation of the average monthly log returns should be 5.65. However, there are two assumption that do not hold here. First of all, the log returns are not normally distributed. Secondly, the conditional variance based on the last part of the stock price process is not the same as the unconditional variance (because there is volatility clustering). Because of volatility clustering, there are going to be months with high volatilty and months with low volatility and so their average (as in the unconditional volatility) is not necessarily representative of the volatility next month.

1.2.7 2.b)

[32]:
```
print('Skewness',pd.DataFrame(sims).skew()[0])
print('Excess Kurtosis',pd.DataFrame(sims).kurt()[0])
print('Skewness of unconditional distribution', stats.skew(df.perc_log_return.
  ↪dropna()))
print('Excess Kurtosis of unconditional distribution', stats.kurtosis(df.
  ↪perc_log_return.dropna()))
```

```
Skewness -0.9769832766948386
Excess Kurtosis 2.186906132475884
Skewness of unconditional distribution -0.29529864386112153
Excess Kurtosis of unconditional distribution 7.090355228425276
```

The skewness of the histogram in 2.a) can be explained due to the use of the EGARCH model, positive return shocks generate less volatility then negative return shocks. Furthermore when returns are negative, changes can occur much faster, therfore we have more outliers when the returns are negative, making the graph have a negative skeweness. We can also see that the distribution of the monthly returns has excess kutosis and this can be explained by the assumption of volatility clustering in the EGARCH model.

1.2.8 2.c)

[33]:
```
params=('GARCH',1,1,1, 'normal', 6)
am = arch_model(df.perc_log_return.dropna(), mean='AR', lags=params[5],␣
  ↪vol=params[0], p=1,o=1,q=1, dist=params[4])
res = am.fit()  # Fit model
print(res.summary())
print(sm.stats.diagnostic.het_arch(res.resid.dropna(), store=False, ddof=p+q))
sims = simulation({'dist': 'normal'}, df.perc_log_return[-1], res.resid[-1], res.
  ↪conditional_volatility[-1])
```

```
Iteration:      1,   Func. Count:     13,   Neg. LLF: 8103.538216350396
Iteration:      2,   Func. Count:     32,   Neg. LLF: 8087.280457713432
Iteration:      3,   Func. Count:     49,   Neg. LLF: 8085.59022449271
Iteration:      4,   Func. Count:     66,   Neg. LLF: 8085.459857124251
Iteration:      5,   Func. Count:     82,   Neg. LLF: 8085.1215202542935
Iteration:      6,   Func. Count:     99,   Neg. LLF: 8085.077114597483
Iteration:      7,   Func. Count:    114,   Neg. LLF: 8084.815879094249
Iteration:      8,   Func. Count:    130,   Neg. LLF: 8084.737316831269
Iteration:      9,   Func. Count:    146,   Neg. LLF: 8084.664783209599
Iteration:     10,   Func. Count:    160,   Neg. LLF: 8080.9473216482165
Iteration:     11,   Func. Count:    175,   Neg. LLF: 8079.771493846083
Iteration:     12,   Func. Count:    189,   Neg. LLF: 8078.046870739004
Iteration:     13,   Func. Count:    204,   Neg. LLF: 8077.824809258805
Iteration:     14,   Func. Count:    218,   Neg. LLF: 8076.273741771482
Iteration:     15,   Func. Count:    232,   Neg. LLF: 8075.379216872994
```

```
Iteration:     16,   Func. Count:    247,   Neg. LLF: 8075.37652443711
Iteration:     17,   Func. Count:    261,   Neg. LLF: 8075.331670270176
Iteration:     18,   Func. Count:    275,   Neg. LLF: 8075.320897694915
Iteration:     19,   Func. Count:    288,   Neg. LLF: 8075.320582995852
Iteration:     20,   Func. Count:    301,   Neg. LLF: 8075.320568947811
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 8075.320568947561
            Iterations: 20
            Function evaluations: 301
            Gradient evaluations: 20
                         AR - GJR-GARCH Model Results
==============================================================================
Dep. Variable:        perc_log_return   R-squared:                       0.004
Mean Model:                        AR   Adj. R-squared:                  0.002
Vol Model:                  GJR-GARCH   Log-Likelihood:               -8075.32
Distribution:                  Normal   AIC:                           16172.6
Method:            Maximum Likelihood   BIC:                           16246.0
                                        No. Observations:                 5809
Date:                Sun, Feb 21 2021   Df Residuals:                     5802
Time:                        22:01:26   Df Model:                            7
                                  Mean Model
================================================================================
=
                     coef    std err          t      P>|t|       95.0% Conf.
Int.
--------------------------------------------------------------------------------
-
Const           4.6693e-03  1.135e-02      0.411      0.681
[-1.757e-02,2.691e-02]
perc...urn[1]     -0.0152  1.429e-02     -1.064      0.287
[-4.321e-02,1.280e-02]
perc...urn[2]     -0.0147  1.451e-02     -1.015      0.310
[-4.317e-02,1.371e-02]
perc...urn[3]     -0.0176  1.428e-02     -1.230      0.219
[-4.556e-02,1.043e-02]
perc...urn[4]     -0.0133  1.417e-02     -0.942      0.346
[-4.111e-02,1.443e-02]
perc...urn[5]     -0.0172  1.401e-02     -1.224      0.221
[-4.461e-02,1.030e-02]
perc...urn[6] -6.5645e-03  1.414e-02     -0.464      0.642
[-3.427e-02,2.114e-02]
                              Volatility Model
=============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
-----------------------------------------------------------------------------
omega          0.0182  3.928e-03      4.642  3.455e-06  [1.053e-02,2.593e-02]
alpha[1]   5.5877e-03  8.018e-03      0.697      0.486 [-1.013e-02,2.130e-02]
gamma[1]       0.1430  1.959e-02      7.300  2.880e-13       [  0.105,  0.181]
```

```
beta[1]         0.9062  1.319e-02     68.689      0.000      [  0.880,  0.932]
=============================================================================

Covariance estimator: robust
(1293.642148323199, 5.648904882022049e-250, 48.73461136204998,
3.3892123825566005e-285)
```

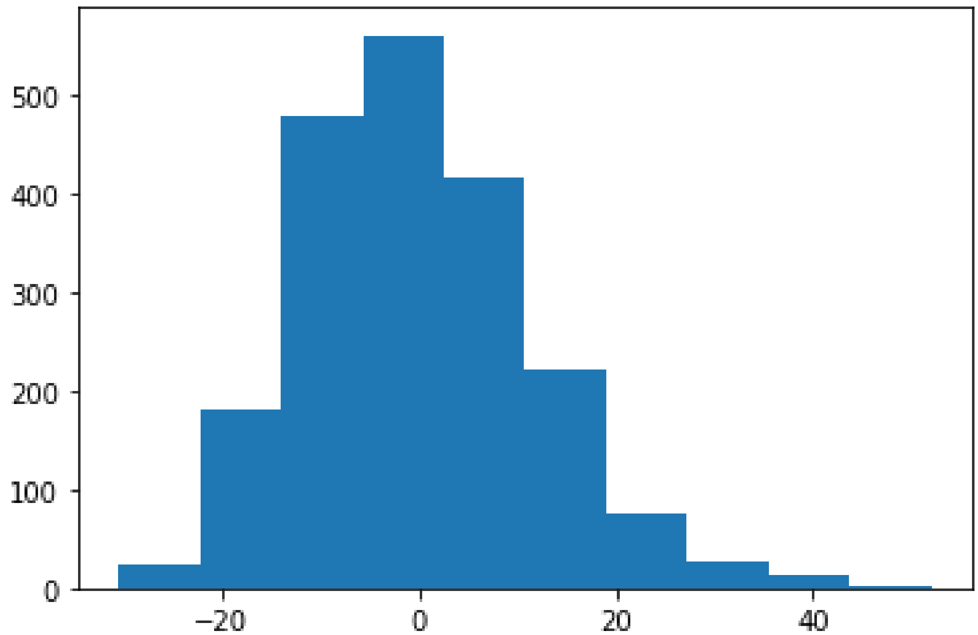
[34]:

```
plt.hist(sims)
print('Skewness', stats.skew(sims))
print('Excess Kurtosis', stats.kurtosis(sims))
print('Skewness of unconditional distribution', stats.skew(df.perc_log_return.
 ↪dropna()))
print('Excess Kurtosis of unconditional distribution', stats.kurtosis(df.
 ↪perc_log_return.dropna()))
pd.DataFrame(sims).describe()
```

```
Skewness 0.5856896148717768
Excess Kurtosis 0.6620021939924423
Skewness of unconditional distribution -0.29529864386112153
Excess Kurtosis of unconditional distribution 7.090355228425276
```

[34]:

	0
count	2000.000000
mean	0.022147
std	11.585573
min	-30.390858
25%	-8.348599
50%	-0.766949
75%	6.834866
max	51.957614



In our model there were two sources of asymmetric effects: the use of EGARCH instead of GARCH and the use of a skewed distribution for the innovation process, of which the former is the most important. To see what the effect of removing them is, we fit a GARCH model with normally distributed innovations to the process and examined the simulation results.

The resulting distributed is quite different. Most notably, the skewness for the fitted GARCH model is positive instead of negative. This shows that it is important to take into account the asymmetry of the NIC because without the skewness of the predicted distribution is likely incorrect.

[35]:
```
p = 1#best_params[0]
q = 1#best_params[1]
arma11 = ARMA(df.perc_log_return.dropna(), order=(p,q)) # Give the (p,q) order␣
↪of the ARMA(p,q) model
results = arma11.fit()
best_params=('EGARCH',1,1,1, 'skewt', 6)
am = arch_model(df.perc_log_return.dropna(), mean='AR', lags=best_params[5],␣
↪vol=best_params[0], p=best_params[1],o=best_params[2],q=best_params[3],␣
↪dist=best_params[4])
res = am.fit()  # Fit model

times = np.arange(1, 5000, 100)
sim_results = []
for time in times:
  sims = simulation([], df.perc_log_return[-time], res.resid[-time], res.
↪conditional_volatility[-time], R=100)
```

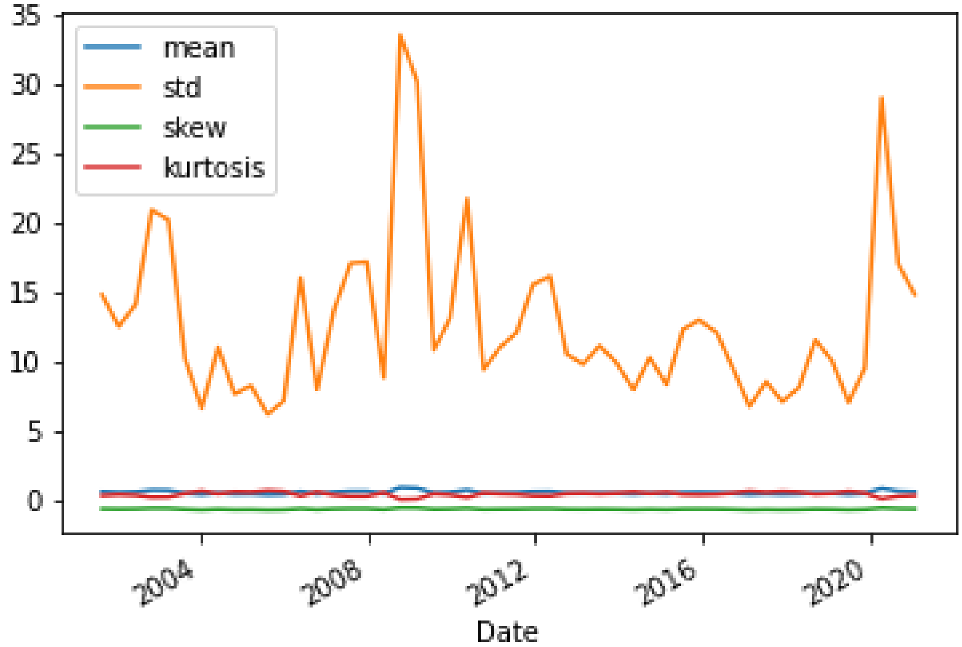
```
 sim_results.append({'time': np.array(df.reset_index('Date')['Date'])[-time],␣
↪'mean':np.mean(sims), 'std': np.std(sims), 'skew': stats.skew(sims),␣
↪'kurtosis': stats.kurtosis(sims)})
sim_results = pd.DataFrame(sim_results)
```

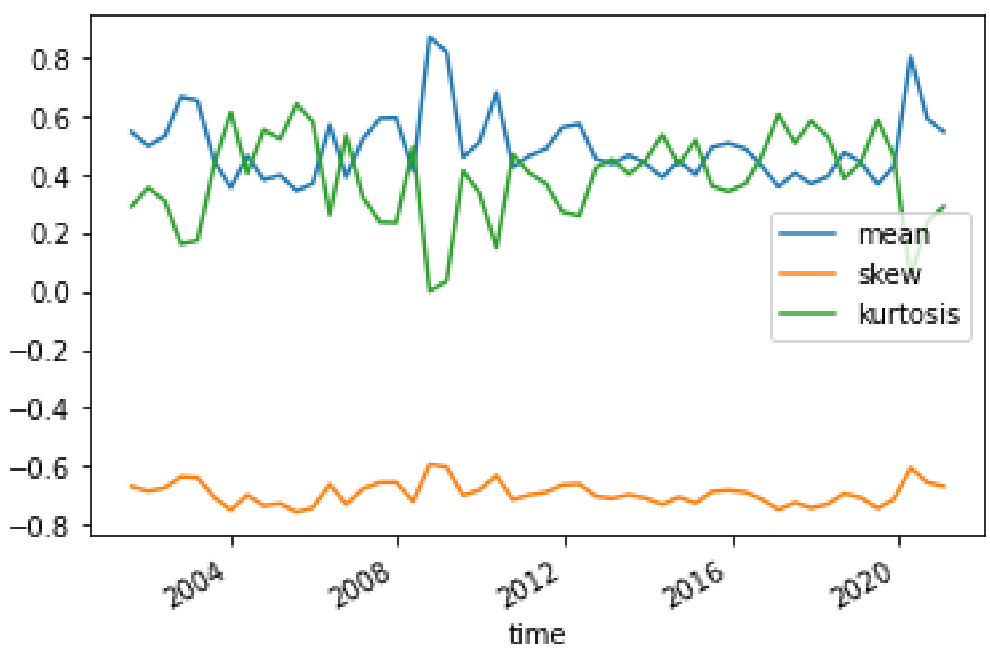
```
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

```
Iteration:      1,   Func. Count:     15,   Neg. LLF: 8045.042466633063
Iteration:      2,   Func. Count:     36,   Neg. LLF: 8022.7514790383675
Iteration:      3,   Func. Count:     55,   Neg. LLF: 8015.621331756515
Iteration:      4,   Func. Count:     74,   Neg. LLF: 8013.138877921244
Iteration:      5,   Func. Count:     93,   Neg. LLF: 8012.210440564211
Iteration:      6,   Func. Count:    112,   Neg. LLF: 8012.047505217472
Iteration:      7,   Func. Count:    131,   Neg. LLF: 8011.929663866598
Iteration:      8,   Func. Count:    148,   Neg. LLF: 8009.497834845549
Iteration:      9,   Func. Count:    164,   Neg. LLF: 7986.890852110325
Iteration:     10,   Func. Count:    182,   Neg. LLF: 7985.89970776809
Iteration:     11,   Func. Count:    201,   Neg. LLF: 7985.891352976838
Iteration:     12,   Func. Count:    219,   Neg. LLF: 7985.873153863778
Iteration:     13,   Func. Count:    236,   Neg. LLF: 7985.626395151281
Iteration:     14,   Func. Count:    252,   Neg. LLF: 7985.013754782052
Iteration:     15,   Func. Count:    269,   Neg. LLF: 7984.420115862295
Iteration:     16,   Func. Count:    285,   Neg. LLF: 7984.401984945364
Iteration:     17,   Func. Count:    300,   Neg. LLF: 7984.31375176232
Iteration:     18,   Func. Count:    315,   Neg. LLF: 7984.291944666487
Iteration:     19,   Func. Count:    330,   Neg. LLF: 7984.289364208852
Iteration:     20,   Func. Count:    345,   Neg. LLF: 7984.288931861026
Iteration:     21,   Func. Count:    360,   Neg. LLF: 7984.288680342083
Iteration:     22,   Func. Count:    377,   Neg. LLF: 7984.288676472373
Iteration:     23,   Func. Count:    393,   Neg. LLF: 7984.288667703696
Iteration:     24,   Func. Count:    409,   Neg. LLF: 7984.288665665798
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 7984.288665368191
            Iterations: 24
            Function evaluations: 411
            Gradient evaluations: 24
```

[41]:
```
sim_results.set_index('time').plot()
plt.xlabel("Date")
sim_results[['time', 'mean', 'skew', 'kurtosis']].set_index('time').plot()
```

[41]: <matplotlib.axes._subplots.AxesSubplot at 0x7f236243ee10>





We ran simulations for the monthly return distributions for different periods of consecutive 22 days with the starting days 100 days apart. To make running times reasonable, we did only 100 runs for each simulation point. Therefore, it should be taken into account that some features of these results could be due to the low number of runs, especially for the higher order moments.

We have plotted various statistics over time so that we can see how they depend on time. We see that the standard deviation has a lot of variation, which is of course caused by the conditional heteroskedasticity. Furthermore, we also see that the kurtosis is also dependent on time. Skewness seems to stay around zero although it does fluctuate a bit.

[37]:
```
ps = [1,2,3,4,5, 6]
qs = [1,2,3,4,5, 6]

best_aic = 20000
best_params = None
for p in ps:
  for q in qs:
    arma11 = ARMA(df.perc_log_return.dropna(), order=(p,q)) # Give the (p,q)␣
 ↪order of the ARMA(p,q) model
    try:
      res = arma11.fit()
      if res.aic < best_aic:
        best_aic = res.aic
        best_params = (p, q)
    except:
      # For some values of p, q, ARMA throws an error saying the process is not␣
 ↪stationary
      pass
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/tsatools.py:668:
RuntimeWarning: overflow encountered in exp
  newparams = ((1-np.exp(-params))/(1+np.exp(-params))).copy()
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/tsatools.py:668:
RuntimeWarning: invalid value encountered in true_divide
  newparams = ((1-np.exp(-params))/(1+np.exp(-params))).copy()
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/tsatools.py:669:
RuntimeWarning: overflow encountered in exp
  tmp = ((1-np.exp(-params))/(1+np.exp(-params))).copy()
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/tsatools.py:669:
RuntimeWarning: invalid value encountered in true_divide
  tmp = ((1-np.exp(-params))/(1+np.exp(-params))).copy()
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)
```

We used the code above to find the best hyperparameters for the ARMA model. To determine which one is better, we used the Akaike information index which makes a trade off between likelihood and number of parameters. This resulted in p=2 and q=5

[42]:
```
print(best_params)
p = best_params[0]
q = best_params[1]
arma11 = ARMA(df.perc_log_return.dropna(), order=(p,q)) # Give the (p,q) order
 ↪of the ARMA(p,q) model
results = arma11.fit()
print(results.summary2())
```

```
results.params
```

```
(2, 5)

/usr/local/lib/python3.6/dist-packages/statsmodels/tsa/base/tsa_model.py:219:
ValueWarning: A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.
  ' ignored when e.g. forecasting.', ValueWarning)

                            Results: ARMA
=====================================================================
Model:                ARMA             BIC:                 18684.1176
Dependent Variable:   perc_log_return  Log-Likelihood:      -9303.1
Date:                 2021-02-21 22:07 Scale:               1.0000
No. Observations:     5815             Method:              css-mle
Df Model:             8                Sample:              0
Df Residuals:         5807                                  5
Converged:            1.0000           S.D. of innovations: 1.198
No. Iterations:       26.0000          HQIC:                18644.978
AIC:                  18624.1038
---------------------------------------------------------------------
                       Coef.  Std.Err.    t    P>|t|   [0.025  0.975]
---------------------------------------------------------------------
const                  0.0037   0.0137  0.2677 0.7889 -0.0232  0.0306
ar.L1.perc_log_return  0.1050   0.1097  0.9571 0.3386 -0.1100  0.3199
ar.L2.perc_log_return -0.6854   0.1218 -5.6275 0.0000 -0.9241 -0.4467
ma.L1.perc_log_return -0.1303   0.1097 -1.1875 0.2351 -0.3454  0.0848
ma.L2.perc_log_return  0.6484   0.1225  5.2924 0.0000  0.4083  0.8885
ma.L3.perc_log_return -0.0753   0.0164 -4.5984 0.0000 -0.1074 -0.0432
ma.L4.perc_log_return  0.0184   0.0166  1.1084 0.2677 -0.0141  0.0509
ma.L5.perc_log_return -0.0799   0.0154 -5.2016 0.0000 -0.1100 -0.0498
-------------------------------------------------------------------------------
                  Real           Imaginary          Modulus          Frequency
-------------------------------------------------------------------------------
AR.1             0.0766            -1.2055           1.2079            -0.2399
AR.2             0.0766             1.2055           1.2079             0.2399
MA.1             0.1455            -1.2147           1.2234            -0.2310
MA.2             0.1455             1.2147           1.2234             0.2310
MA.3             2.0970            -0.0000           2.0970            -0.0000
MA.4            -1.0790            -1.6807           1.9973            -0.3408
MA.5            -1.0790             1.6807           1.9973             0.3408
=====================================================================
```

[42]:
```
const                    0.003677
ar.L1.perc_log_return    0.104970
ar.L2.perc_log_return   -0.685393
ma.L1.perc_log_return   -0.130304
```

```
ma.L2.perc_log_return    0.648384
ma.L3.perc_log_return   -0.075311
ma.L4.perc_log_return    0.018373
ma.L5.perc_log_return   -0.079872
dtype: float64
```

[38]:

```

```