Question 1

In [24]:
```
import numpy as np
import math
%matplotlib inline
import matplotlib.pyplot as plt
import scipy
```

In [133…
```
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


    # (e) Verified that PM = 0
    assert np.allclose(P_mat@M_mat, 0)

    # (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
```

In [137…
```
data = np.genfromtxt('Data_PG_UN.csv', delimiter=',', skip_header=1,
                     dtype='datetime64[D],f,f,f,f,f', names='Date,PG,UN,HH,MKT,RF')
```

In [138…
```
X = data['MKT'] - data['RF']
Y = data['UN'] - data['RF']
```

In [140…
```
print("\n**Q1. Running OLS on MT and UL**\n")

output = ols_estimator(Y,  X[None, :].T)
```

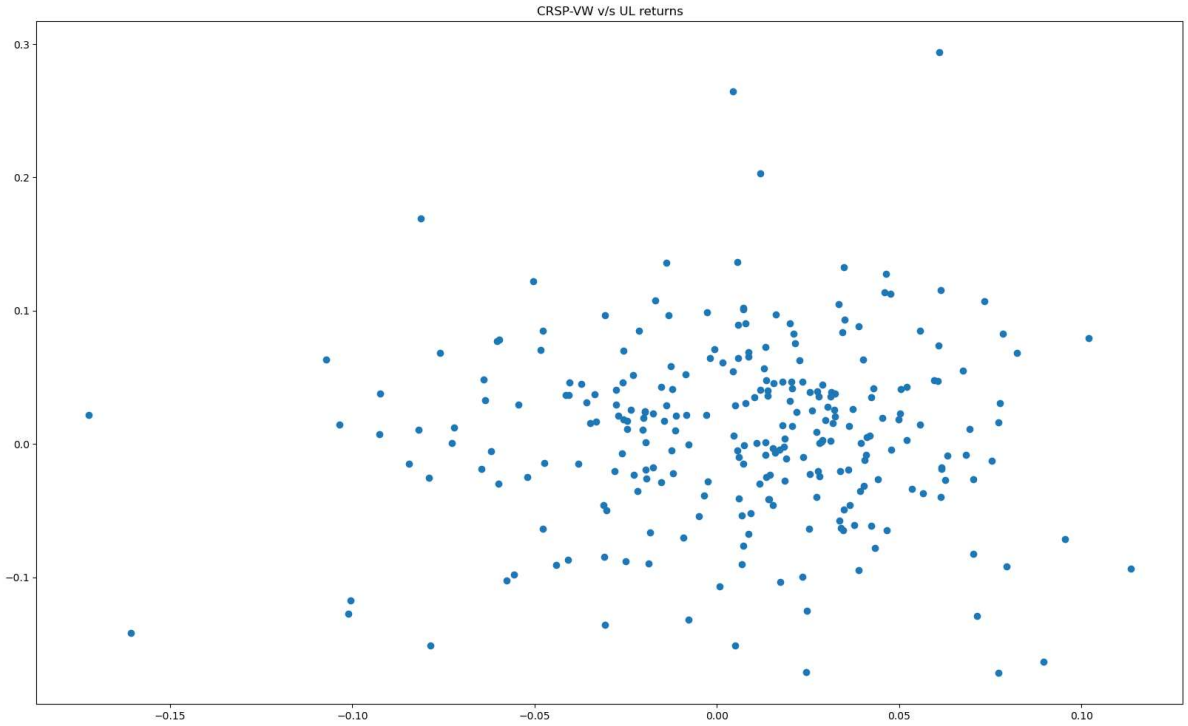
```
**Q1. Running OLS on MT and UL**
```

In [141…
```
print(f"(a) Alpha: {output['betas'][0]}")
print(f"(a) Beta: {output['betas'][1]}")
print(f"(a) R Square: {output['R_square']}")
print(f"(a) Adjusted R Square {output['R_square_adj']}")
```

```
(a) Alpha: 0.008043473475692635
(a) Beta: 0.05242949391999781
(a) R Square: 0.001177492295487581
(a) Adjusted R Square 0.001177492295487581
```

In [142…
```
#(b)
fig, ax = plt.subplots(1, figsize=(20,12))
ax.scatter(X, Y)
ax.set_title(f"CRSP-VW v/s UL returns")
fig.show()
```

```
/tmp/ipykernel_6598/3342360173.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()
```



In [143…
```
#(c)
print("**Variance-covariance matrix under homoskedasticity**\n")
print(output['ols_variance_matrix'])
```

Variance-covariance matrix under homoskedasticity

[[1.74836363e-05 -5.37728491e-05]
 [-5.37728491e-05 8.83167379e-03]]

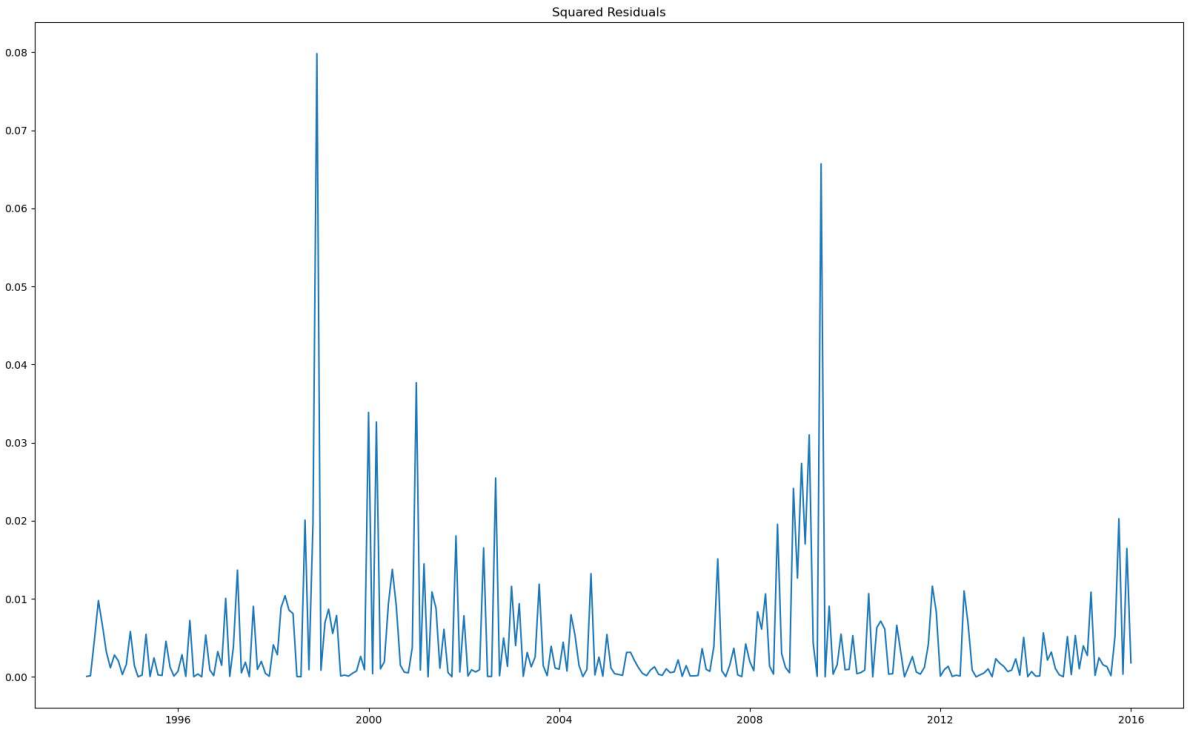
In [144…
```
#(d)
TEST_ALPHAS = [0.01, 0.05, 0.1]

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['ols_stderr'], output['N']), 0)
```

Running T-test

t values: [1.92365751 0.55789717]
Test beta=0 rejected at alpha= 1%: [False False]
Test beta=0 rejected at alpha= 5%: [True False]
Test beta=0 rejected at alpha=10%: [True False]

In [145…
```
#(e)
fig, ax = plt.subplots(1, figsize=(20,12))
ax.plot(data['Date'], output['residuals']**2)
ax.set_title(f"Squared Residuals")
fig.show()
```

```
/tmp/ipykernel_6598/1734843182.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 [146…
```
def confidence_intervals(b_cap, stderr, N):
    for alpha in TEST_ALPHAS:
        critical_value = scipy.stats.t.ppf(1 - alpha/2, df=N)
        high = critical_value*stderr + b_cap
        low = b_cap - critical_value*stderr
        print(f"{(1 - alpha)*100:.0f}% confidence interval alpha is [{low[0]:.4f},
        print(f"{(1 - alpha)*100:.0f}% confidence interval beta is [{low[1]:.4f},
```

In [147…
```
#(f)
print(f"Standard errors are {output['ols_stderr']} under homoskedasticity")
confidence_intervals(output['betas'], output['ols_stderr'], output['N'])
```

```
Standard errors are [0.00418134 0.09397699] under homoskedasticity
99% confidence interval alpha is [-0.0028, 0.0189]
99% confidence interval beta is [-0.1914, 0.2963]
95% confidence interval alpha is [-0.0002, 0.0163]
95% confidence interval beta is [-0.1326, 0.2375]
90% confidence interval alpha is [0.0011, 0.0149]
90% confidence interval beta is [-0.1027, 0.2076]
```

In [148…
```
#(g)
print("**Variance-covariance matrix under hetroskedasticity**\n")
print(output['white_variance_matrix'])
```

```
**Variance-covariance matrix under hetroskedasticity**

[[ 1.73373723e-05 -5.30367406e-05]
 [-5.30367406e-05  1.25353337e-02]]
```

In [149…
```
#(h)
print(f"Standard errors are {output['white_stderr']} using white")
confidence_intervals(output['betas'], output['white_stderr'], output['N'])
```

```
Standard errors are [0.00416382 0.1119613 ] using white
99% confidence interval alpha is [-0.0028, 0.0188]
99% confidence interval beta is [-0.2381, 0.3429]
95% confidence interval alpha is [-0.0002, 0.0162]
95% confidence interval beta is [-0.1680, 0.2729]
90% confidence interval alpha is [0.0012, 0.0149]
90% confidence interval beta is [-0.1324, 0.2372]
```

In [150…
```
# (i)
print_test_output(t_test(output['betas'], 0, output['white_stderr'], output['N']),
```

```
t values: [1.93175479 0.46828227]
Test beta=0 rejected at alpha= 1%: [False False]
Test beta=0 rejected at alpha= 5%: [ True False]
Test beta=0 rejected at alpha=10%: [ True False]
```

In [151…
```
# (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: -671.6612307946733
BIC: -664.5093325883806
HQ: -672.2243057035266
```

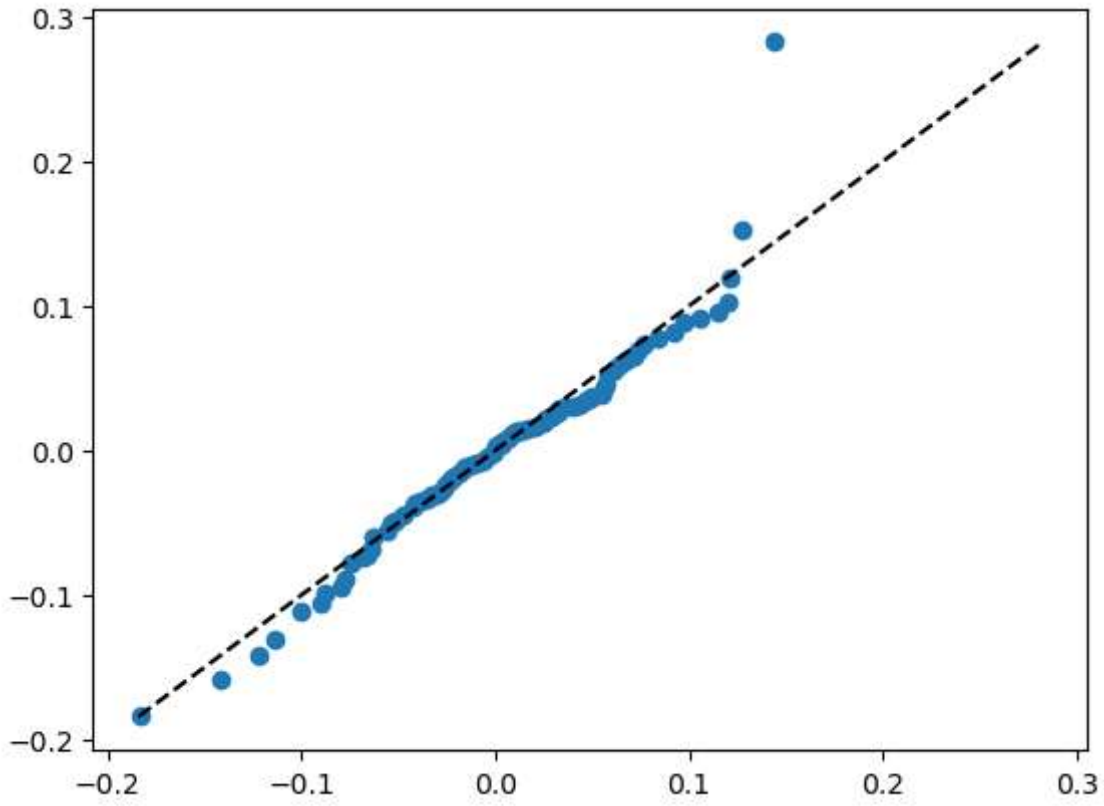
In [152…
```
#(k)
def compute_wd(residuals):
    return np.sum((residuals[1:] - residuals[:-1])**2)/np.sum(residuals[1:]**2)

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: 2.314061317821813
Breusch Godfrey: 6.555284060427799 with p value 0.010457314373484072
```

The p value of Breusch Godfrey says there is some auto correlation in the errors

In [153…
```
#(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 [154…

```
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 29.59168817307244 with p value of 3.751859427580939e-07
```

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

In [156…

```
# (m)
output['condition_number']
```

Out[156]:

```
22.689591316808578
```

Condition number is low, showing no multicollinearity

In [183…

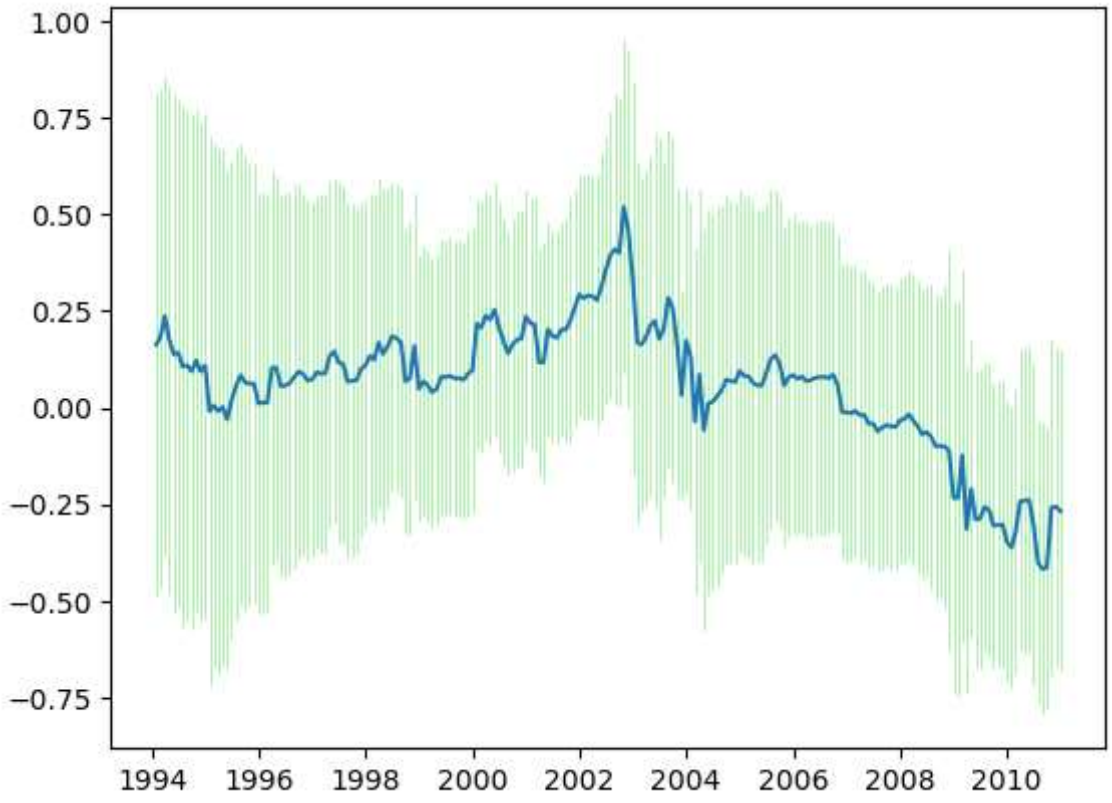
```
# (n)
A = X[None, :].T
B = np.append(A, np.power(A, 2), 1)
n_out = ols_estimator(Y, B)
critical_value = scipy.stats.t.ppf(0.975, df=b_out['N'])
sq_beta, sq_stderr = n_out['betas'][2], n_out['white_stderr'][2]
```

```
print(f"t value of the coefficient of square term is {abs(sq_beta)/sq_stderr}")
print(f"This is less than 2, which means it is not statictically significant")
```

```
t value of the coefficient of square term is 1.6922059657570958
This is less than 2, which means it is not statictically significant
```

In [169…

```
# (o)
x_arr = []
beta_arr = []
error_array = []
for i in range(X.shape[0] - 60):
    x = data['Date'][i]
    b_out = ols_estimator(Y[i:60+i],  X[None, :].T[i:60+i, :])
    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

Question 2

In [186…

```
X1 = data['MKT'] - data['RF']
X2 = data['HH'] - data['RF']
```

```
Y = data['UN'] - data['RF']
XX = np.append(X1[None, :].T, X2[None, :].T, 1)
```

In [188…
```
print("\n**Q2. Running OLS on MKT, HH and UN**\n")
output = ols_estimator(Y,  XX)
```

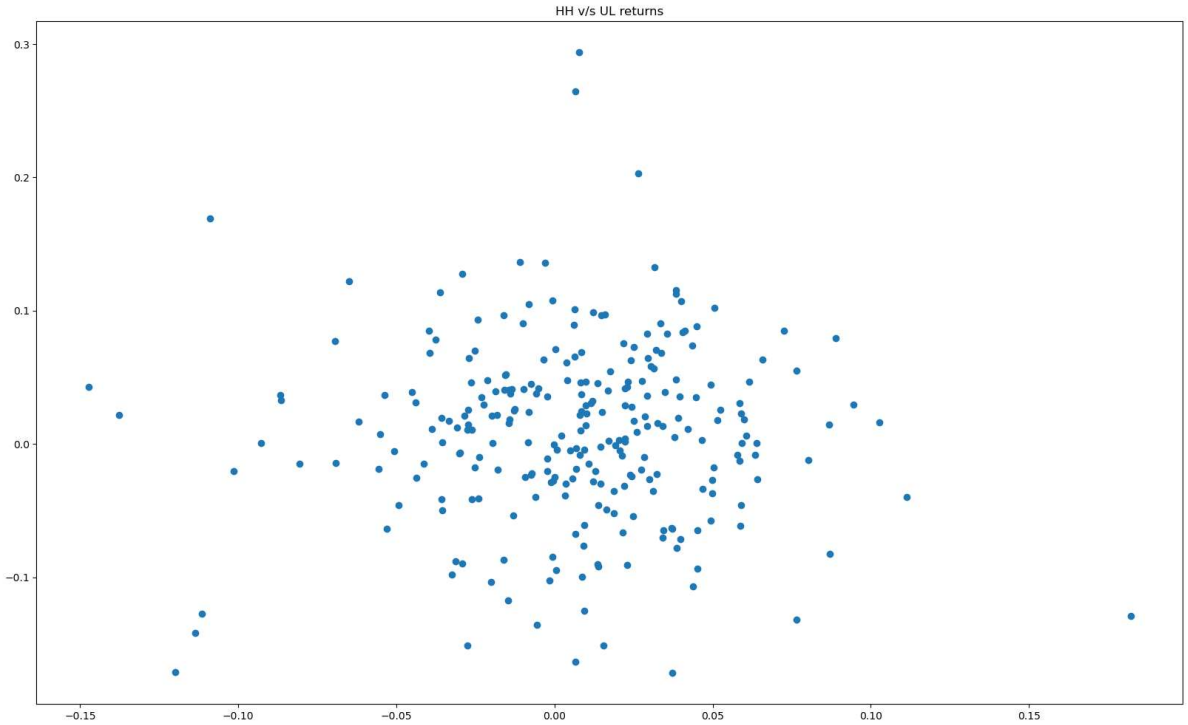
Q2. Running OLS on MKT, HH and UN

In [190…
```
print(f"(a) Alpha: {output['betas'][0]}")
print(f"(a) Beta1: {output['betas'][1]}")
print(f"(a) Beta2: {output['betas'][2]}")
print(f"(a) R Square: {output['R_square']}")
print(f"(a) Adjusted R Square {output['R_square_adj']}")
```

```
(a) Alpha: 0.008243558628830663
(a) Beta1: 0.09675667700539844
(a) Beta2: -0.0748651379251548
(a) R Square: 0.002516664659637069
(a) Adjusted R Square -0.0012905236431888234
```

In [193…
```
#(b)
fig, ax = plt.subplots(1, figsize=(20,12))
ax.scatter(X2, Y)
ax.set_title(f"HH v/s UL returns")
fig.show()
```

```
/tmp/ipykernel_6598/911908669.py:5: UserWarning: Matplotlib is currently using mod
ule://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot show
the figure.
  fig.show()
```



In [194…
```
#(c)
print("**Variance-covariance matrix under homoskedasticity**\n")
print(output['ols_variance_matrix'])
```

```
**Variance-covariance matrix under homoskedasticity**

[[ 1.75731473e-05 -2.86771271e-05 -4.22629430e-05]
 [-2.86771271e-05  1.43636066e-02 -9.36299963e-03]
 [-4.22629430e-05 -9.36299963e-03  1.58133725e-02]]
```

In [195…

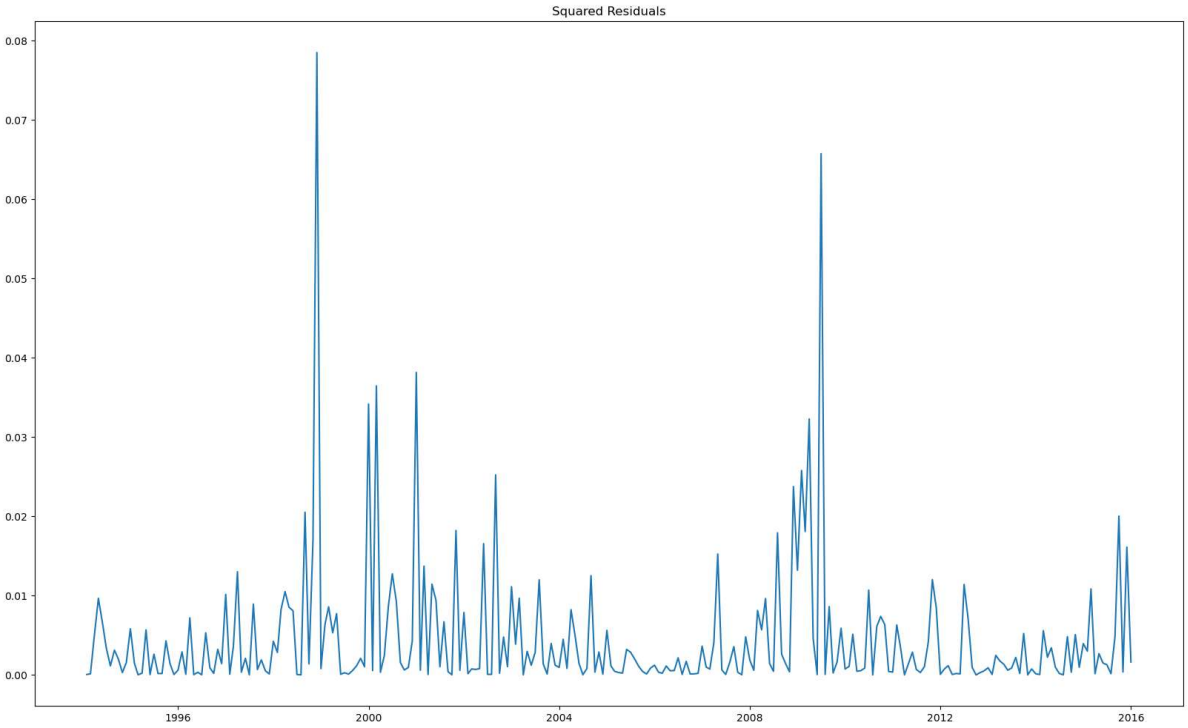
```
#(d)
TEST_ALPHAS = [0.01, 0.05, 0.1]

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['ols_stderr'], output['N']), 0)
```

```
**Running T-test**

t values: [1.96648191 0.80732647 0.59534317]
Test beta=0 rejected at alpha= 1%: [False False False]
Test beta=0 rejected at alpha= 5%: [ True False False]
Test beta=0 rejected at alpha=10%: [ True False False]
```

In [196…

```
#(e)
fig, ax = plt.subplots(1, figsize=(20,12))
ax.plot(data['Date'], output['residuals']**2)
ax.set_title(f"Squared Residuals")
fig.show()
```

```
/tmp/ipykernel_6598/1734843182.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 [198…
```
#(f)
def confidence_intervals2(b_cap, stderr, N):
    for alpha in TEST_ALPHAS:
        critical_value = scipy.stats.t.ppf(1 - alpha/2, df=N)
        high = critical_value*stderr + b_cap
        low = b_cap - critical_value*stderr
        print(f"{(1 - alpha)*100:.0f}% confidence interval alpha is [{low[0]:.4f},
        print(f"{(1 - alpha)*100:.0f}% confidence interval beta1 is [{low[1]:.4f},
        print(f"{(1 - alpha)*100:.0f}% confidence interval beta2 is [{low[2]:.4f},
print(f"Standard errors are {output['ols_stderr']} under homoskedasticity")
confidence_intervals2(output['betas'], output['ols_stderr'], output['N'])
```

```
Standard errors are [0.00419203 0.11984827 0.12575123] under homoskedasticity
99% confidence interval alpha is [-0.0026, 0.0191]
99% confidence interval beta1 is [-0.2142, 0.4077]
99% confidence interval beta2 is [-0.4011, 0.2514]
95% confidence interval alpha is [-0.0000, 0.0165]
95% confidence interval beta1 is [-0.1392, 0.3327]
95% confidence interval beta2 is [-0.3225, 0.1727]
90% confidence interval alpha is [0.0013, 0.0152]
90% confidence interval beta1 is [-0.1011, 0.2946]
90% confidence interval beta2 is [-0.2824, 0.1327]
```

In [199…
```
#(g)
print("**Variance-covariance matrix under hetroskedasticity**\n")
print(output['white_variance_matrix'])
```

```
**Variance-covariance matrix under hetroskedasticity**

[[ 1.79191665e-05  3.63092127e-05 -1.45685170e-04]
 [ 3.63092127e-05  2.29124546e-02 -1.61751193e-02]
 [-1.45685170e-04 -1.61751193e-02  2.50740590e-02]]
```

In [200…
```
#(h)
print(f"Standard errors are {output['white_stderr']} using white")
confidence_intervals2(output['betas'], output['white_stderr'], output['N'])
```

```
Standard errors are [0.0042331 0.1513686 0.1583479] using white
99% confidence interval alpha is [-0.0027, 0.0192]
99% confidence interval beta1 is [-0.2960, 0.4895]
99% confidence interval beta2 is [-0.4857, 0.3360]
95% confidence interval alpha is [-0.0001, 0.0166]
95% confidence interval beta1 is [-0.2013, 0.3948]
95% confidence interval beta2 is [-0.3867, 0.2369]
90% confidence interval alpha is [0.0013, 0.0152]
90% confidence interval beta1 is [-0.1531, 0.3466]
90% confidence interval beta2 is [-0.3362, 0.1865]
```

In [201…

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

```
t values: [1.94740297 0.63921232 0.47278894]
Test beta=0 rejected at alpha= 1%: [False False False]
Test beta=0 rejected at alpha= 5%: [ True False False]
Test beta=0 rejected at alpha=10%: [ True False False]
```

In [202…

```
# (j)
print("**AIC, BIC, HQ**\n")
k = 3
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: -670.0154265792513
BIC: -659.2875792698123
HQ: -670.8600389425312
```

In [203…

```
#(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)}")
```

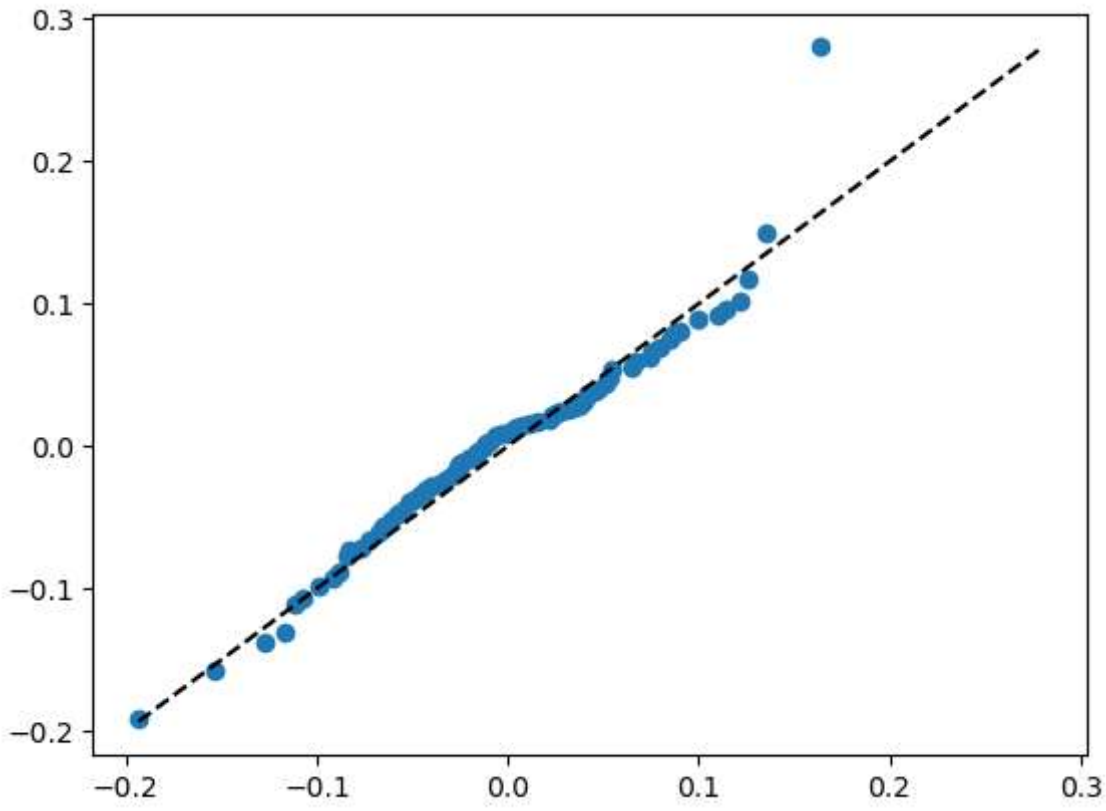
```
DW: 2.281704514565686
Breusch Godfrey: 5.2735323079167795 with p value 0.021652034081948712
```

The p value of Breusch Godfrey says there is some auto correlation in the errors

In [205…

```
#(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 [206…
```
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 29.267689975531315 with p value of 4.411658356406889e-07
```

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

In [207…
```
# (m)
output['condition_number']
```

Out[207]:
```
37.80057546523
```

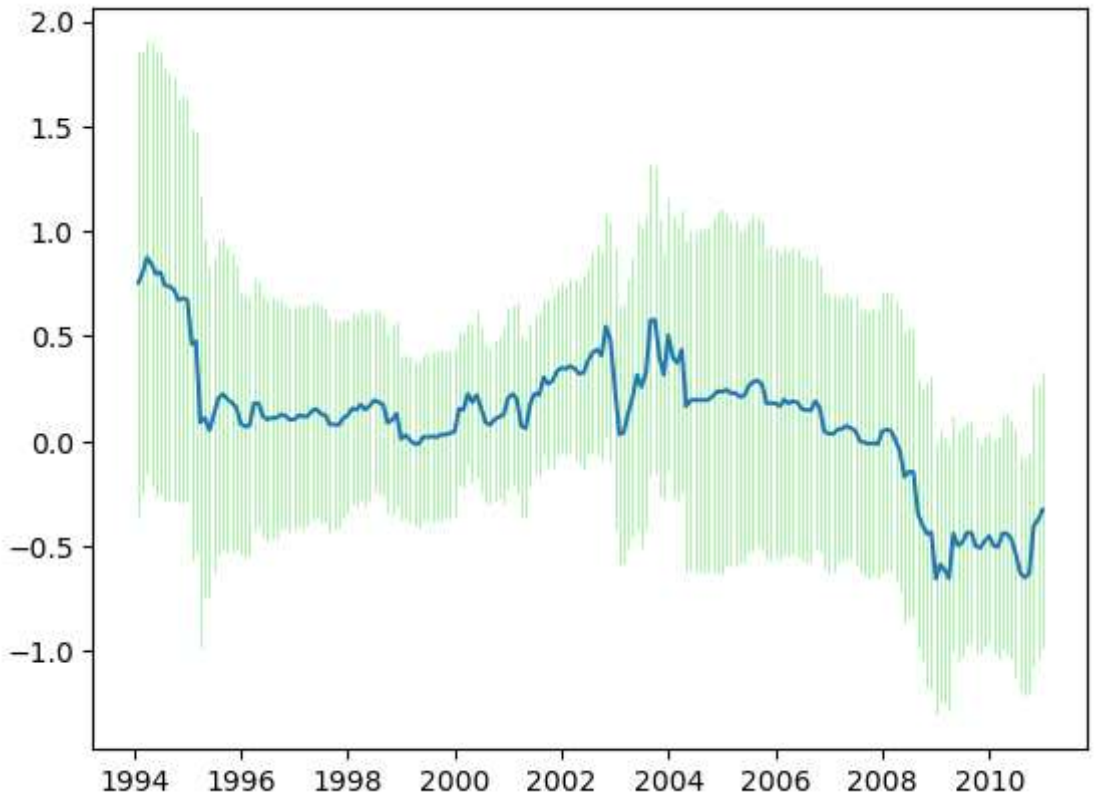
Condition number is low, showing no multicollinearity

In [211…
```
# (n)
A = XX
B = np.append(A, np.power(A, 2), 1)
n_out = ols_estimator(Y, B)
critical_value = scipy.stats.t.ppf(0.975, df=b_out['N'])
sq_beta, sq_stderr = n_out['betas'][3], n_out['white_stderr'][3]
print(f"t value of the coefficient of first square term is {abs(sq_beta)/sq_stderr}
print(f"This is less than 2, which means it is not statictically significant")

sq_beta, sq_stderr = n_out['betas'][4], n_out['white_stderr'][4]
print(f"t value of the coefficient of second square term is {abs(sq_beta)/sq_stderr
print(f"This is less than 2, which means it is not statictically significant")
```

```
t value of the coefficient of first square term is 0.9921225003608571
This is less than 2, which means it is not statictically significant
t value of the coefficient of second square term is 1.045487683206051
This is less than 2, which means it is not statictically significant
```

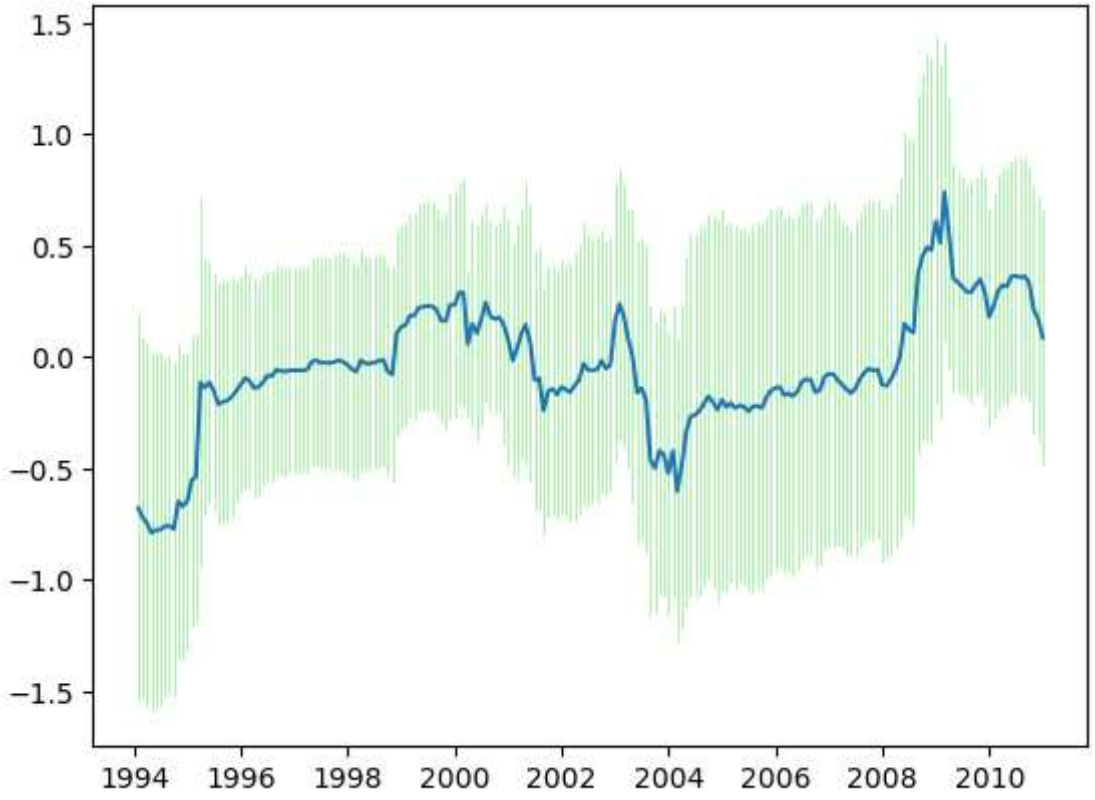
In [213…

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

beta2_arr = []
error2_array = []

for i in range(XX.shape[0] - 60):
    x = data['Date'][i]
    b_out = ols_estimator(Y[i:60+i],  XX[i:60+i, :])
    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 [214…

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



Both betas have seasonal patterns and changes significantly over time

Question 3

To me, both models seem to be close and quite comparable. Adjusted R quare is greater in the first model, but the BIC and AIC are lower for second model, though these are close. The beta standard errors of the second model are larger, indicating instability in the model. This can also be seen in the seasonal trend of both the betas. Model 1 is a lot more stable and it seems to be the better model overall.

Question 4

PG-MKT beta is 0.3282 with a 95% confidence interval of [0.176, 0.481]

UL-MKT beta is 0.0524 with a 95% confidence interval of [-0.1326, 0.2375]