Question 1

In [50]:
```
import numpy as np
import math
%matplotlib inline
import matplotlib.pyplot as plt
import scipy
import itertools
```

In [80]:
```
phis = [0, 0.5, 0.95, 0.99, 1]
Ts = [50, 100, 1000, 5000]

params = list(itertools.product(phis, Ts))
output = {}
for p, t in params:
    white_noise = np.random.normal(size=t-1)

    sample = [0]
    for i in range(t-1):
        sample.append(sample[-1]*p + white_noise[i])

    sample = np.array(sample)
    est_phi = np.sum(sample[1:]*sample[:-1])/np.sum(sample[1:]*sample[1:])
    std_phi = math.sqrt((1 - est_phi**2)/t)
    output[(p, t)] = (est_phi, std_phi)
```

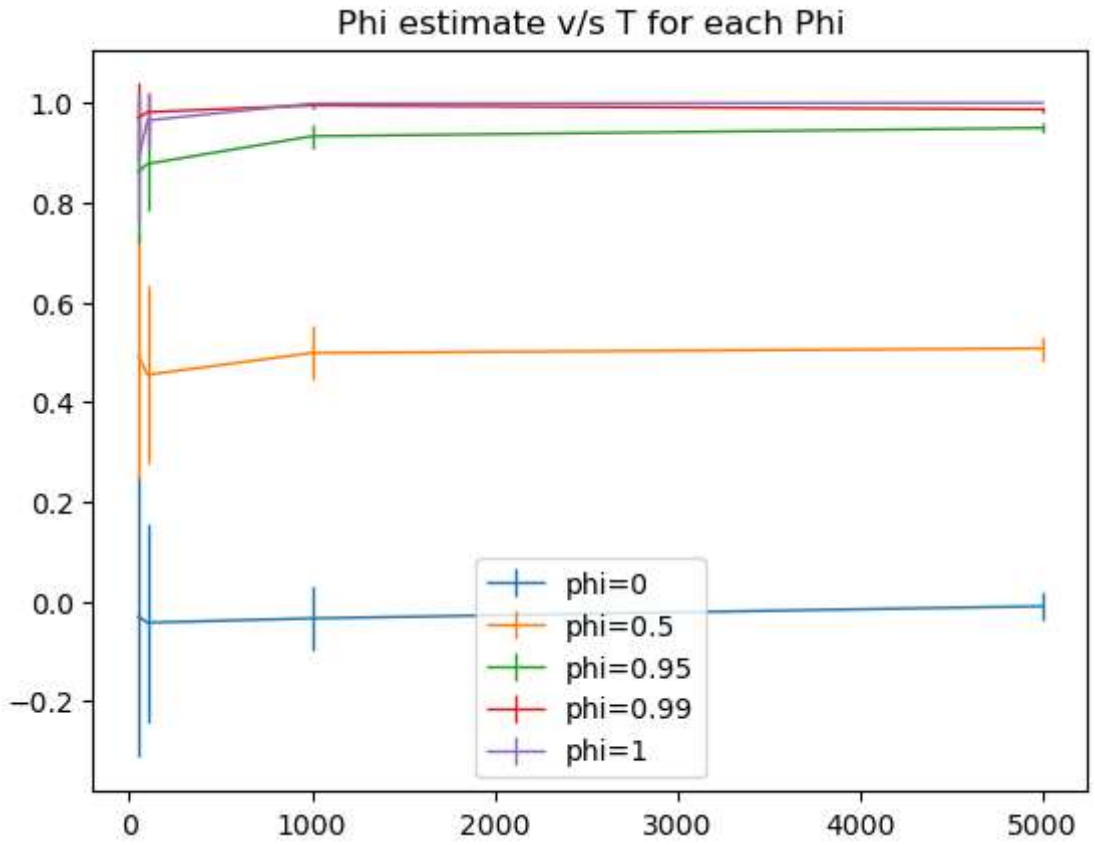
In [91]:
```
# (a), (b)
for p, t in output:
    o = output[(p, t)]
    print(f"Phi={p:.2f}, T={t:.4f} --> Est. Phi is {o[0]:.4f} with std {o[1]:.4f}")
```

```
Phi=0.00, T=50.0000 --> Est. Phi is -0.0307 with std 0.1414
Phi=0.00, T=100.0000 --> Est. Phi is -0.0427 with std 0.0999
Phi=0.00, T=1000.0000 --> Est. Phi is -0.0341 with std 0.0316
Phi=0.00, T=5000.0000 --> Est. Phi is -0.0098 with std 0.0141
Phi=0.50, T=50.0000 --> Est. Phi is 0.4924 with std 0.1231
Phi=0.50, T=100.0000 --> Est. Phi is 0.4543 with std 0.0891
Phi=0.50, T=1000.0000 --> Est. Phi is 0.4981 with std 0.0274
Phi=0.50, T=5000.0000 --> Est. Phi is 0.5071 with std 0.0122
Phi=0.95, T=50.0000 --> Est. Phi is 0.8615 with std 0.0718
Phi=0.95, T=100.0000 --> Est. Phi is 0.8773 with std 0.0480
Phi=0.95, T=1000.0000 --> Est. Phi is 0.9326 with std 0.0114
Phi=0.95, T=5000.0000 --> Est. Phi is 0.9494 with std 0.0044
Phi=0.99, T=50.0000 --> Est. Phi is 0.9696 with std 0.0346
Phi=0.99, T=100.0000 --> Est. Phi is 0.9811 with std 0.0194
Phi=0.99, T=1000.0000 --> Est. Phi is 0.9945 with std 0.0033
Phi=0.99, T=5000.0000 --> Est. Phi is 0.9866 with std 0.0023
Phi=1.00, T=50.0000 --> Est. Phi is 0.8875 with std 0.0652
Phi=1.00, T=100.0000 --> Est. Phi is 0.9643 with std 0.0265
Phi=1.00, T=1000.0000 --> Est. Phi is 0.9982 with std 0.0019
Phi=1.00, T=5000.0000 --> Est. Phi is 0.9996 with std 0.0004
```

In [94]:
```
# (c)
p_data = {p:np.array([output[(p, t)][0] for t in Ts]) for p in phis}
p_data1 = {p:np.array([output[(p, t)][1] for t in Ts]) for p in phis}

for p in p_data:
```

```
    plt.errorbar(Ts, p_data[p], yerr=p_data1[p]*2, linewidth=1, label=f"phi={p}")
plt.legend(loc='lower center')
plt.title("Phi estimate v/s T for each Phi")
```

Out[94]: Text(0.5, 1.0, 'Phi estimate v/s T for each Phi')



(d)

OLS is knows to be an unbiased estimator, but we saw in the lecture that it is biased in the case of AR timeseries, because the errors are not exogenous. Because of this the distribution is biased and this can be seen in the above plots.

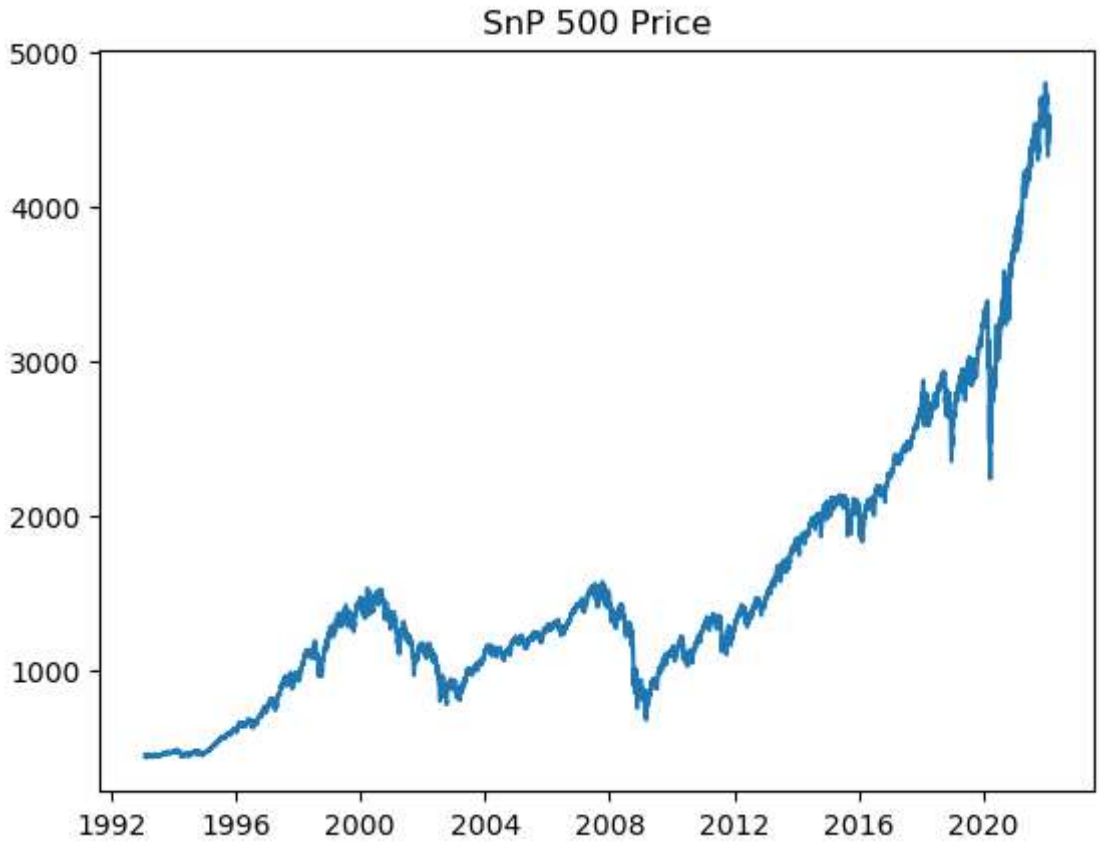
Question 2

In [95]:
```
import yfinance as yf
```

In [117…
```
df = yf.download('^GSPC', start='1993-02-01', end='2022-02-12')
price = np.array(df['Adj Close'])
log_price = np.log(price)
dates = np.array(df.index,dtype='datetime64[D]')
```

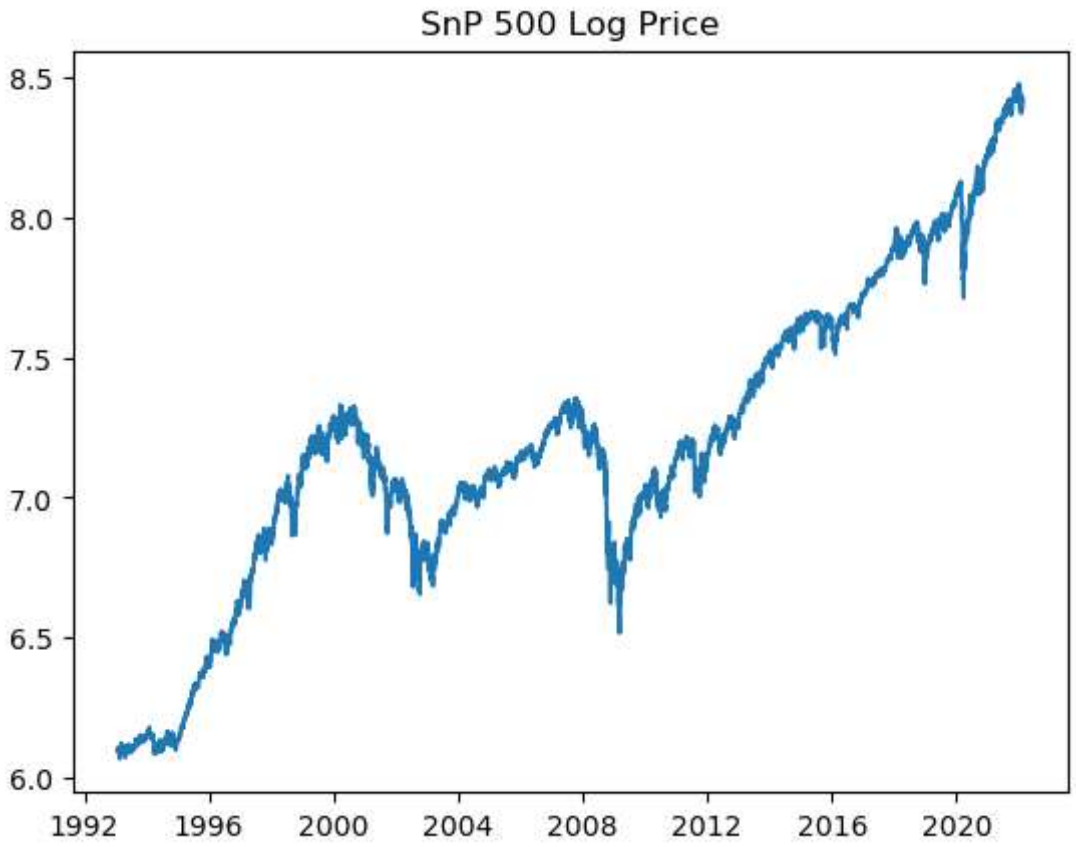
```
[**********100%**********]  1 of 1 completed
```

In [118…
```
#(a)
plt.plot(dates, price)
plt.title("SnP 500 Price")
plt.show()
```



In [120…

```
#(a)
plt.plot(dates, log_price)
plt.title("SnP 500 Log Price")
plt.show()
```

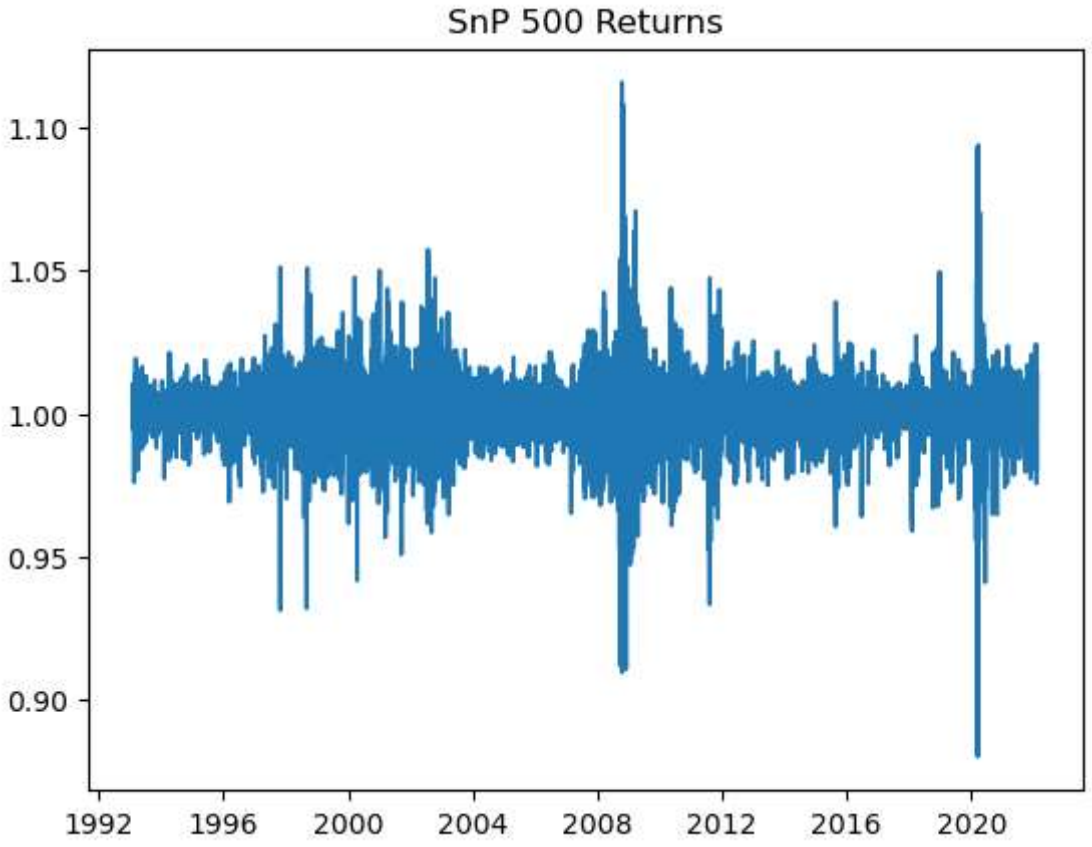


(a)

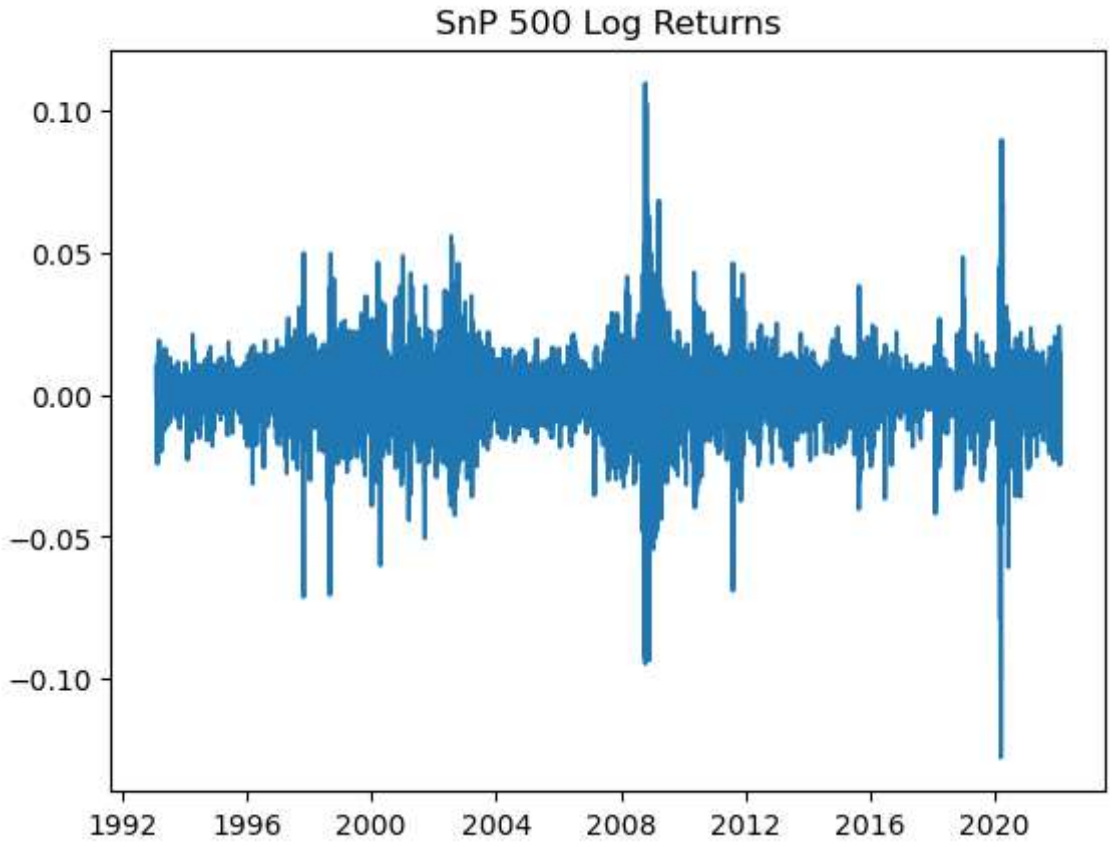
The prices seem to be exponentially increasing with a big departure from the trend between 2000 and 2010, similarly the log prices seem to be linearly increasing except for two big drawdowns between 2000 and 2010.

Log returns look closer to an AR process, with non-zero mean and coefficients which vary significantly over time, probably driven by big economic events.

In [336…
```
#(b)
returns = price[1:]/price[:-1]

plt.plot(dates[1:], returns)
plt.title("SnP 500 Returns")
plt.show()
```



In [337…
```
#(b)
log_returns = log_price[1:] - log_price[:-1]
plt.plot(dates[1:], log_returns)
plt.title("SnP 500 Log Returns")
plt.show()
```



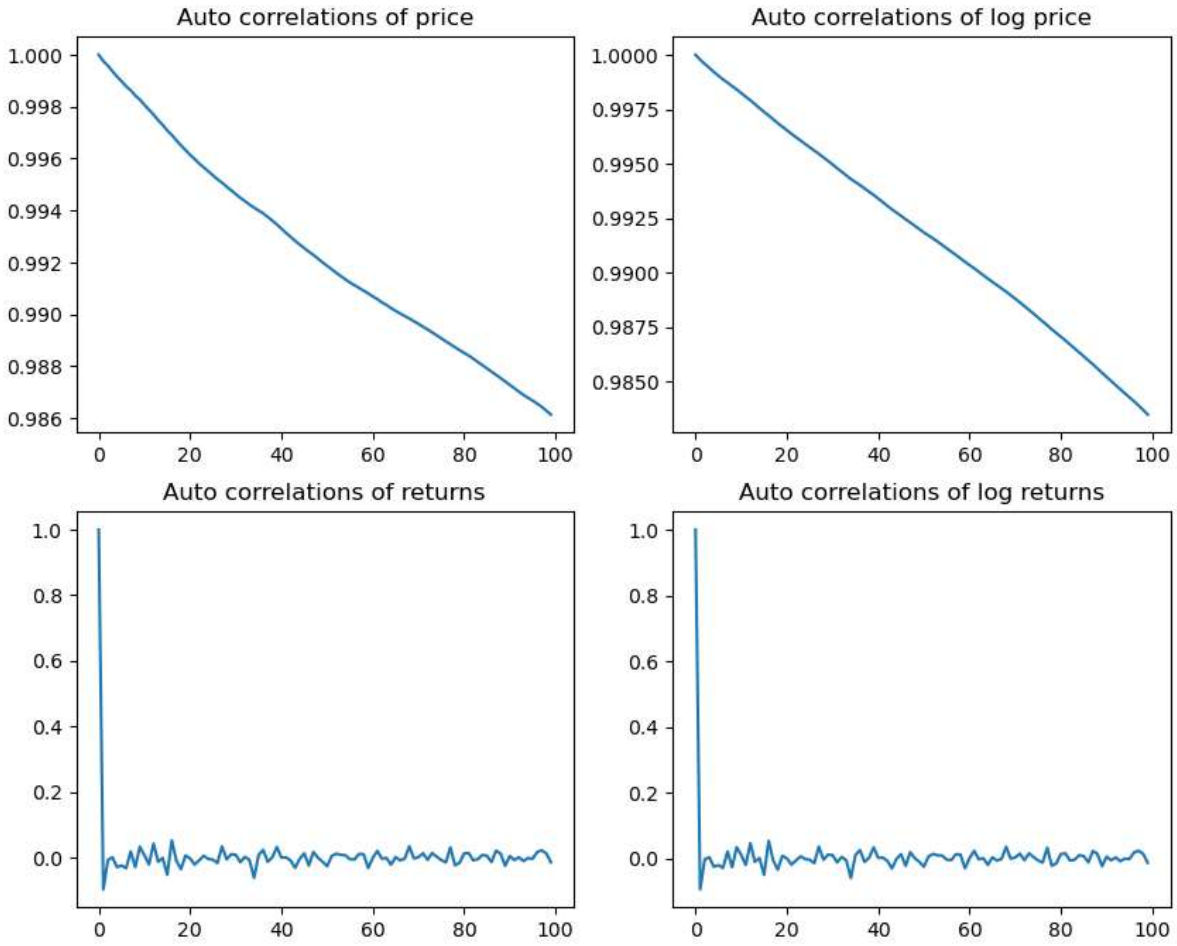
(b)

Returns and log returns both look like white noise with little to no patterns between returns. Returns have some big outliers and the variance also varies from one period to another.

In [340…
```
def acf(x, lag=1):
    return np.corrcoef(x[:-lag], x[lag:])[0, 1]

def acfs(x, length=100):
    return np.array([1]+[acf(x, i) for i in range(1, length)])
```

In [341…
```
fig, ax = plt.subplots(2, 2, figsize=(10,8))
ax[0, 0].plot(acfs(price))
ax[0, 0].set_title(f"Auto correlations of price")

ax[0, 1].plot(acfs(log_price))
ax[0, 1].set_title(f"Auto correlations of log price")

ax[1, 0].plot(acfs(returns))
ax[1, 0].set_title(f"Auto correlations of returns")

ax[1, 1].plot(acfs(log_returns))
ax[1, 1].set_title(f"Auto correlations of log returns")
```

Out[341]:
```
Text(0.5, 1.0, 'Auto correlations of log returns')
```



(c)

The auto correlation plots for prices decrease almost at a constant rate but it's hard to tell the exact shape of the curve. The important part is that the auto-correlations are very close to 1, though definitly less than 1. This again looks close to an AR process.

The returns behave more like white noise with very little anto-correlation. Though I haven't potted the first error-bars, the first auto-correlation seems to be significantly negative, which is line with our expectation for returns of financial assets.

In [145…

```
def ols_estimator(y, X, include_constant=True):

    N = X.shape[0]
    if include_constant:
        X_local = np.append(np.ones(N)[None, :].T, X, 1)
    else:
        X_local = np.array(X)

    xt_x_inv = np.linalg.inv(X_local.T@X_local)

    ols_beta = xt_x_inv@(X_local.T@y)
    point_estimates = ols_beta@X_local.T
    residuals = y - point_estimates

    error_vars = residuals*residuals
    error_variance_estimate = np.mean(error_vars)

    ols_variance_matrix = xt_x_inv * error_variance_estimate
    ols_stderr = np.sqrt(np.diag(ols_variance_matrix))
```

```
    D_mat = np.diag(error_vars)
    white_variance_matrix = xt_x_inv@(X_local.T@D_mat@X_local)@xt_x_inv
    white_stderr = np.sqrt(np.diag(white_variance_matrix))

    R_2 = 1 - error_variance_estimate/np.var(y)
    Adj_R_2 = 1 - (1 - R_2) * (N - 1) / (N - X.shape[1])

    max_log_likelyhood = -N/2*(math.log(2*math.pi) + 1) - N/2*math.log(error_variar

    P_mat = X_local@xt_x_inv@X_local.T
    M_mat = np.identity(N) - P_mat

    # (f) Verified ex=0
    np.allclose(residuals @ X, 0)

    # (g) Qxx = E(x_i x_i')
    condition_number = np.linalg.cond(X_local) ## Close to singular

    output = {'N': N,
              'betas': ols_beta,
              'point_estimates': point_estimates,
              'residuals': residuals,
              'ols_variance_matrix': ols_variance_matrix,
              'ols_stderr': ols_stderr,
              'white_variance_matrix': white_variance_matrix,
              'white_stderr': white_stderr,
              'R_square': R_2,
              'R_square_adj': Adj_R_2,
              'max_log_likelyhood': max_log_likelyhood,
              'P': P_mat,
              'M': M_mat,
              'condition_number': condition_number
             }


    return output
```

Starting with log returns

I ran all AR(0), AR(1) and AR(2) models but I only show AR(1) because it turned out to be the best on based on variors factors such as the BIC and adjusted R^2 and more. This is mostly as close judgement call because both AR(1) and AR(2) don't perform very well and having fewer predictors takes the cake.

In [278…

```
ar_p = 1
inp = log_returns

Y = inp[ar_p:]
X = np.ones(len(inp) - ar_p)[None, :].T
if ar_p == 1:
    X = np.append(X, inp[:-ar_p][None, :].T, 1)
    pass
if ar_p == 2:
    X = np.append(X, inp[1:-ar_p+1][None, :].T, 1)
    X = np.append(X, inp[:-ar_p][None, :].T, 1)

X.shape, Y.shape
```

Out[278]: ((7311, 2), (7311,))

In [279…
```
output = ols_estimator(Y,  X, include_constant=False)
```

In [280…
```
print(f"(a) Beta: {output['betas']}")
print(f"(a) R Square: {output['R_square']}")
print(f"(a) Adjusted R Square {output['R_square_adj']}")
```

```
(a) Beta: [ 0.00034472 -0.0944791 ]
(a) R Square: 0.008922911356127217
(a) Adjusted R Square 0.0087873145455315
```

In [281…
```
#(c)
print("**Variance-covariance matrix under white**\n")
print(output['white_variance_matrix'])
```

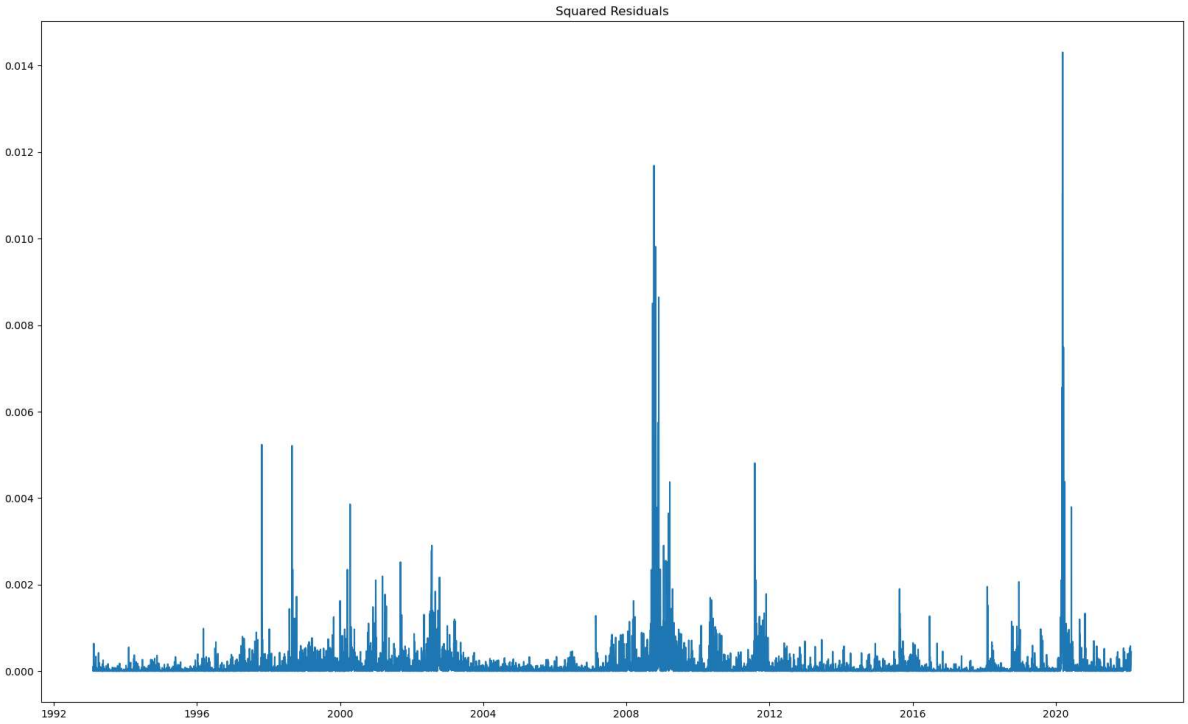
```
**Variance-covariance matrix under white**

[[ 1.89217163e-08 -7.50388195e-07]
 [-7.50388195e-07  6.30852548e-04]]
```

In [282…
```
#(d)
TEST_ALPHAS = [0.01, 0.05, 0.1]

def t_test(b_cap, b_o, stderr, N):
    t_value = np.abs((b_cap - b_o)/stderr)
    reject = {}

    for alpha in TEST_ALPHAS:
        critical_value = scipy.stats.t.ppf(1 - alpha, df=N)
        reject[alpha] = t_value > critical_value
    return t_value, reject

def print_test_output(t_output, value):
    print(f"t values: {t_output[0]}")
    for alpha in t_output[1]:
        out = t_output[1][alpha]
        print(f"Test beta={value} rejected at alpha={alpha*100:2.0f}%: {out}")

print("**Running T-test**\n")

print_test_output(t_test(output['betas'], 0, output['white_stderr'], output['N']),
```

```
**Running T-test**

t values: [2.50603108 3.76159301]
Test beta=0 rejected at alpha= 1%: [ True  True]
Test beta=0 rejected at alpha= 5%: [ True  True]
Test beta=0 rejected at alpha=10%: [ True  True]
```

In [283…
```
#(e)
fig, ax = plt.subplots(1, figsize=(20,12))
ax.plot(dates[:-2], output['residuals']**2)
ax.set_title(f"Squared Residuals")
fig.show()
```

```
/tmp/ipykernel_2187/1775103293.py:5: UserWarning: Matplotlib is currently using mo
dule://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot sho
w the figure.
  fig.show()
```



There are clear periods of high and low variance in the data. So it is heteroskedastic.

In [284...
```
# (j)
print("**AIC, BIC, HQ**\n")
k = 2
print(f"AIC: {-2*output['max_log_likelyhood'] + 2*k}")
print(f"BIC: {-2*output['max_log_likelyhood'] + k*math.log(output['N'])}")
print(f"HQ: {-2*output['max_log_likelyhood'] + k*math.log(math.log(output['N']))}")
```

```
**AIC, BIC, HQ**

AIC: -44373.75795230543
BIC: -44359.963681620844
HQ: -44373.38649359888
```

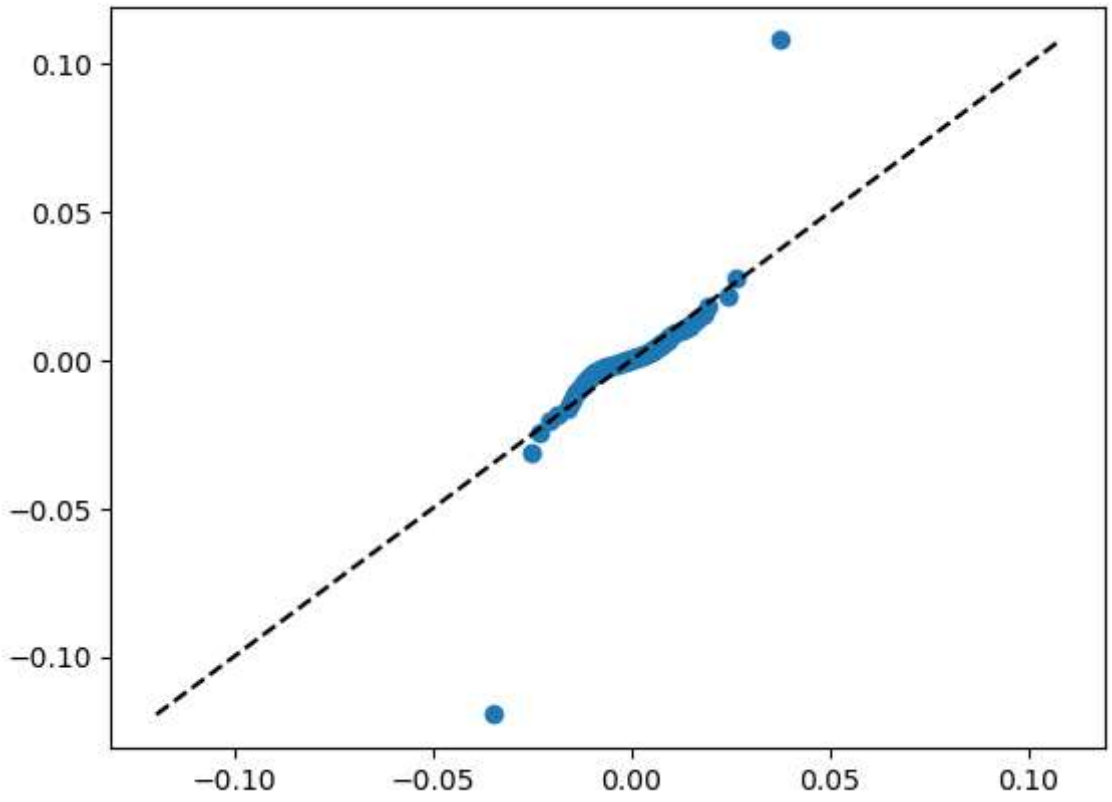
In [285...
```
#(k)
def compute_wd(residuals):
    return np.sum((residuals[1:] - residuals[:-1])**2)/np.sum(residuals[1:]**2)

def compute_breusch_godfrey(residuals, N):
    bg_output = ols_estimator(residuals[1:],  residuals[:-1][None, :].T)
    return (N - 1)*bg_output['R_square']

print(f"DW: {compute_wd(output['residuals'])}")
bg = compute_breusch_godfrey(output['residuals'], output['N'])

print(f"Breusch Godfrey: {bg} with p value {1- scipy.stats.chi2.cdf(bg, 1)}")
```

```
DW: 2.001999469049835
Breusch Godfrey: 0.010107539115340458 with p value 0.9199185991492385
```

The p value of Breusch Godfrey says there is no auto correlation in the errors

In [286...
```
#(l)
a = np.random.normal(0,np.std(output['residuals']),250)
b = output['residuals']

percs = np.linspace(0,100,80)
qn_a = np.percentile(a, percs)
```

```
qn_b = np.percentile(b, percs)

plt.plot(qn_a,qn_b, ls="", marker="o")

x = np.linspace(np.min((qn_a.min(),qn_b.min())), np.max((qn_a.max(),qn_b.max())))
plt.plot(x,x, color="k", ls="--")

plt.show()
```



In [287…
```
def calculate_jb(residuals, N):
    kurt = scipy.stats.kurtosis(residuals)
    skew = scipy.stats.skew(residuals)
    return N*((skew**2) + ((kurt)**2)/4)/6

jb = calculate_jb(output['residuals'], output['N'])

print(f"JB is {jb} with p value of {1- scipy.stats.chi2.cdf(jb, 2)}")
```

```
JB is 35679.72972690094 with p value of 0.0
```

The erros are not normally distributed based on the qq plot and jb value

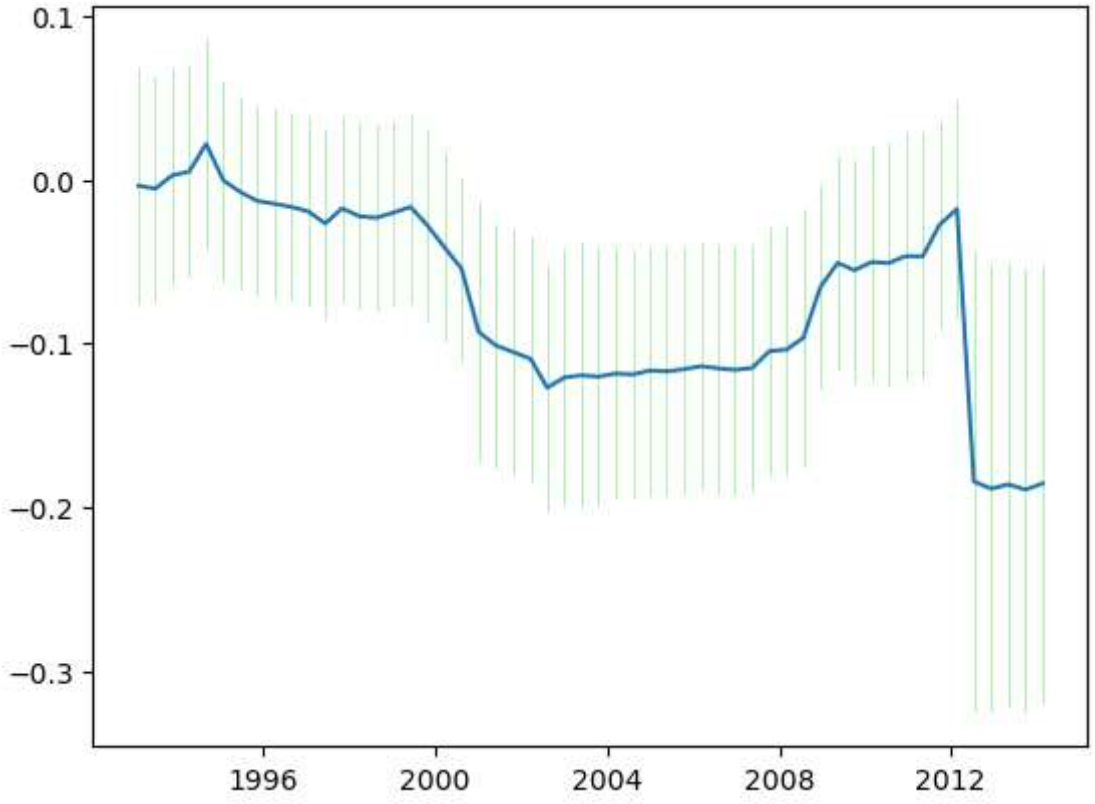
In [288…
```
# (m)
output['condition_number']
```

Out[288]:
```
85.59665189954043
```

Condition number is low, showing no multicollinearity

In [290…
```
# (o)
x_arr = []
beta_arr = []
error_array = []
for i in range(X.shape[0] - 2000):
    if i % 100 != 0:
```

```
        continue
    x = dates[i]
    b_out = ols_estimator(Y[i:2000+i],  X[i:2000+i, :], include_constant=False)
    beta = b_out['betas'][1]
    error = b_out['white_stderr'][1]
    critical_value = scipy.stats.t.ppf(0.975, df=b_out['N'])
    bar = critical_value*error

    x_arr.append(x)
    beta_arr.append(beta)
    error_array.append(bar)

# low = beta - critical_value*error

plt.errorbar(x_arr,beta_arr, yerr=error_array, ecolor = 'lightgreen', elinewidth =
plt.show()
```



Beta does have seasonal patterns and changes significantly over time

Running on log prices

I ran all AR(0), AR(1) and AR(2) models but I again only show AR(2) because it turned out to be the best on based on variors factors such as the BIC and adjusted R^2 and more. This is a close judgement call because both AR(1) and AR(2) perform almost equally well but the additional coefficient is statistically significant and gives lower BIC.

In [322…

```
ar_p = 2
inp = log_price

Y = inp[ar_p:]
X = np.ones(len(inp) - ar_p)[None, :].T
if ar_p == 1:
    X = np.append(X, inp[:-ar_p][None, :].T, 1)
```

```
    pass
if ar_p == 2:
    X = np.append(X, inp[1:-ar_p+1][None, :].T, 1)
    X = np.append(X, inp[:-ar_p][None, :].T, 1)

X.shape, Y.shape
```

Out[322]: ((7311, 3), (7311,))

In [323…
```
output = ols_estimator(Y,  X, include_constant=False)
```

In [324…
```
print(f"(a) Beta: {output['betas']}")
print(f"(a) R Square: {output['R_square']}")
print(f"(a) Adjusted R Square {output['R_square_adj']}")
```

```
(a) Beta: [0.00142612 0.90546618 0.0943835 ]
(a) R Square: 0.9995457973002025
(a) Adjusted R Square 0.999545672997329
```

In [325…
```
#(c)
print("**Variance-covariance matrix under white**\n")
print(output['white_variance_matrix'])
```

```
**Variance-covariance matrix under white**

[[ 2.42685029e-06 -1.82333058e-06  1.48962544e-06]
 [-1.82333058e-06  6.30878662e-04 -6.30729683e-04]
 [ 1.48962544e-06 -6.30729683e-04  6.30626949e-04]]
```

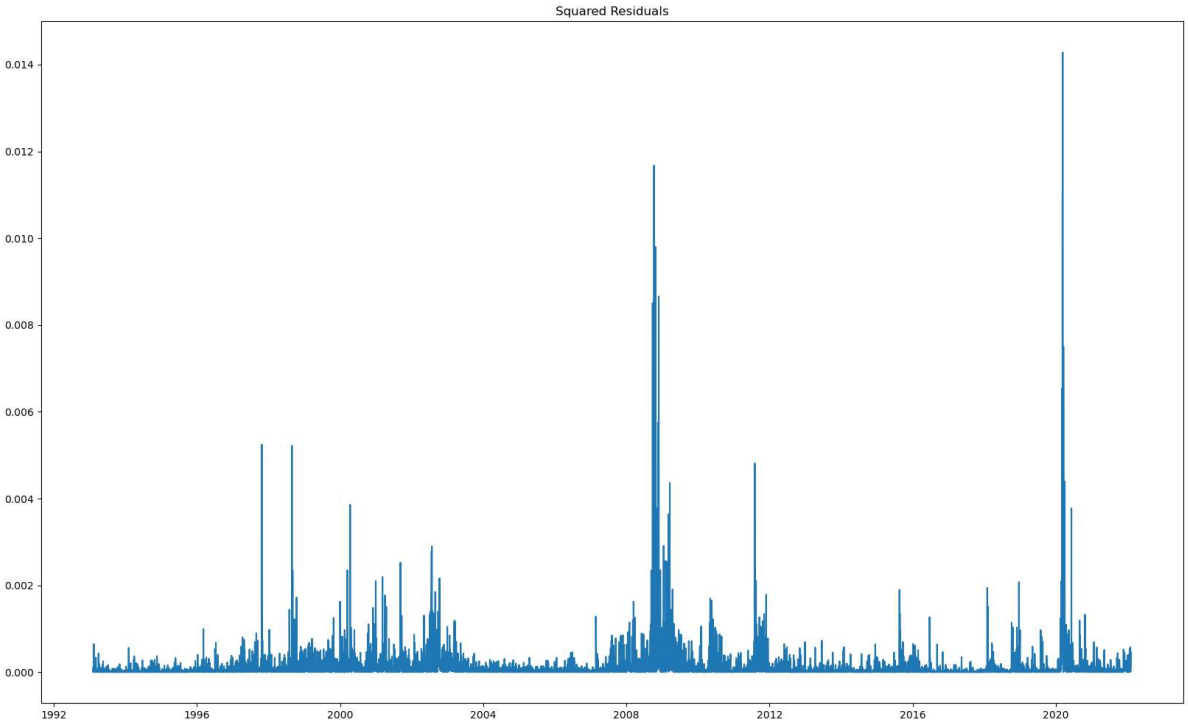
In [326…
```
#(d)
TEST_ALPHAS = [0.01, 0.05, 0.1]

def t_test(b_cap, b_o, stderr, N):
    t_value = np.abs((b_cap - b_o)/stderr)
    reject = {}

    for alpha in TEST_ALPHAS:
        critical_value = scipy.stats.t.ppf(1 - alpha, df=N)
        reject[alpha] = t_value > critical_value
    return t_value, reject

def print_test_output(t_output, value):
    print(f"t values: {t_output[0]}")
    for alpha in t_output[1]:
        out = t_output[1][alpha]
        print(f"Test beta={value} rejected at alpha={alpha*100:2.0f}%: {out}")

print("**Running T-test**\n")

print_test_output(t_test(output['betas'], 0, output['white_stderr'], output['N']),
```

```
**Running T-test**

t values: [ 0.91544938 36.04950572  3.75845907]
Test beta=0 rejected at alpha= 1%: [False  True  True]
Test beta=0 rejected at alpha= 5%: [False  True  True]
Test beta=0 rejected at alpha=10%: [False  True  True]
```

In [327…
```
#(e)
fig, ax = plt.subplots(1, figsize=(20,12))
ax.plot(dates[:-2], output['residuals']**2)
```

```
ax.set_title(f"Squared Residuals")
fig.show()
```

```
/tmp/ipykernel_2187/1775103293.py:5: UserWarning: Matplotlib is currently using mo
dule://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot sho
w the figure.
  fig.show()
```



There are clear periods of high and low variance in the data. So it is heteroskedastic.

In [328…

```
# (j)
print("**AIC, BIC, HQ**\n")
k = 2
print(f"AIC: {-2*output['max_log_likelyhood'] + 2*k}")
print(f"BIC: {-2*output['max_log_likelyhood'] + k*math.log(output['N'])}")
print(f"HQ: {-2*output['max_log_likelyhood'] + k*math.log(math.log(output['N']))}")
```

```
**AIC, BIC, HQ**

AIC: -44374.12154054141
BIC: -44360.327269856825
HQ: -44373.75008183486
```

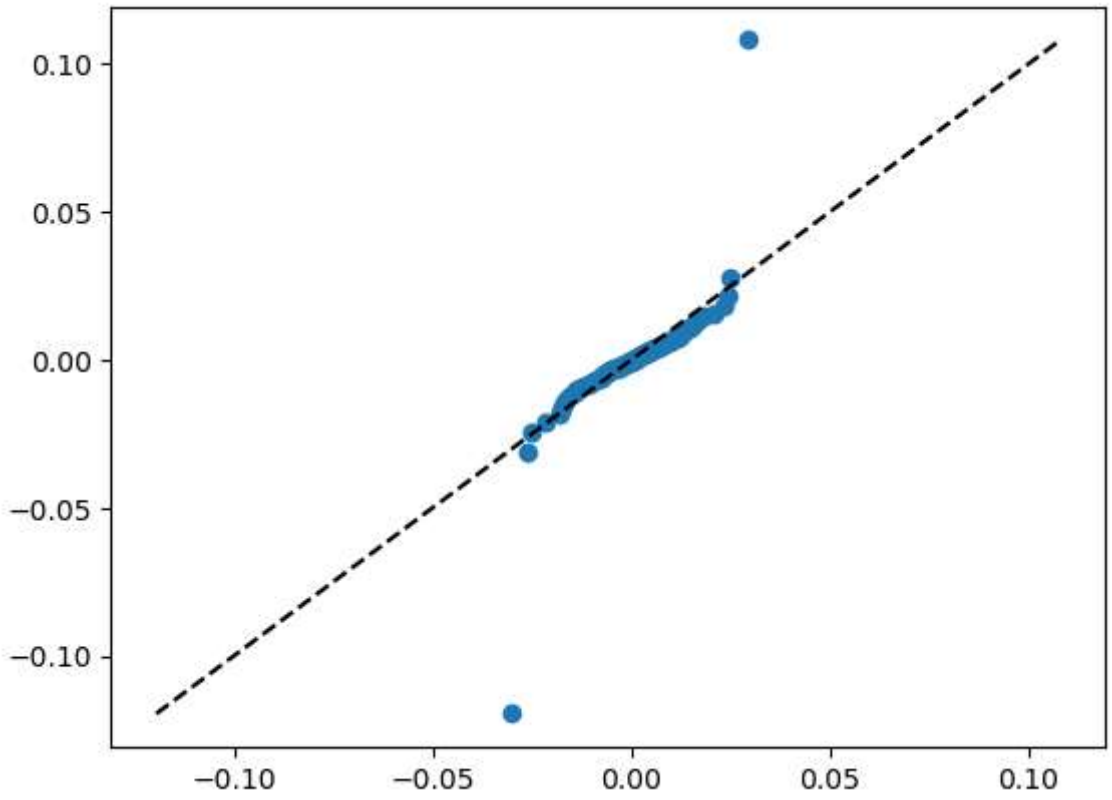
In [329…

```
#(k)
def compute_wd(residuals):
    return np.sum((residuals[1:] - residuals[:-1])**2)/np.sum(residuals[1:]**2)

def compute_breusch_godfrey(residuals, N):
    bg_output = ols_estimator(residuals[1:],  residuals[:-1][None, :].T)
    return (N - 1)*bg_output['R_square']

print(f"DW: {compute_wd(output['residuals'])}")
bg = compute_breusch_godfrey(output['residuals'], output['N'])

print(f"Breusch Godfrey: {bg} with p value {1- scipy.stats.chi2.cdf(bg, 1)}")
```

```
DW: 2.0019806056242873
Breusch Godfrey: 0.00990742037581871 with p value 0.9207126856837395
```

The p value of Breusch Godfrey says there is no auto correlation in the errors

In [330…
```
#(l)
a = np.random.normal(0,np.std(output['residuals']),250)
b = output['residuals']

percs = np.linspace(0,100,80)
qn_a = np.percentile(a, percs)
qn_b = np.percentile(b, percs)

plt.plot(qn_a,qn_b, ls="", marker="o")

x = np.linspace(np.min((qn_a.min(),qn_b.min())), np.max((qn_a.max(),qn_b.max())))
plt.plot(x,x, color="k", ls="--")

plt.show()
```



In [331…
```
def calculate_jb(residuals, N):
    kurt = scipy.stats.kurtosis(residuals)
    skew = scipy.stats.skew(residuals)
    return N*((skew**2) + ((kurt)**2)/4)/6

jb = calculate_jb(output['residuals'], output['N'])

print(f"JB is {jb} with p value of {1- scipy.stats.chi2.cdf(jb, 2)}")
```

```
JB is 35624.06367854314 with p value of 0.0
```

The erros are not normally distributed based on the qq plot and jb value

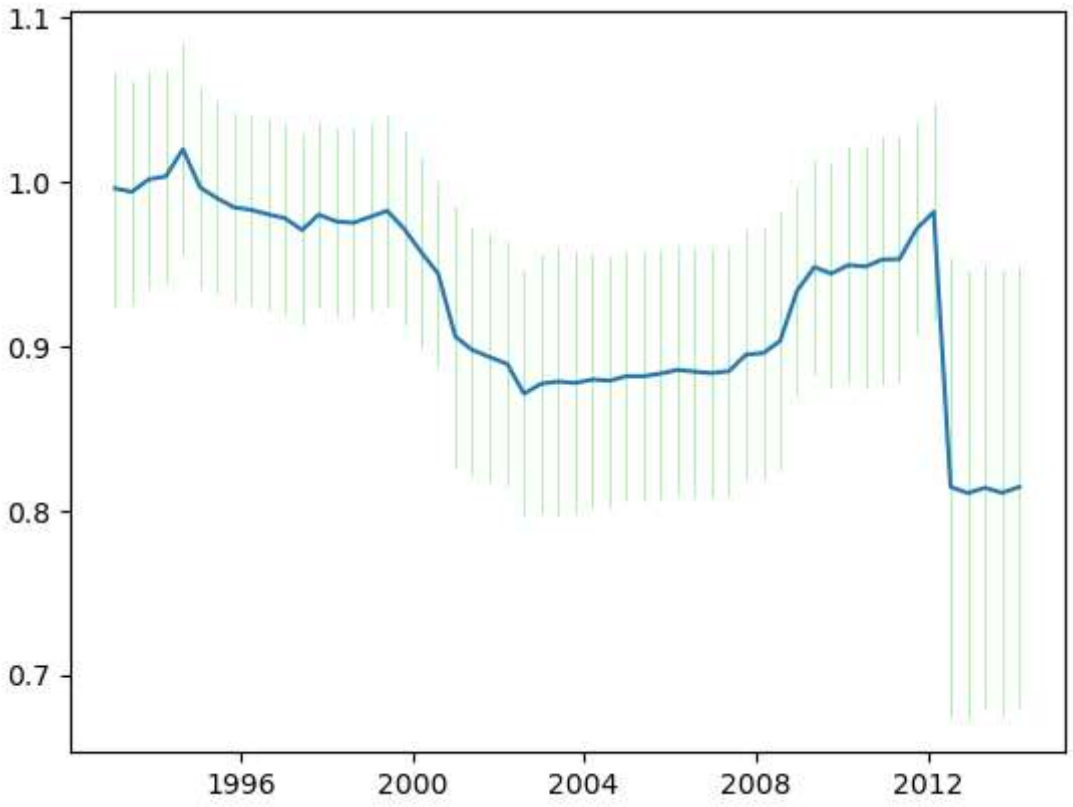
In [332…
```
# (m)
output['condition_number']
```

Out[332]:
```
1241.0171992995388
```

Condition number is high, showing multicollinearity in the predictors. This is expected given the high auto correlation in the 1-lagged and 2-lagged prices.

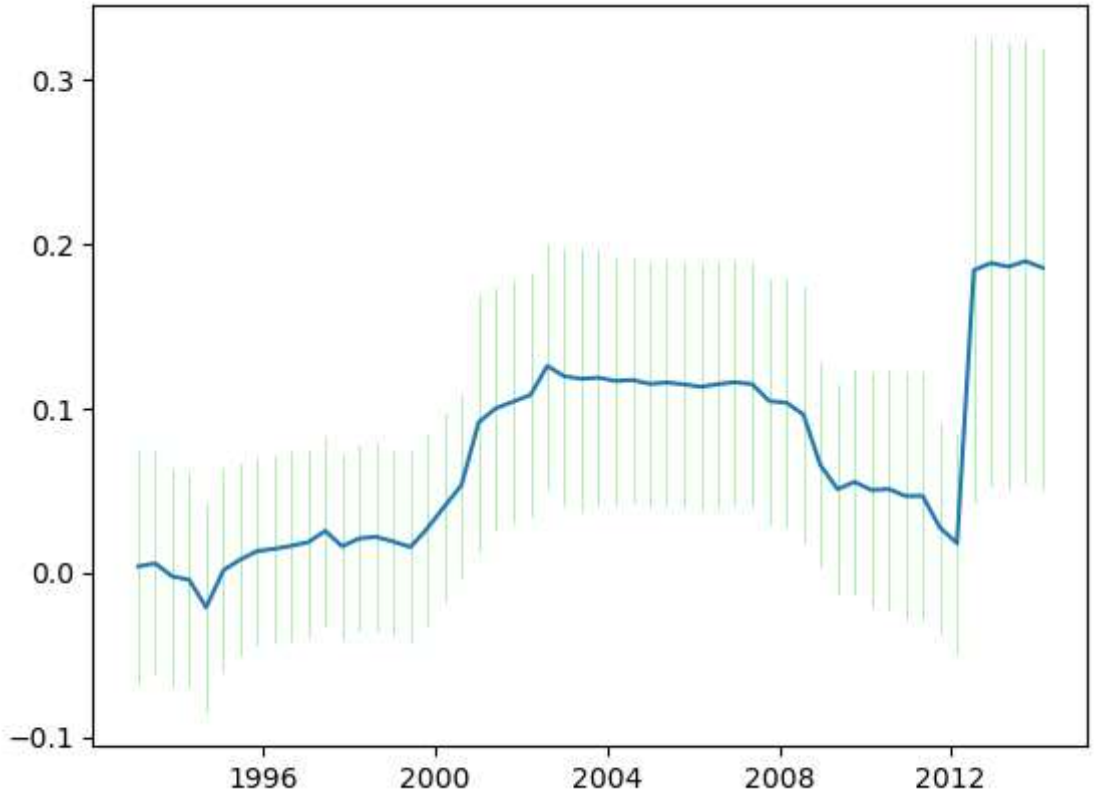
In [334…

```
# (o)
x_arr = []
beta1_arr = []
error1_array = []

beta2_arr = []
error2_array = []
for i in range(X.shape[0] - 2000):
    if i % 100 != 0:
        continue
    x = dates[i]
    b_out = ols_estimator(Y[i:2000+i],  X[i:2000+i, :], include_constant=False)
    beta1 = b_out['betas'][1]
    error1 = b_out['white_stderr'][1]
    beta2 = b_out['betas'][2]
    error2 = b_out['white_stderr'][2]
    critical_value = scipy.stats.t.ppf(0.975, df=b_out['N'])

    x_arr.append(x)
    beta1_arr.append(beta1)
    error1_array.append(critical_value*error1)
    beta2_arr.append(beta2)
    error2_array.append(critical_value*error2)

# low = beta - critical_value*error

plt.errorbar(x_arr,beta1_arr, yerr=error1_array, ecolor = 'lightgreen', elinewidth
plt.show()
```



In [335…

```
plt.errorbar(x_arr,beta2_arr, yerr=error2_array, ecolor = 'lightgreen', elinewidth
plt.show()
```



Both betas have seasonal patterns and changes significantly over time