Problem Set 8

Problem 1

Environment Setup

In [ ]:
```
# library imports
import numpy as np
import pandas as pd
from IPython.display import display
import math
from scipy import stats
import matplotlib.pyplot as plt
import statsmodels.api as sm
```

In [ ]:
```
# notebook settings
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

%matplotlib inline
```

In [ ]:
```
# constants
PHI_CASES = [0, 0.5, 0.95, 0.99, 1]
T_CASES = [50, 100, 1000, 5000]
RUNS = 1000
```

Part (a)

Going per the explanation in the lecture, we can estimate the AR coefficients through OLS as follows:

If $z_t = \vec{\phi}^T \vec{z} + \epsilon_t$, where $\vec{z}$ represents the lagged values of the autoregressed variable, then $\hat{\phi} = (X^T X)^{-1} X^T y$, where X and y represent the time series variable used as the independent and dependent variable in OLS.

Under the assumptions of classic OLS, we expect the estimates $\hat{\phi}$ of the AR coefficients to have the following properties:

- $E[\hat{\phi}] = \phi$, that is, $\hat{\phi}$ is unbiased,
- if the regressors are normally distributed, then so will be $\hat{\phi}$, and
- $\hat{\phi}$ is consistent.

Parts (b) and (c)

In [ ]:

```
# simulating the AR(1) process with normal white noise
# with time series length T and coeff phi
def get_ar_time_series(T, phi, init_value = 0):
  e = stats.norm.rvs(size = T) # get the epsilons
  z, curr_z = [], init_value
  for i in range(e.shape[0]):
    curr_z = curr_z * phi + e[i]
    z.append(curr_z)
  return np.array(z)

# utility for fitting linear regression
def fit_ols(X, y):
  inv_XTX = np.linalg.inv(np.matmul(
      np.transpose(X), X
  ))
  XTy = np.matmul(np.transpose(X), y)
  beta = np.matmul(inv_XTX, XTy)
  return beta

# getting phi as estimated through OLS
def estimate_phi(z):
  x, y = z[: -1], z[1:]
  X = np.vstack((
      np.ones(x.shape), x
  )).transpose()
  beta = fit_ols(X, y)
  return beta[1]

# simulating case of specific phi and T
def simulate_case(phi, T, runs = RUNS):
  results = dict()
  results["estimates"] = []
  for run in range(runs):
    z = get_ar_time_series(T, phi)
    results["estimates"].append(estimate_phi(z))
  results["estimates"] = np.array(results["estimates"])
  results["mean"] = np.mean(results["estimates"])
  results["std"] = np.std(results["estimates"])
  return results
```

In [ ]:

```
# utility for displaying the table
# table_type = "mean" or "std"
def display_table(results_dict, table_type):
  cols_T = ["T={}".format(T) for T in T_CASES]
  columns = ["phi"] + cols_T
  df = pd.DataFrame({col: [] for col in columns})
  for phi in PHI_CASES:
    curr_row = {col: results_dict[phi][col][table_type] for col in cols_T}
    curr_row["phi"] = phi
    df = df.append(curr_row, ignore_index = True)
  return df

# plotting the distribution of phi estimates for each phi, T
def plot_distributions(results_dict):
  fig, axes = plt.subplots(
      nrows = len(PHI_CASES),
      ncols = len(T_CASES),
      figsize = (15, 20)
  )
  fig.suptitle(
      "Distribution of Phi Estimates for Different True Phi and T",
      fontsize = 20
  )
  for i, phi in enumerate(PHI_CASES):
    for j, T in enumerate(T_CASES):
      axes[i, j].set_title(
          "True Phi = {}, T = {}".format(phi, T),
          fontsize = 12
      )
      axes[i, j].hist(
          results_dict[phi]["T={}".format(T)]["estimates"],
          bins = T // 10 if T <= 100 else T // 20
      )
  plt.show()
```

In [ ]:

```
# running the experiments
results = dict()
for phi in PHI_CASES:
  results[phi] = dict()
  for T in T_CASES:
    print("Case: phi = {}, T = {}".format(phi, T))
    results[phi]["T={}".format(T)] = simulate_case(phi, T)
```

```
Case: phi = 0, T = 50
Case: phi = 0, T = 100
Case: phi = 0, T = 1000
Case: phi = 0, T = 5000
Case: phi = 0.5, T = 50
Case: phi = 0.5, T = 100
Case: phi = 0.5, T = 1000
Case: phi = 0.5, T = 5000
Case: phi = 0.95, T = 50
Case: phi = 0.95, T = 100
Case: phi = 0.95, T = 1000
Case: phi = 0.95, T = 5000
Case: phi = 0.99, T = 50
Case: phi = 0.99, T = 100
Case: phi = 0.99, T = 1000
Case: phi = 0.99, T = 5000
Case: phi = 1, T = 50
Case: phi = 1, T = 100
Case: phi = 1, T = 1000
Case: phi = 1, T = 5000
```

In [ ]:

```
print("----Table of Means----\n")
display_table(results, "mean")
```

```
----Table of Means----


<ipython-input-47-090f391b7f82>:10: FutureWarning: The frame.append method is depre
cated and will be removed from pandas in a future version. Use pandas.concat instea
d.
  df = df.append(curr_row, ignore_index = True)
<ipython-input-47-090f391b7f82>:10: FutureWarning: The frame.append method is depre
cated and will be removed from pandas in a future version. Use pandas.concat instea
d.
  df = df.append(curr_row, ignore_index = True)
<ipython-input-47-090f391b7f82>:10: FutureWarning: The frame.append method is depre
cated and will be removed from pandas in a future version. Use pandas.concat instea
d.
  df = df.append(curr_row, ignore_index = True)
<ipython-input-47-090f391b7f82>:10: FutureWarning: The frame.append method is depre
cated and will be removed from pandas in a future version. Use pandas.concat instea
d.
  df = df.append(curr_row, ignore_index = True)
<ipython-input-47-090f391b7f82>:10: FutureWarning: The frame.append method is depre
cated and will be removed from pandas in a future version. Use pandas.concat instea
d.
  df = df.append(curr_row, ignore_index = True)
```

Out[ ]:

	phi	T=50	T=100	T=1000	T=5000
0	0.00	-0.015837	-0.013122	-0.000893	-0.000186
1	0.50	0.450861	0.475710	0.496654	0.499795
2	0.95	0.849945	0.902261	0.945207	0.949290
3	0.99	0.881865	0.936045	0.985461	0.989167
4	1.00	0.898955	0.945610	0.994661	0.998892

In [ ]:

```
print("----Table of Standard Deviations----\n")
display_table(results, "std")
```

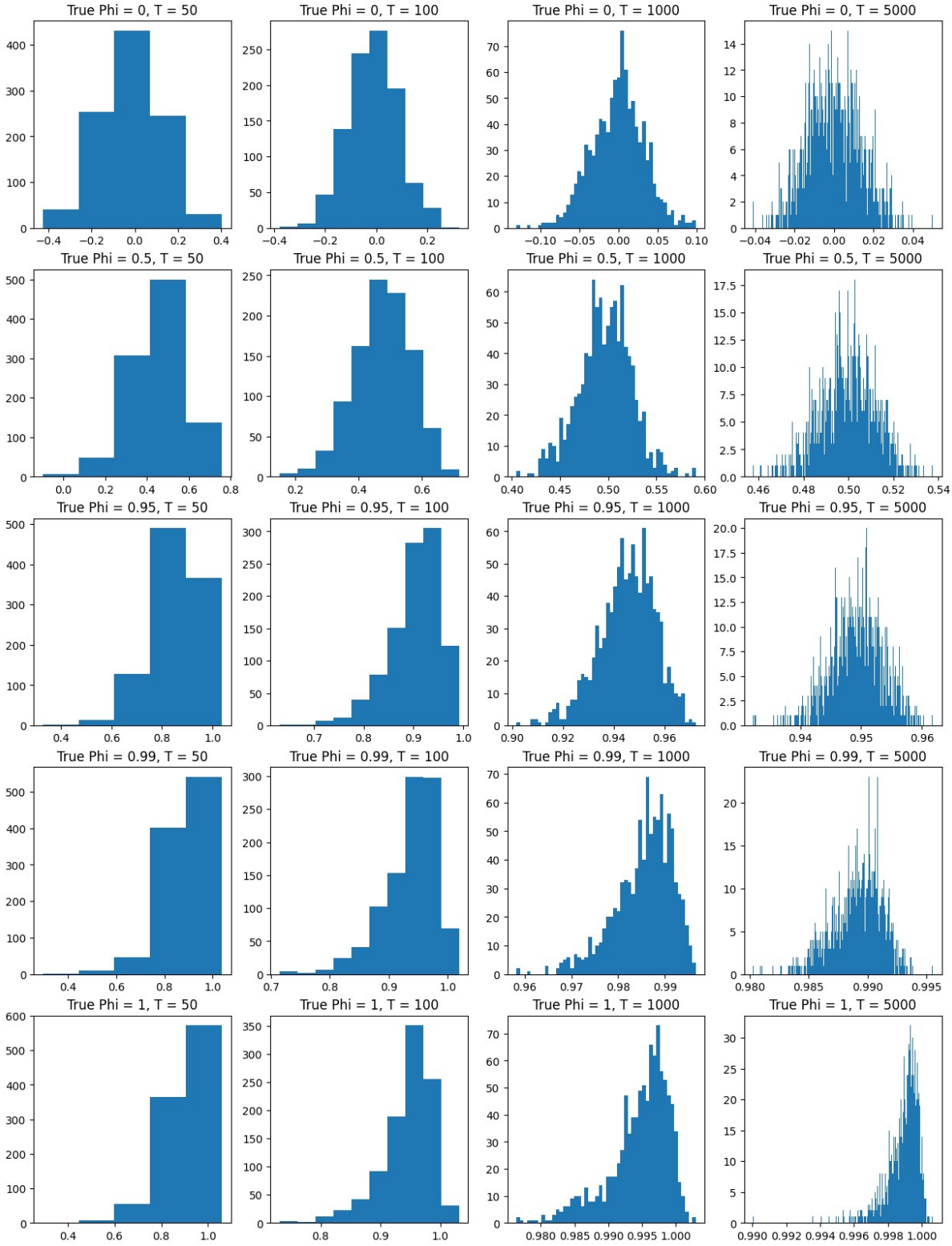
```
----Table of Standard Deviations----


<ipython-input-47-090f391b7f82>:10: FutureWarning: The frame.append method is depre
cated and will be removed from pandas in a future version. Use pandas.concat instea
d.
  df = df.append(curr_row, ignore_index = True)
<ipython-input-47-090f391b7f82>:10: FutureWarning: The frame.append method is depre
cated and will be removed from pandas in a future version. Use pandas.concat instea
d.
  df = df.append(curr_row, ignore_index = True)
<ipython-input-47-090f391b7f82>:10: FutureWarning: The frame.append method is depre
cated and will be removed from pandas in a future version. Use pandas.concat instea
d.
  df = df.append(curr_row, ignore_index = True)
<ipython-input-47-090f391b7f82>:10: FutureWarning: The frame.append method is depre
cated and will be removed from pandas in a future version. Use pandas.concat instea
d.
  df = df.append(curr_row, ignore_index = True)
<ipython-input-47-090f391b7f82>:10: FutureWarning: The frame.append method is depre
cated and will be removed from pandas in a future version. Use pandas.concat instea
d.
  df = df.append(curr_row, ignore_index = True)
```

Out[ ]:

	phi	T=50	T=100	T=1000	T=5000
0	0.00	0.143638	0.097229	0.033725	0.013838
1	0.50	0.128468	0.089366	0.027628	0.012589
2	0.95	0.091106	0.051068	0.011109	0.004381
3	0.99	0.086564	0.045554	0.006179	0.002128
4	1.00	0.084677	0.042617	0.004465	0.000961

In [ ]:

```
plot_distributions(results)
```


Distribution of Phi Estimates for Different True Phi and T
True Phi = 0, T = 50
True Phi = 0, T = 100
True Phi = 0, T = 1000
True Phi = 0, T = 5000
True Phi = 0.5, T = 50
True Phi = 0.5, T = 100
True Phi = 0.5, T = 1000
True Phi = 0.5, T = 5000
True Phi = 0.95, T = 50
True Phi = 0.95, T = 100
True Phi = 0.95, T = 1000
True Phi = 0.95, T = 5000
True Phi = 0.99, T = 50
True Phi = 0.99, T = 100
True Phi = 0.99, T = 1000
True Phi = 0.99, T = 5000
True Phi = 1, T = 50
True Phi = 1, T = 100
True Phi = 1, T = 1000
True Phi = 1, T = 5000

Part (d)

From the table displaying means of the estimators $\hat{\phi}$, it may be observed that while the estimates are close to the true value ϕ, they are always a bit smaller than the true ϕ. For instance, $\hat{\phi} = 0.89, 0.95, 0.99, 0.999$ when $\phi = 1.0$, thus always being smaller than the true value. Hence, $\hat{\phi}$ is biased (as opposed to our theoretical claim that it should be unbiased).

From the table of standard deviations, or standard errors, of $\hat{\phi}$, it is easy to note that the standard deviations decline in value as T increases, lending support to the theoretical claim on consistency.

Finally, the plots of distributions of the estimator $\hat{\phi}$s indicate that as the true ϕ increases, the distributions becomes more and more assymetric (skewed) and hence not normal. As such the theoretical claim on normality is not followed.

Problem Set 8

Problem 2

Environment Setup

In [1]:
```
# library imports
import numpy as np
import pandas as pd
from IPython.display import display
import math
from scipy import stats
import matplotlib.pyplot as plt
import statsmodels.api as sm
import yfinance
import datetime
```

```
/usr/local/lib/python3.10/dist-packages/yfinance/base.py:48: FutureWarning: The def
ault dtype for empty Series will be 'object' instead of 'float64' in a future versi
on. Specify a dtype explicitly to silence this warning.
  _empty_series = pd.Series()
```

In [2]:
```
# notebook settings
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

%matplotlib inline
```

Getting the Data

In [3]:
```
df = yfinance.download(
    ["^GSPC"],
    start = datetime.date(2000, 1, 1),
    end = datetime.date(2023, 12, 31),
    interval = "1mo"
)
```

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

In [4]:
```
df.shape
df.columns
df.head()
```

Out[4]:
```
(288, 6)
```

Out[4]:
```
Index(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
```

Out[4]:

Date	Open	High	Low	Close	Adj Close	Volume
2000-01-01	1469.250000	1478.000000	1350.140015	1394.459961	1394.459961	21494400000
2000-02-01	1394.459961	1444.550049	1325.069946	1366.420044	1366.420044	20912000000
2000-03-01	1366.420044	1552.869995	1346.619995	1498.579956	1498.579956	26156200000
2000-04-01	1498.579956	1527.189941	1339.400024	1452.430054	1452.430054	20106460000
2000-05-01	1452.430054	1481.510010	1361.089966	1420.599976	1420.599976	19898300000

In [5]:
```
df.isna().sum()
```

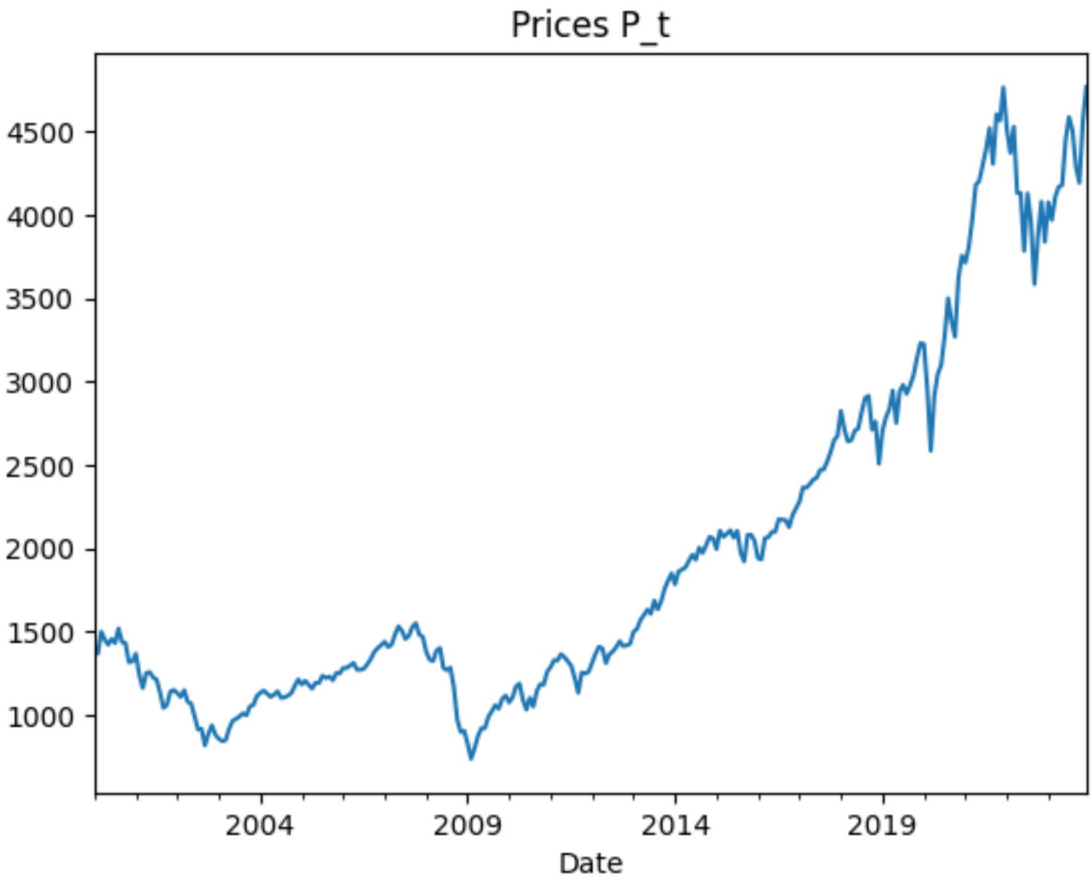
Out[5]:
```
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

In [6]:
```
df["Pt"] = df["Adj Close"]
```

Part (a)

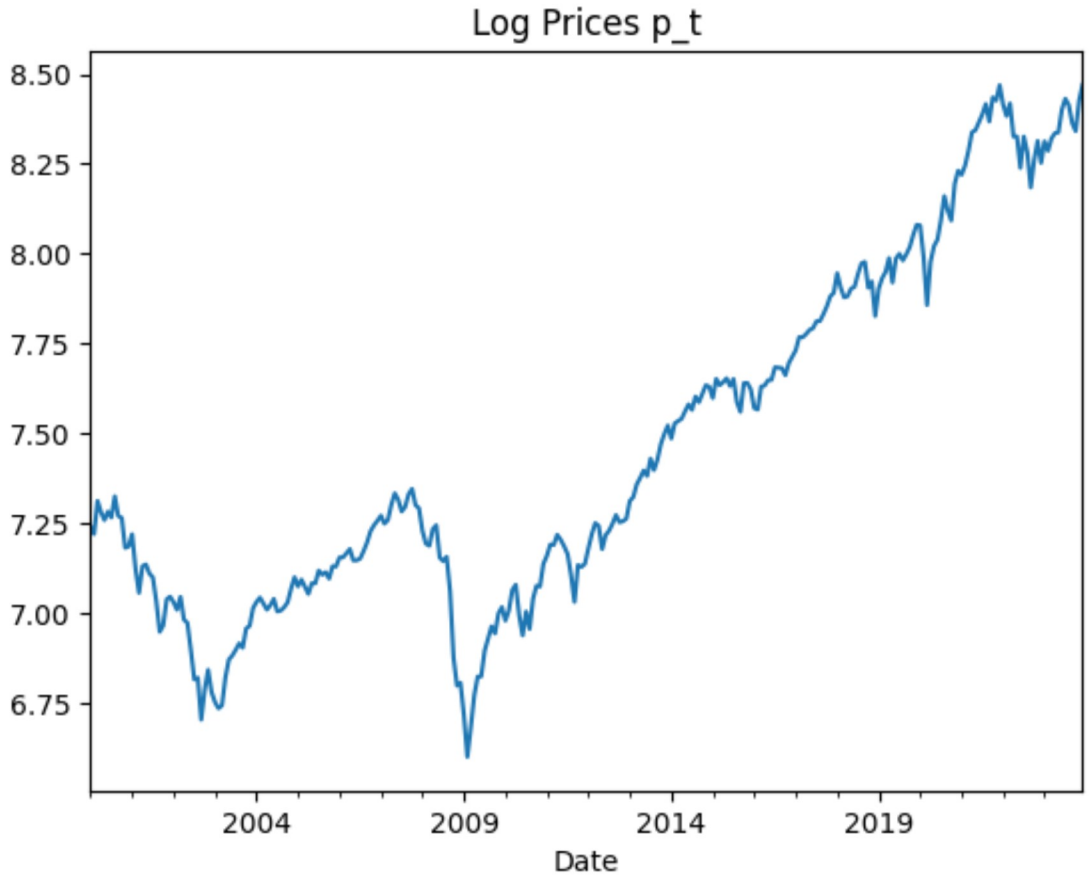
In [7]:
```
df["Pt"].plot(title = "Prices P_t")
```

Out[7]:
```
<Axes: title={'center': 'Prices P_t'}, xlabel='Date'>
```



In [8]:
```
df["pt"] = np.log(df["Pt"])
df["pt"].plot(title = "Log Prices p_t")
```

Out[8]:
```
<Axes: title={'center': 'Log Prices p_t'}, xlabel='Date'>
```



From the above plots, it is easy to observe that there is a general rise in the monthly prices over the studied time horizon (2000 - 2023). Therefore, the series is not stationary.

The prices increase approximately exponentially in the period 2010 to 2020, as indicated by the plot of the raw prices P_t. This can be converted to a linear trend by taking the log of the prices as done for $p_t = log(P_t)$.

Part (b)

In [9]:
```
Pt, pt = df["Pt"].to_numpy(), df["pt"].to_numpy()
Rt = Pt[1:] / Pt[:-1] # returns
rt = pt[1:] - pt[:-1] # log returns
```

In [10]:
```
Rt.shape
rt.shape
Rt[:10]
rt[:10]
```

Out[10]:
```
(287,)
```

Out[10]:
```
(287,)
```

Out[10]:
```
array([0.97989192, 1.09671983, 0.96920424, 0.97808495, 1.02393355,
       0.98365872, 1.0606991 , 0.94651703, 0.99505051, 0.91993139])
```

Out[10]:
```
array([-0.020313  ,  0.09232375, -0.03127991, -0.02215875,  0.02365163,
       -0.01647627,  0.05892822, -0.05496632, -0.00496177, -0.08345619])
```

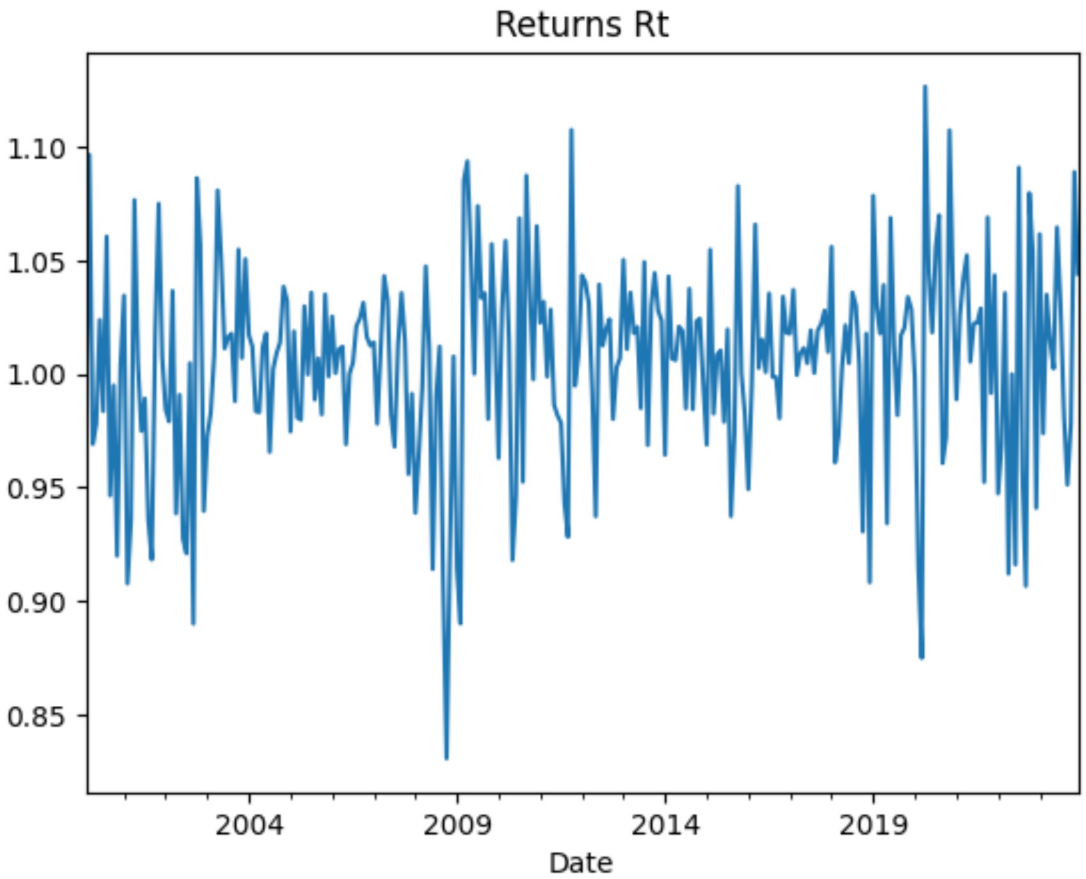
In [11]:
```
df_returns = pd.DataFrame(index = pd.Index(df.index.to_series()[1:]))
df_returns["Rt"], df_returns["rt"] = Rt, rt
df_returns.head()
```

Out[11]:

Date	Rt	rt
2000-02-01	0.979892	-0.020313
2000-03-01	1.096720	0.092324
2000-04-01	0.969204	-0.031280
2000-05-01	0.978085	-0.022159
2000-06-01	1.023934	0.023652

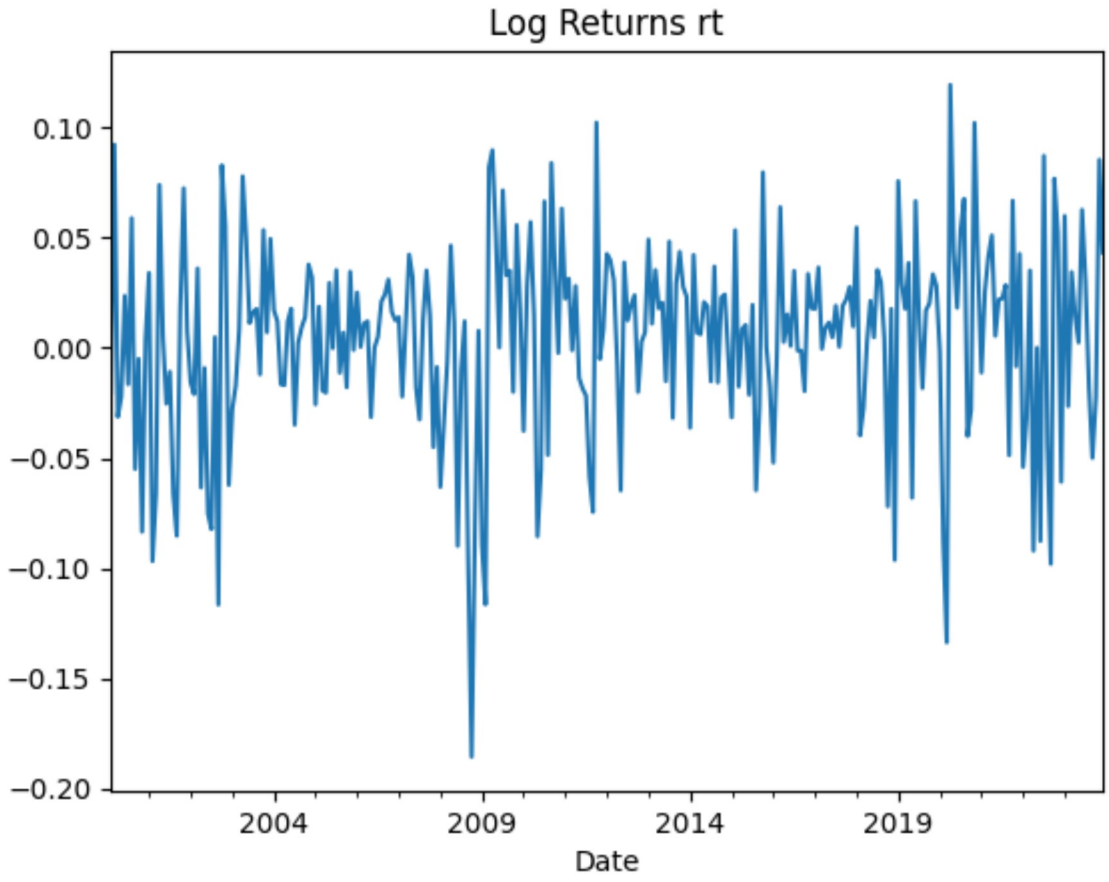
In [12]:
```
df_returns["Rt"].plot(title = "Returns Rt")
```

Out[12]:
```
<Axes: title={'center': 'Returns Rt'}, xlabel='Date'>
```



In [13]:
```
df_returns["rt"].plot(title = "Log Returns rt")
```

Out[13]:
```
<Axes: title={'center': 'Log Returns rt'}, xlabel='Date'>
```

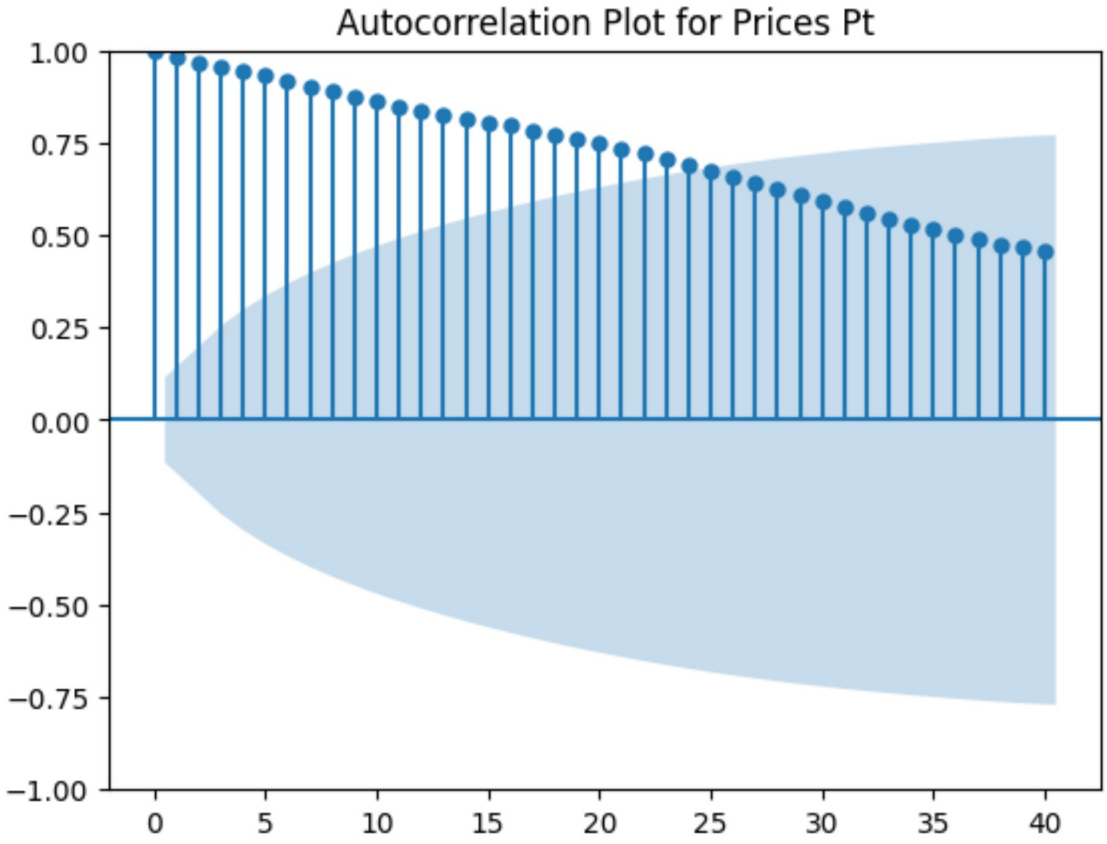


By taking the ratio of successive terms $R_t = \frac{P_{t+1}}{P_t}$, we eliminate the increasing trend found in the plots of P_t and p_t. Both R_t and $r_t = log(R_t)$ show no such trend.

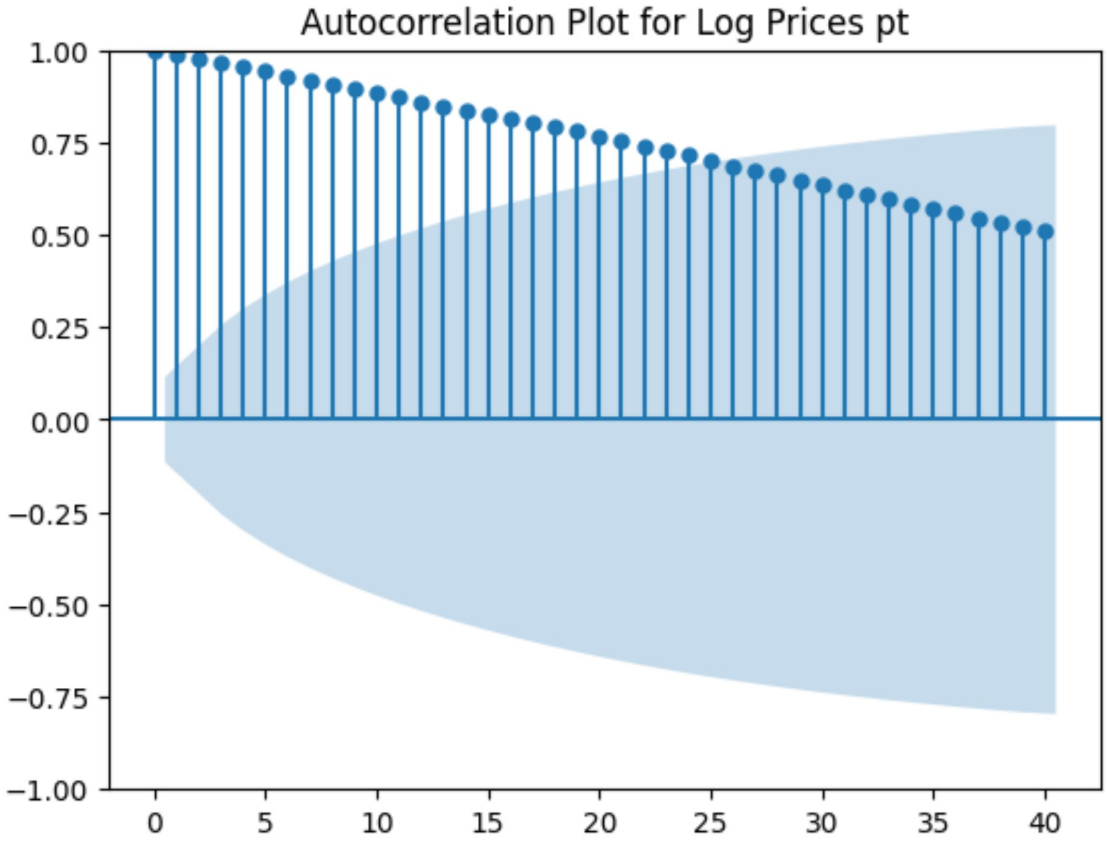
This signifies that although the prices have increased by a lot (overall) on from 2000 to 2023, the monthly returns have remained between certain bounds.

Part (c)

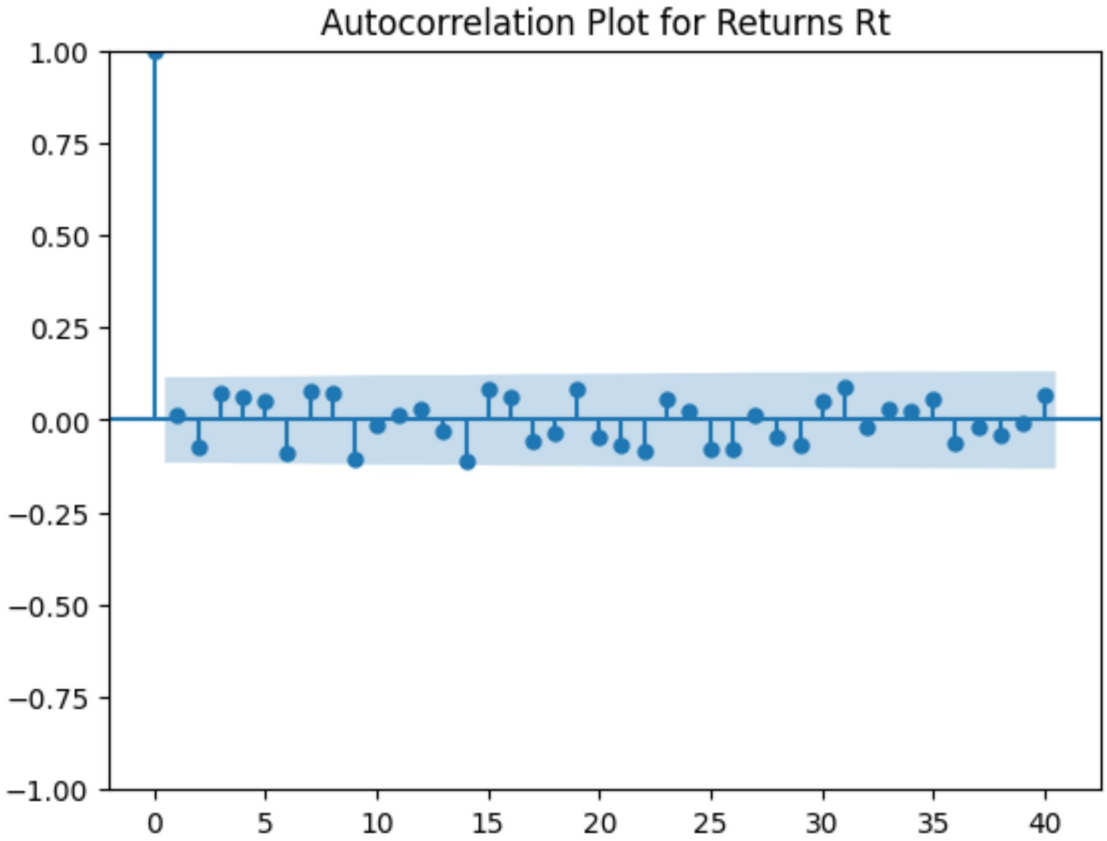
In [14]:
```
_ = sm.graphics.tsa.plot_acf(Pt, lags = 40, title = "Autocorrelation Plot for Price
```



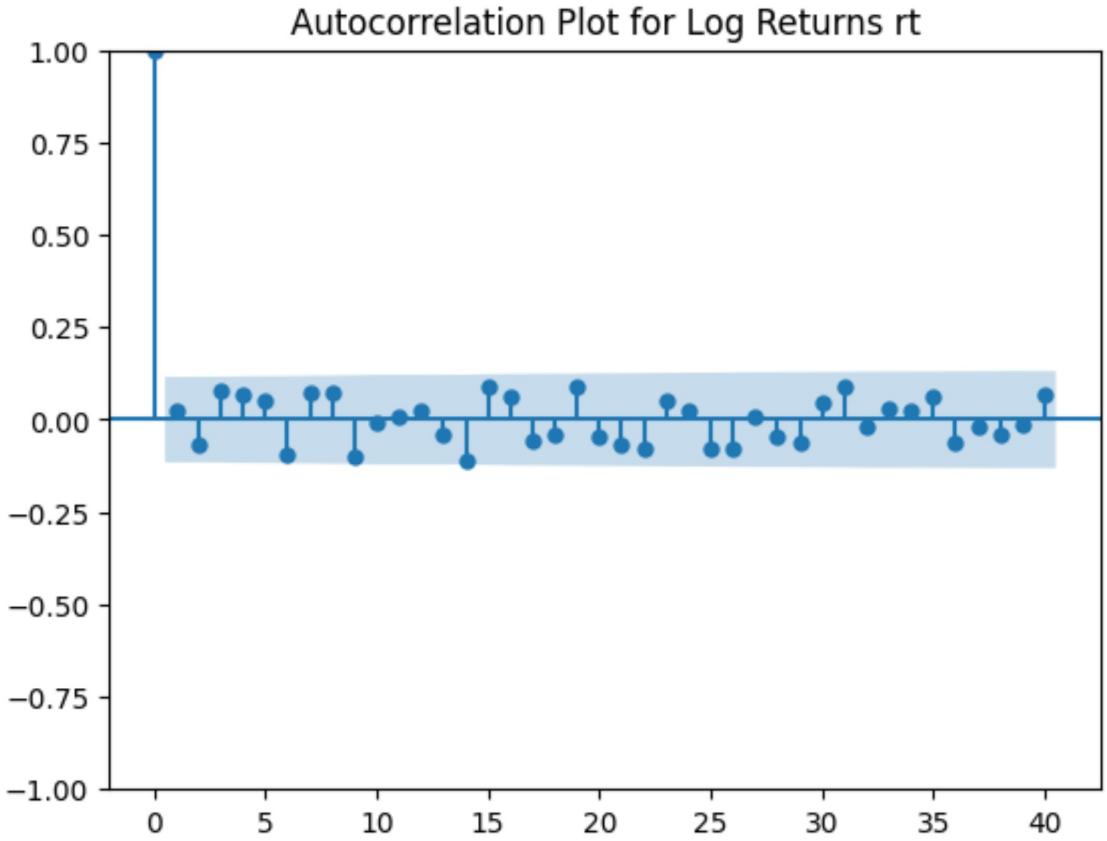
In [15]:
```
_ = sm.graphics.tsa.plot_acf(pt, lags = 40, title = "Autocorrelation Plot for Log P
```



In [16]:
```
_ = sm.graphics.tsa.plot_acf(Rt, lags = 40, title = "Autocorrelation Plot for Retur
```



In [17]:
```
_ = sm.graphics.tsa.plot_acf(rt, lags = 40, title = "Autocorrelation Plot for Log R
```



The autocorrelation function (ACF) plot for prices P_t show a gradually declining pattern with increasing lags. This pattern indicates the possibility of an underlying trend in the time series (which can be identified as a general growth from the plot of P_t in part (a)). Not surprisingly, the plot for log prices p_t shows similar gradual decay (albeit more linear).

The ACF for returns R_t and log returns r_t, in contrast, show a sharp drop for lags > 1. This indicates that an $AR(1)$ process might help us model the time series since the values R_t and r_t can be explained to a fair degree using just their immediate predecessors, R_{t-1} and r_{t-1}.

Parts (d) and (e)

In [24]:
```
# variables for AR(0)
ypt0, yrt0 = pt, rt
Xpt0, Xrt0 = np.ones((ypt0.shape[0], 1)), np.ones((yrt0.shape[0], 1))
ypt0.shape, yrt0.shape, Xpt0.shape, Xrt0.shape

# variables for AR(1)
ypt1, yrt1 = pt[1:], rt[1:]
Xpt1, Xrt1 = np.concatenate(
    (Xpt0[1:, :], pt[:-1].reshape((pt[:-1].shape[0], 1))),
    axis = 1
), np.concatenate(
    (Xrt0[1:, :], rt[:-1].reshape((rt[:-1].shape[0], 1))),
    axis = 1
)
ypt1.shape, yrt1.shape, Xpt1.shape, Xrt1.shape

# variables for AR(2)
ypt2, yrt2 = pt[2:], rt[2:]
Xpt2, Xrt2 = np.concatenate(
    (Xpt1[1:, :], pt[:-2].reshape((pt[:-2].shape[0], 1))),
    axis = 1
), np.concatenate(
    (Xrt1[1:, :], rt[:-2].reshape((rt[:-2].shape[0], 1))),
    axis = 1
)
ypt2.shape, yrt2.shape, Xpt2.shape, Xrt2.shape
```

Out[24]:
```
((288,), (287,), (288, 1), (287, 1))
```

Out[24]:
```
((287,), (286,), (287, 2), (286, 2))
```

Out[24]:
```
((286,), (285,), (286, 3), (285, 3))
```

In [33]:

```
# part a
# fitting the regression to calculate beta
def fit_ols(X, y):
  inv_XTX = np.linalg.inv(np.matmul(
      np.transpose(X), X
  ))
  XTy = np.matmul(np.transpose(X), y)
  beta = np.matmul(inv_XTX, XTy)
  return beta

# calculate R2 and adjusted R2
def calculate_r2(X, y, beta):
  sst = np.sum((y - np.mean(y)) ** 2)
  y_hat = np.matmul(X, beta)
  ssr = np.sum((y_hat - np.mean(y)) ** 2)
  r2 = ssr / sst

  n, k = X.shape[0], X.shape[1]
  adj_r2 = 1 - (1 - r2) * (n - 1) / (n - k - 1)
  return r2, adj_r2

# part c
# calculating the covariance matrix under assumption of homoskedasticity
def homoskedastic_covariance_matrix(X, y, beta):
  y_hat = np.matmul(X, beta)
  sse = np.sum((y - y_hat) ** 2)
  error_variance = sse / X.shape[0]
  inv_XTX = np.linalg.inv(np.matmul(
      np.transpose(X), X
  ))
  cov_matrix = error_variance * inv_XTX
  return cov_matrix

# part d and i
# null hypothesis: beta = b0
def test_regression_coefficients(X, beta, V, b0 = 0):
  sb = V.diagonal() ** 0.5
  t = (beta - b0) / sb
  n, k = X.shape[0], X.shape[1]
  p_value = 2 * (1 - stats.t.cdf(abs(t), n - k))
  return p_value

# part e
# plot squared residuals
def assess_homoskedasticity(X, y, beta):
  y_hat = np.matmul(X, beta)
  e_sq = (y - y_hat) ** 2
  plt.plot(e_sq)
  plt.title("Squared Residuals", fontsize = 20)
  plt.show()

# part f and h
# getting the standard errors and
# 0.9, 0.95, and 0.99 confidence intervals for coefficients
def get_ci_estimates(X, beta, V):
  ci_intervals = np.array([0.9, 0.95, 0.99])
  dof = X.shape[0] - X.shape[1]
  t_values = -stats.t.ppf((1 - ci_intervals) / 2, dof)
  sb = V.diagonal() ** 0.5
```

```
  sb = v.diagonal() ** 0.5
  result = dict()
  result["standard_errors"] = sb
  for i in range(len(ci_intervals)):
    t_value = t_values[i]
    lower_bounds = beta - sb * t_value
    upper_bounds = beta + sb * t_value
    result[ci_intervals[i]] = [(l, u) for l, u in zip(
        lower_bounds, upper_bounds
    )]
  return result

# part g
# calculating covariance matrix under assumption of heteroskedasticity
def heteroskedastic_covariance_matrix(X, y, beta):
  y_hat = np.matmul(X, beta)
  sse = np.sum((y - y_hat) ** 2)
  error_variance = sse / X.shape[0]
  D = error_variance * np.identity(y.shape[0])
  XTDX = np.matmul(np.transpose(X), np.matmul(D, X))
  inv_XTX = np.linalg.inv(np.matmul(
      np.transpose(X), X
  ))
  V = np.matmul(np.matmul(inv_XTX, XTDX), inv_XTX)
  return V

# part j
# calculation of AIC, BIC, and Hannah-Quinn IC
def calculate_information_criteria(X, y, beta):
  y_hat = np.matmul(X, beta)
  sse = np.sum((y - y_hat) ** 2)
  error_variance = sse / X.shape[0]
  logl = np.sum(np.log(
      1 / (2 * math.pi * error_variance) * np.exp(
          -1 / (2 * error_variance) * (y - y_hat) ** 2
      )
  ))
  n, k = X.shape[0], X.shape[1]
  aic = -2 * logl + 2 * k
  bic = -2 * logl + k * np.log(n)
  hqc = -2 * logl + 2 * k * np.log(np.log(n))
  return aic, bic, hqc

# part k
# durbin-watson and breusch-godfrey tests
def residual_autocorrelation_tests(X, y, beta, significance = 0.95):
  y_hat = np.matmul(X, beta)
  e = y - y_hat

  # durbin-watson test
  result = dict()
  result["dw"] = np.sum((e[1:] - e[:-1]) ** 2) / np.sum(e[1:] ** 2)

  # breusch-godfrey test with 1st order autocorrelation
  e_y, e_X = e[1:], e[:-1]
  e_X = np.concatenate([
      np.ones((e_X.shape[0], 1)), np.reshape(e_X, (e_X.shape[0], 1))
  ], axis = 1)
  beta = fit_ols(e_X, e_y)
  n2   = calculate_n2(e_X, e_y, beta)
```

```
  r2, _ = calculate_r2(e_X, e_y, beta)
  result["bg"] = (e[1:].shape[0] - 1) * r2

  # 1-alpha critical values
  result["l_critical"] = stats.chi2.ppf((1 - significance) / 2, df = 1)
  result["u_critical"] = stats.chi2.ppf(1 - (1 - significance) / 2, df = 1)
  return result

# part l
# assessing normality of residuals
def test_normality(X, y, beta, significance = 0.95):
  y_hat = np.matmul(X, beta)
  e = y - y_hat

  # qq plot
  e_std = (e - np.mean(e)) / np.std(e)
  stats.probplot(e_std, dist = "norm", plot = plt)
  plt.show()

  # jarque-bera test
  skew = np.sum((e - np.mean(e)) ** 3) / e.shape[0] / (np.std(e) ** 3)
  kurtosis = np.sum((e - np.mean(e)) ** 4) / e.shape[0] / (np.std(e) ** 4)
  result = dict()
  result["jb"] = e.shape[0] / 6 * ( skew ** 2  + 0.25 * (kurtosis - 3) ** 2)

  # 1-alpha critical values
  result["l_critical"] = stats.chi2.ppf((1 - significance) / 2, df = 2)
  result["u_critical"] = stats.chi2.ppf(1 - (1 - significance) / 2, df = 2)
  return result

# part m
# multicollinearity based on Q number
def calculate_condition_number(X):
  Q = np.zeros((X.shape[1], X.shape[1]))
  for i in range(X.shape[0]):
    Q = Q + np.matmul(X[i, :], np.transpose(X[i, :]))
  Q /= X.shape[0]
  return np.linalg.cond(Q)

# part n
# test of linear model specification
def assess_linear_model(X, y, beta):
  y_hat = np.matmul(X, beta)
  new_X = np.concatenate([
      X, np.reshape(y_hat ** 2, (X.shape[0], 1))
  ], axis = 1)
  beta = fit_ols(new_X, y)
  V = heteroskedastic_covariance_matrix(new_X, y, beta)
  p_value = test_regression_coefficients(new_X, beta, V)
  return p_value[X.shape[1]]

# part o
# running rolling regressions to test parameter stability
def plot_rolling(X, y, l = 60, alpha = 0.95):
  # (1-alpha) t-values
  t_value = -stats.t.ppf((1 - alpha) / 2, l - X.shape[1])
  betas, lower_bounds, upper_bounds = [], [], []
  for i in range(X.shape[0] - l):
    curr_X, curr_y = X[i: i + l, :], y[i: i + l]
    beta = fit_ols(curr_X, curr_y)
```

```
    beta = fit_ols(curr_X, curr_y)
    V = heteroskedastic_covariance_matrix(curr_X, curr_y, beta)
    sb = V.diagonal() ** 0.5
    betas.append(list(beta))
    lower_bounds.append(list(beta - sb * t_value))
    upper_bounds.append(list(beta + sb * t_value))
  betas, lower_bounds, upper_bounds = np.array(betas), np.array(lower_bounds), np.a
  for i in range(betas.shape[1]):
    plt.figure()
    plt.plot(betas[:, i], color = "red", label = "Estimated Beta")
    plt.plot(lower_bounds[:, i], color = "blue", label = "CI Bound")
    plt.plot(upper_bounds[:, i], color = "blue", label = "CI Bound")
    plt.title("Regression Coefficient {} Rolled on {}-month Windows".format(i + 1,
    plt.show()
```

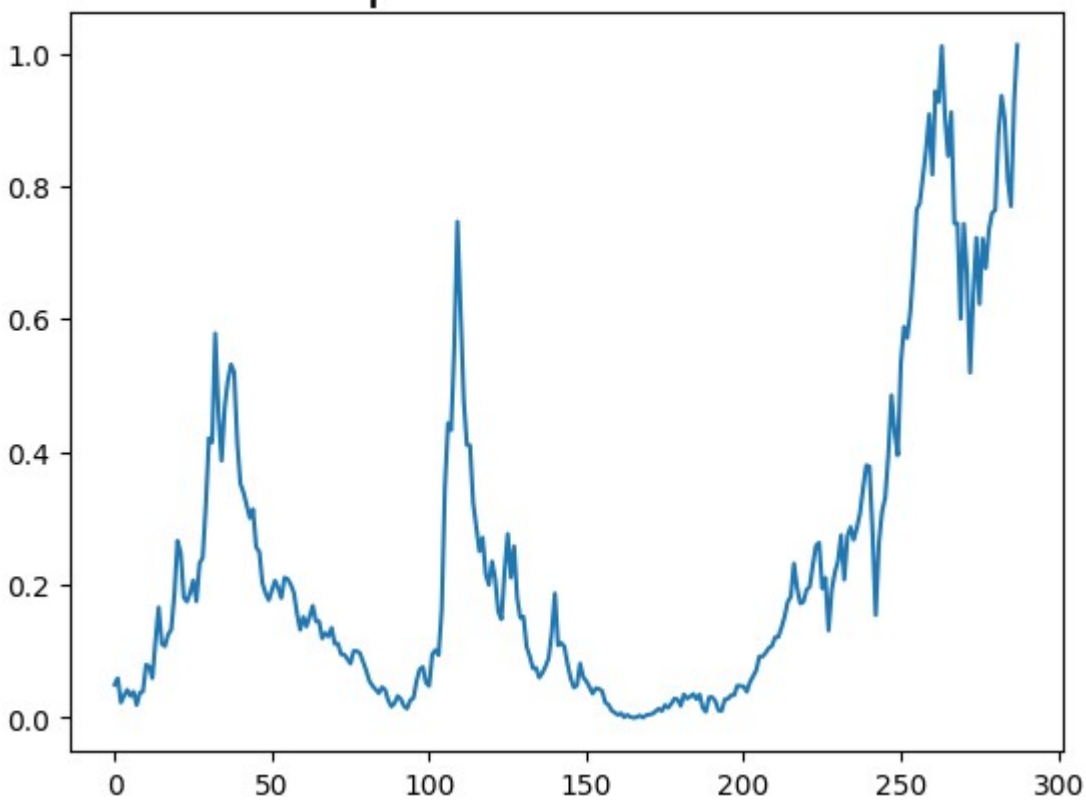
In [30]:

```
# an execution utility for running all the regression functions
def get_regression_results(X, y):
  print("\n----Part a----")
  print("Running the regression...")
  beta = fit_ols(X, y)
  print("Regression Coefficients Beta = {}".format(beta))
  r2, r2_adj = calculate_r2(X, y, beta)
  print("R^2 = {}".format(r2))
  print("Adjusted R^2 = {}".format(r2_adj))

  print("\n----Part c----")
  V_hom = homoskedastic_covariance_matrix(X, y, beta)
  print("Under homoskedasticity assumption, covariance matrix = {}".format(V_hom))

  print("\n----Part d----")
  p_values = test_regression_coefficients(X, beta, V_hom)
  print("H0: beta = 0; H1: beta != 0")
  print("Under homoskedasticity, p-values for regression coefficients = {}".format(

  print("\n----Part e----")
  print("Plotting squared residuals to assess homoskedasticity...")
  assess_homoskedasticity(X, y, beta)

  print("\n----Part f----")
  result_f = get_ci_estimates(X, beta, V_hom)
  print("Under Homoskedasticity assumption,")
  for i in range(beta.shape[0]):
    print("For coeff {},".format(i + 1), end = " ")
    print("Standard Error = {}".format(result_f["standard_errors"][i]), end = " ")
    print("90% CI = {}".format(result_f[0.9][i]), end = " ")
    print("95% CI = {}".format(result_f[0.95][i]), end = " ")
    print("99% CI = {}".format(result_f[0.99][i]))

  print("\n----Part g----")
  V_het = heteroskedastic_covariance_matrix(X, y, beta)
  print("Under heteroskedasticity assumption, covariance matrix = {}".format(V_het)

  print("\n----Part h----")
  result_h = get_ci_estimates(X, beta, V_het)
  print("Under Heteroskedasticity assumption,")
  for i in range(beta.shape[0]):
    print("For coeff {},".format(i + 1), end = " ")
    print("Standard Error = {}".format(result_h["standard_errors"][i]), end = " ")
    print("90% CI = {}".format(result_h[0.9][i]), end = " ")
```

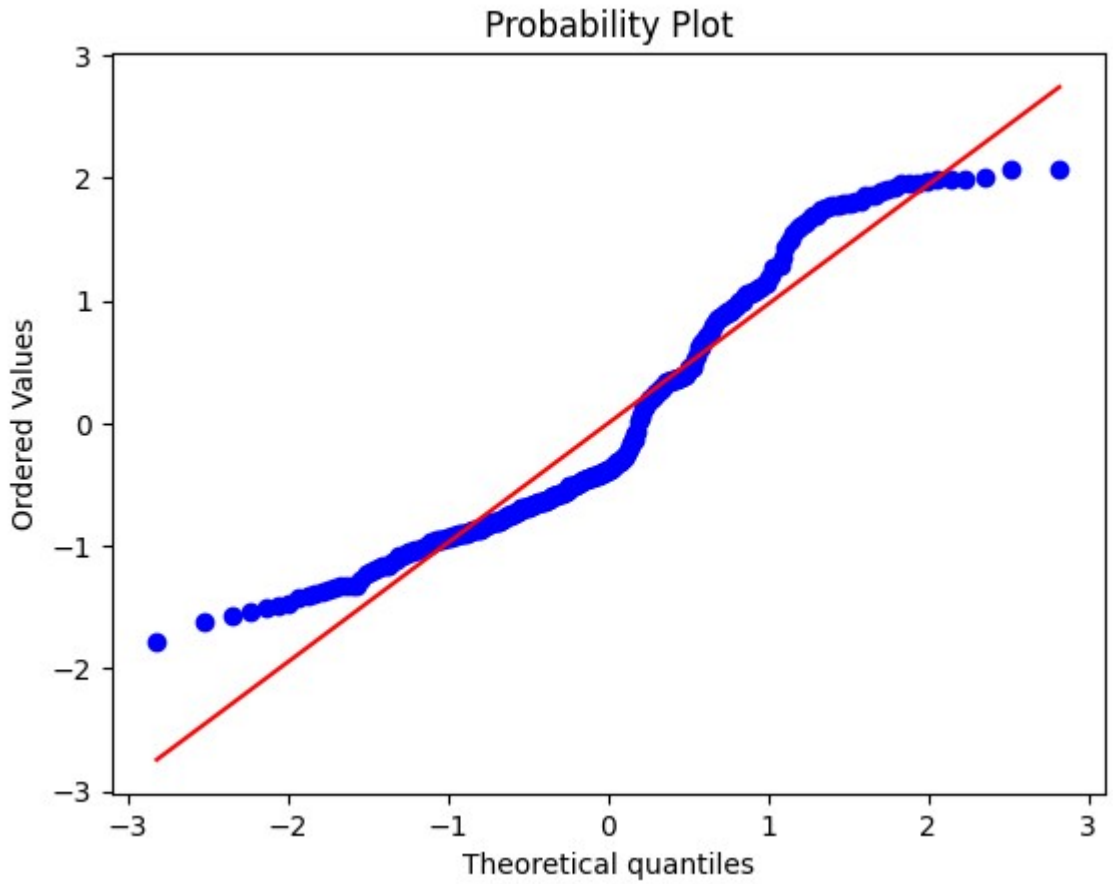
```
    print("95% CI = {}".format(result_h[0.95][i]), end = " ")
    print("99% CI = {}".format(result_h[0.99][i]))

  print("\n----Part i----")
  p_values = test_regression_coefficients(X, beta, V_het)
  print("H0: beta = 0; H1: beta != 0")
  print("Under heteroskedasticity, p-values for regression coefficients = {}".forma

  print("\n----Part j----")
  aic, bic, hqc = calculate_information_criteria(X, y, beta)
  print("AIC = {}".format(aic))
  print("BIC = {}".format(bic))
  print("Hannah-Quinn IC = {}".format(hqc))

  print("\n----Part k----")
  result_k = residual_autocorrelation_tests(X, y, beta)
  print("Durbin-Watson Stastistic = {}".format(result_k["dw"]))
  print("Breusch-Godfrey Statistic = {}".format(result_k["bg"]))
  print("At 95% Significance,", end = " ")
  print("Lower Critical Value for BG Test = {}; Upper Critical Value for BG Test =
      result_k["l_critical"], result_k["u_critical"]
  ))

  print("\n----Part l----")
  print("Plotting the QQ plot of residuals...")
  result_l = test_normality(X, y, beta)
  print("Jarque-Bera Test for Normality at 95% Significance:")
  print("JB-Statistic = {}; Lower Crit Value = {}; Upper Crit Value = {}".format(
      result_l["jb"], result_l["l_critical"], result_l["u_critical"]
```

In [43]:
```
# AR(0) for p_t
get_regression_results(Xpt0, ypt0)
  print("\n----Part m----")
  cond_number = calculate_condition_number(X)
  print("Condition Number = {}".format(cond_number))

  print("\n----Part n----")
  p_value = assess_linear_model(X, y, beta)
  print("With Ramsey RESET test, p-value = {}".format(p_value))

  print("\n----Part o----")
  print("Running rolling regressions...")
  plot_rolling(X, y)
```

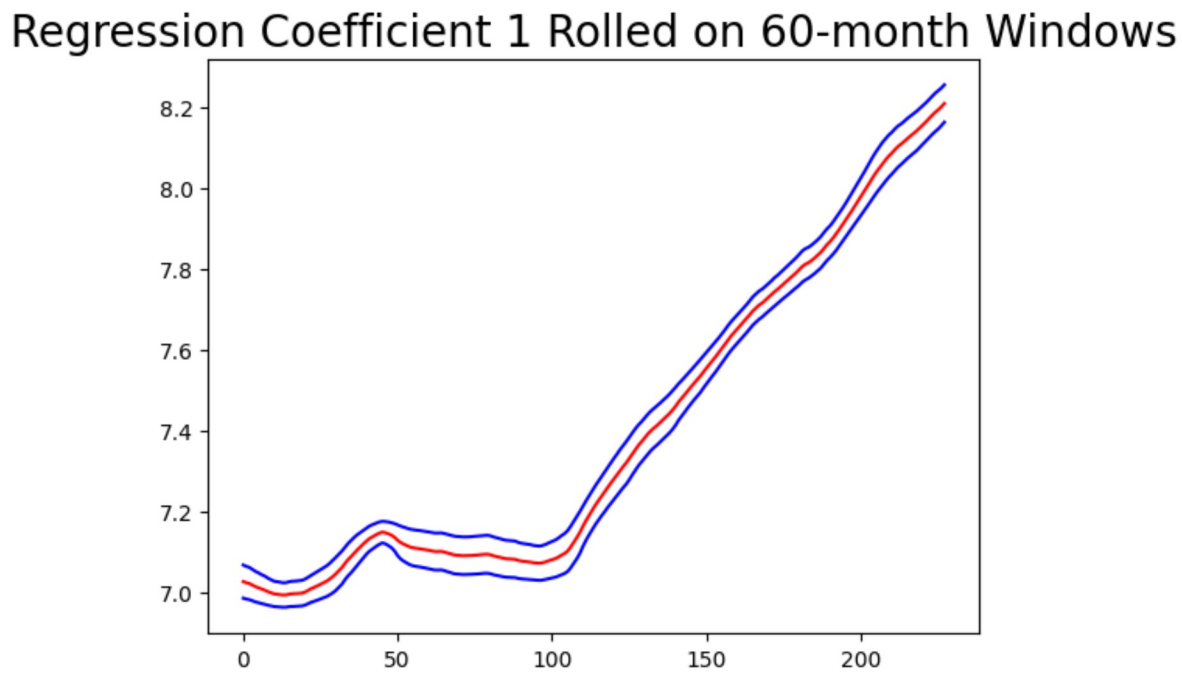
```
H0: beta = 0; H1: beta != 0
Under homoskedasticity, p-values for regression coefficients = [0.]

----Part e----
Plotting squared residuals to assess homoskedasticity...
```


Squared Residuals
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250
300

```
----Part f----
Under Homoskedasticity assumption,
For coeff 1, Standard Error = 0.028596696942588064 90% CI = (7.4168065970930845, 7.
511186003866501) 95% CI = (7.407710448142467, 7.520282152817119) 99% CI = (7.389843
123441692, 7.538149477517893)

----Part g----
Under heteroskedasticity assumption, covariance matrix = [[0.00081777]]

----Part h----
Under Heteroskedasticity assumption,
For coeff 1, Standard Error = 0.02859669694258806 90% CI = (7.4168065970930845, 7.5
11186003866501) 95% CI = (7.407710448142467, 7.520282152817119) 99% CI = (7.3898431
23441693, 7.5381494775178925)

----Part i----
H0: beta = 0; H1: beta != 0
Under heteroskedasticity, p-values for regression coefficients = [0.]

----Part j----
AIC = 515.739830445928
BIC = 519.4027909260639
Hannah-Quinn IC = 517.2077240629242

----Part k----
Durbin-Watson Stastistic = 0.008603197811787571
Breusch-Godfrey Statistic = 283.56408758669744
At 95% Significance, Lower Critical Value for BG Test = 0.0009820691171752583; Uppe
r Critical Value for BG Test = 5.023886187314888

----Part l----
Plotting the QQ plot of residuals...
```

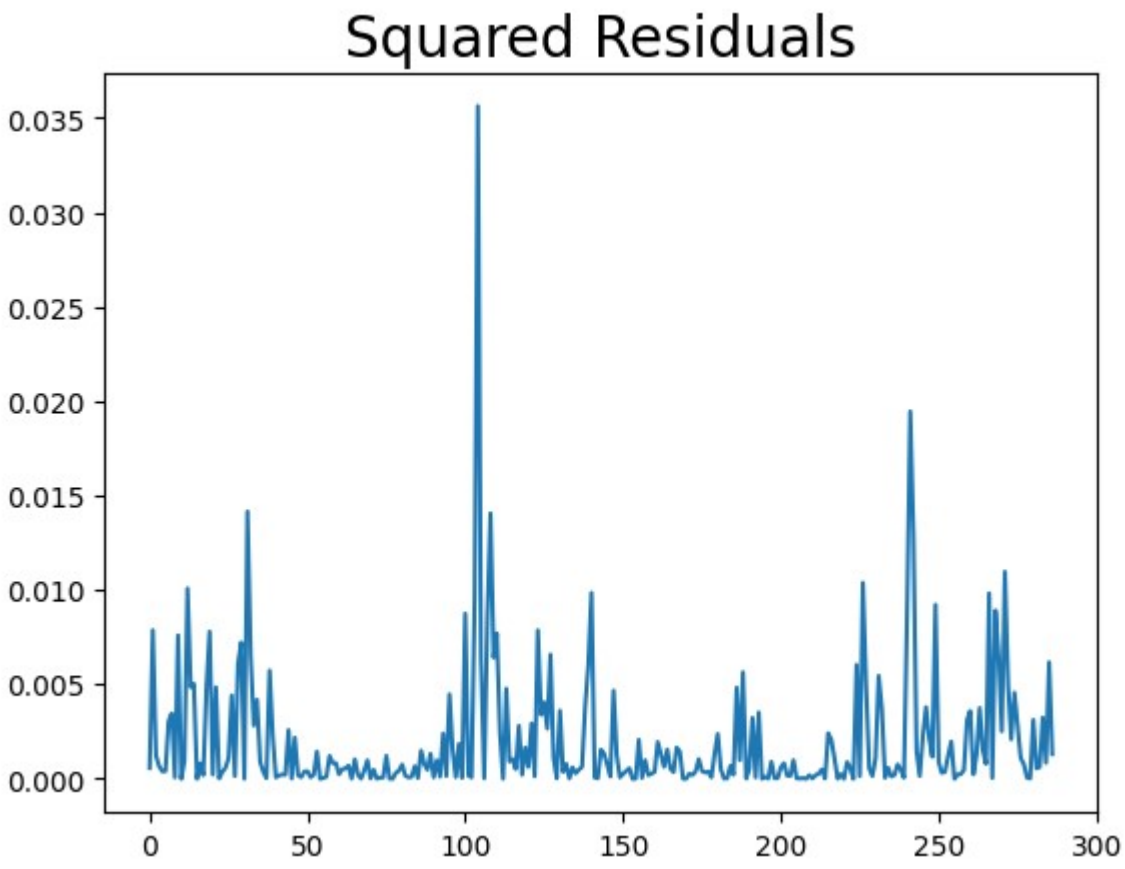



```
Jarque-Bera Test for Normality at 95% Significance:
JB-Statistic = 23.243083194295167; Lower Crit Value = 0.050635615968579795; Upper C
rit Value = 7.377758908227871

----Part m----
Condition Number = 1.0

----Part n----
With Ramsey RESET test, p-value = 0.9999898561955356

----Part o----
Running rolling regressions...
```

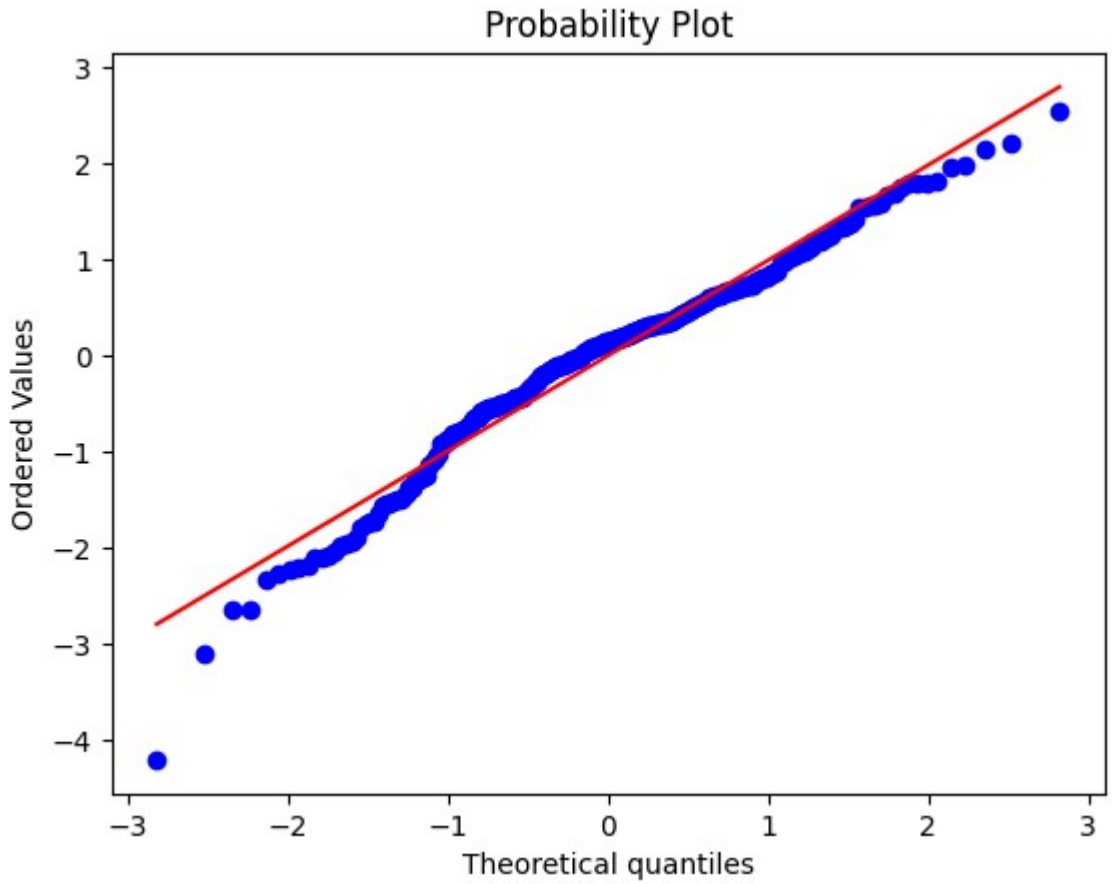


In [44]:
```
# AR(1) for p_t
get_regression_results(Xpt1, ypt1)
```

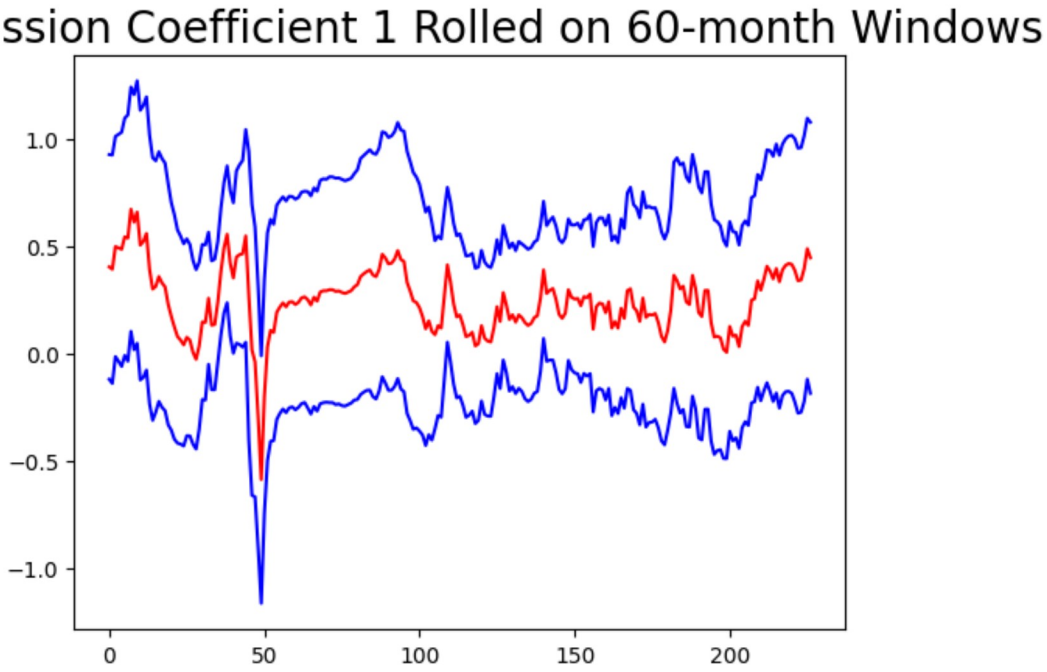
```
----Part a----
Running the regression...
Regression Coefficients Beta = [-0.0173563   1.00290079]
R^2 = 0.9914828237296432
Adjusted R^2 = 0.9914228436150633

----Part c----
Under homoskedasticity assumption, covariance matrix = [[ 1.68264303e-03 -2.2460110
7e-04]
 [-2.24601107e-04  3.01054062e-05]]

----Part d----
H0: beta = 0; H1: beta != 0
Under homoskedasticity, p-values for regression coefficients = [0.6725286 0.
]

----Part e----
Plotting squared residuals to assess homoskedasticity...
```

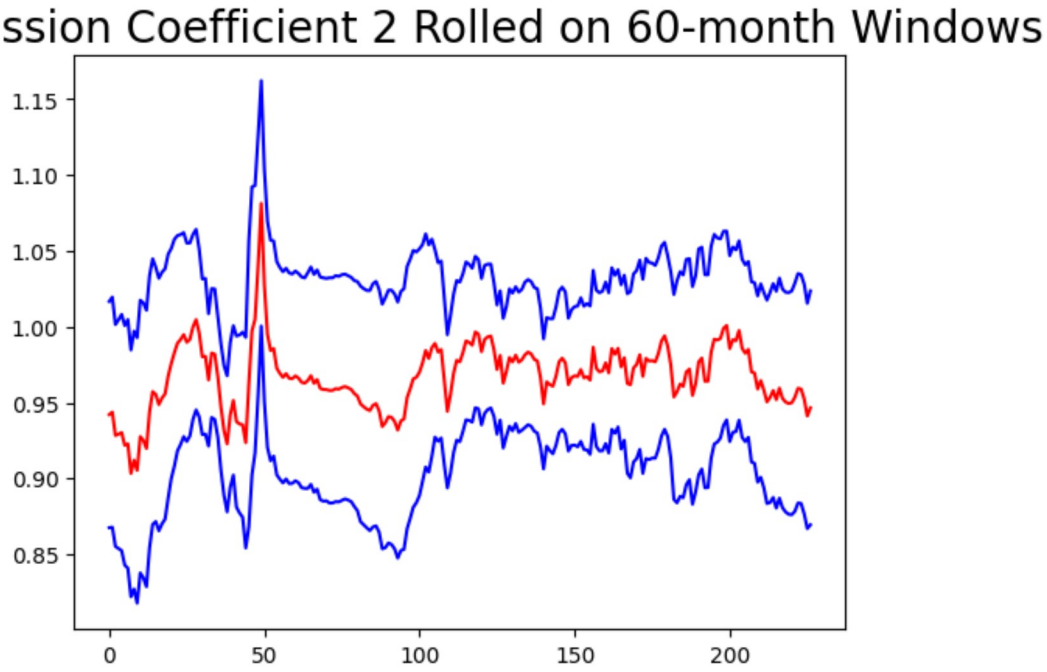

Squared Residuals
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
50
100
150
200
250
300

```
----Part f----
Under Homoskedasticity assumption,
For coeff 1, Standard Error = 0.04102003202737872 90% CI = (-0.08504827874124446,
0.05033568811254152) 95% CI = (-0.0980969515307889, 0.06338436090208596) 99% CI =
(-0.12372901753122718, 0.08901642690252425)
For coeff 2, Standard Error = 0.005486839364073687 90% CI = (0.9938463115151263, 1.
0119552705257384) 95% CI = (0.9921009210809403, 1.0137006609599244) 99% CI = (0.988
6723760076486, 1.0171292060332162)

----Part g----
Under heteroskedasticity assumption, covariance matrix = [[ 1.68264303e-03 -2.24601
107e-04]
 [-2.24601107e-04  3.01054062e-05]]

----Part h----
Under Heteroskedasticity assumption,
For coeff 1, Standard Error = 0.04102003202738807 90% CI = (-0.08504827874125989,
0.05033568811255695) 95% CI = (-0.0980969515308073, 0.06338436090210436) 99% CI =
(-0.12372901753125143, 0.08901642690254849)
For coeff 2, Standard Error = 0.005486839364074903 90% CI = (0.9938463115151243, 1.
0119552705257404) 95% CI = (0.992100921080938, 1.0137006609599268) 99% CI = (0.9886
723760076453, 1.0171292060332193)

----Part i----
H0: beta = 0; H1: beta != 0
Under heteroskedasticity, p-values for regression coefficients = [0.6725286 0.
]

----Part j----
AIC = -2217.967519693356
BIC = -2210.6485552618365
Hannah-Quinn IC = -2215.0341900664926

----Part k----
Durbin-Watson Stastistic = 1.9553969873159043
Breusch-Godfrey Statistic = 0.1339915285101467
At 95% Significance, Lower Critical Value for BG Test = 0.0009820691171752583; Uppe
r Critical Value for BG Test = 5.023886187314888

----Part l----
Plotting the QQ plot of residuals...
```



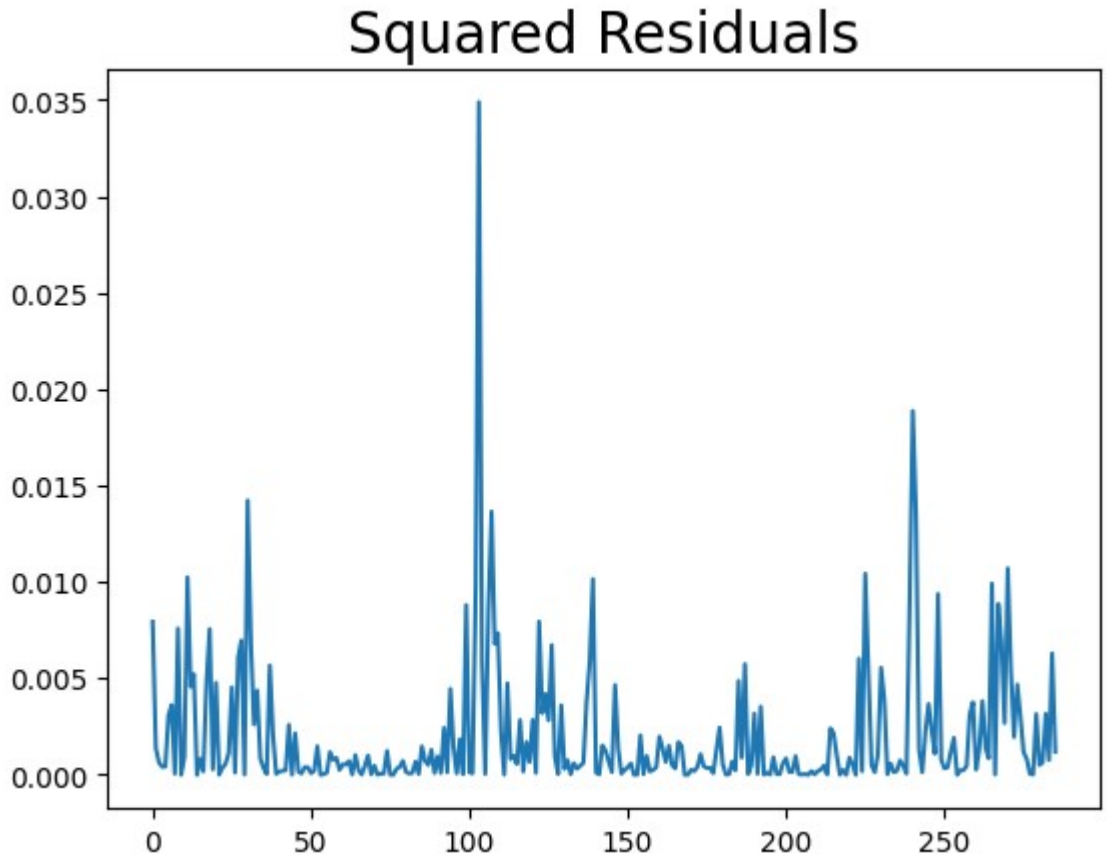

```
Jarque-Bera Test for Normality at 95% Significance:
JB-Statistic = 33.52333997064332; Lower Crit Value = 0.050635615968579795; Upper Cr
it Value = 7.377758908227871

----Part m----
Condition Number = 1.1323322629987092e+16

----Part n----
With Ramsey RESET test, p-value = 0.8852091163992699

----Part o----
Running rolling regressions...
```



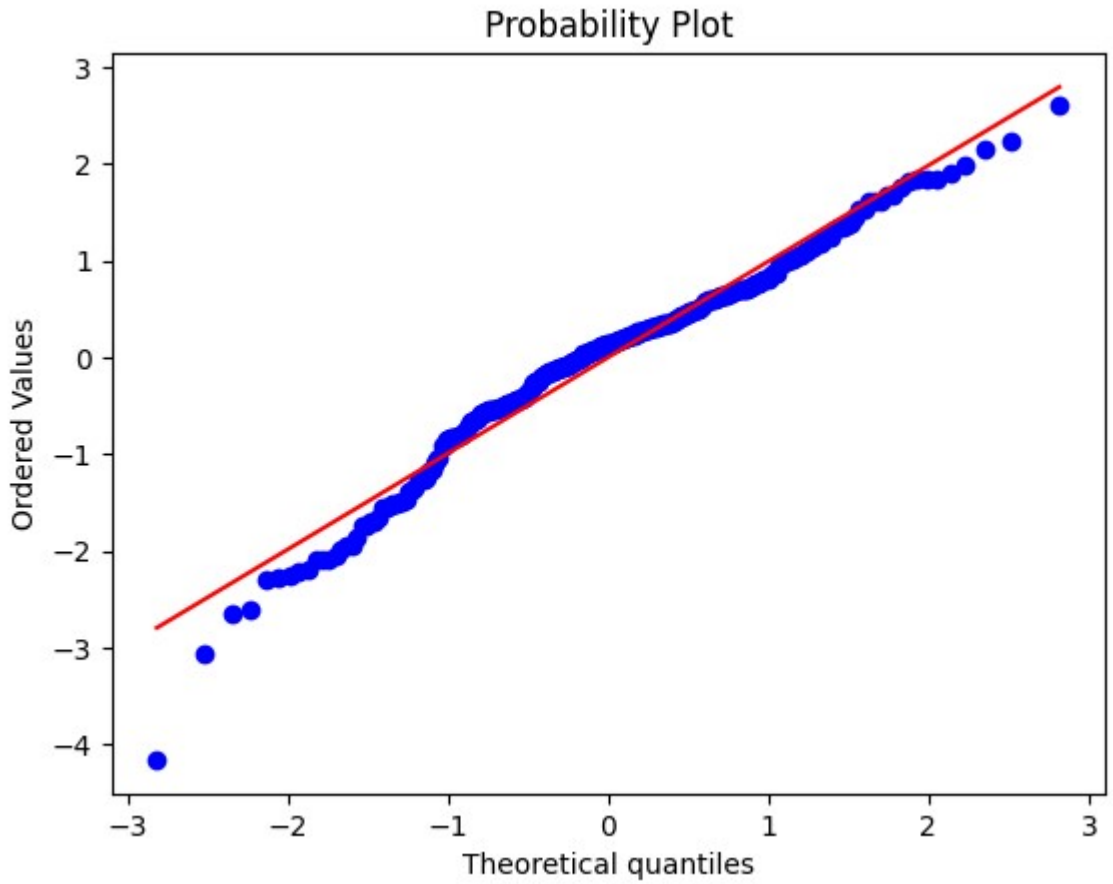


In [45]:
```
# AR(2) for p_t
get_regression_results(Xpt2, ypt2)
```

```
----Part a----
Running the regression...
Regression Coefficients Beta = [-0.01497145  1.02458182 -0.02200167]
R^2 = 0.9914878335437375
Adjusted R^2 = 0.9913972785814368

----Part c----
Under homoskedasticity assumption, covariance matrix = [[ 1.70903828e-03  4.7278347
9e-05 -2.75541940e-04]
 [ 4.72783479e-05  3.49942658e-03 -3.50771349e-03]
 [-2.75541940e-04 -3.50771349e-03  3.54661519e-03]]

----Part d----
H0: beta = 0; H1: beta != 0
Under homoskedasticity, p-values for regression coefficients = [0.71751062 0.
0.71207311]

----Part e----
Plotting squared residuals to assess homoskedasticity...
```

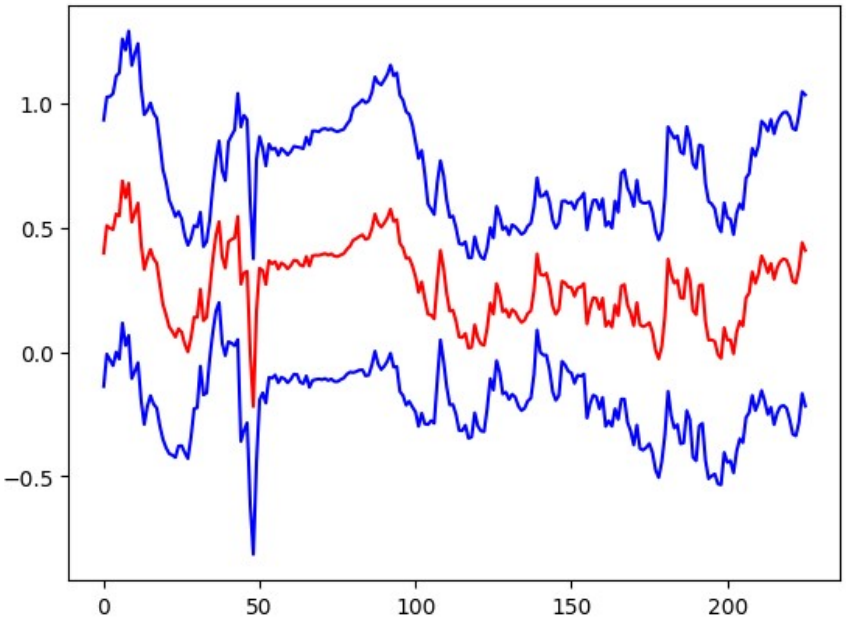


```
----Part f----
Under Homoskedasticity assumption,
For coeff 1, Standard Error = 0.041340516186721916 90% CI = (-0.08319387009572356,
0.05325097870318967) 95% CI = (-0.09634537046355382, 0.06640247907101993) 99% CI =
(-0.12218034841145536, 0.09223745701892147)
For coeff 2, Standard Error = 0.05915595138279123 90% CI = (0.9269593746613889, 1.1
222042720769003) 95% CI = (0.9081403180038427, 1.1410233287344467) 99% CI = (0.8711
719189505087, 1.1779917277877805)
For coeff 3, Standard Error = 0.05955346495807591 90% CI = (-0.12028011684475541,
0.07627677869458822) 95% CI = (-0.13922563297629473, 0.09522229482612754) 99% CI =
(-0.1764424506610044, 0.13243911251083718)

----Part g----
Under heteroskedasticity assumption, covariance matrix = [[ 1.70903828e-03  4.72783
479e-05 -2.75541940e-04]
 [ 4.72783479e-05  3.49942658e-03 -3.50771349e-03]
 [-2.75541940e-04 -3.50771349e-03  3.54661519e-03]]

----Part h----
Under Heteroskedasticity assumption,
For coeff 1, Standard Error = 0.04134051618671634 90% CI = (-0.08319387009571434,
0.053250978703180456) 95% CI = (-0.09634537046354284, 0.06640247907100895) 99% CI =
(-0.12218034841144089, 0.092237457018907)
For coeff 2, Standard Error = 0.059155951382297324 90% CI = (0.926959374662204, 1.1
222042720760852) 95% CI = (0.9081403180048149, 1.1410233287334743) 99% CI = (0.8711
719189517895, 1.1779917277864997)
For coeff 3, Standard Error = 0.05955346495750659 90% CI = (-0.12028011684381587,
0.07627677869364868) 95% CI = (-0.1392256329751741, 0.0952222948250069) 99% CI =
(-0.17644245065952796, 0.13243911250936075)

----Part i----
H0: beta = 0; H1: beta != 0
Under heteroskedasticity, p-values for regression coefficients = [0.71751062 0.
0.71207311]

----Part j----
AIC = -2207.0733510636746
BIC = -2196.105375631215
Hannah-Quinn IC = -2202.6770581793644

----Part k----
Durbin-Watson Stastistic = 2.0011960530106454
Breusch-Godfrey Statistic = 0.007922380415402438
At 95% Significance, Lower Critical Value for BG Test = 0.0009820691171752583; Uppe
r Critical Value for BG Test = 5.023886187314888

----Part l----
Plotting the QQ plot of residuals...
```

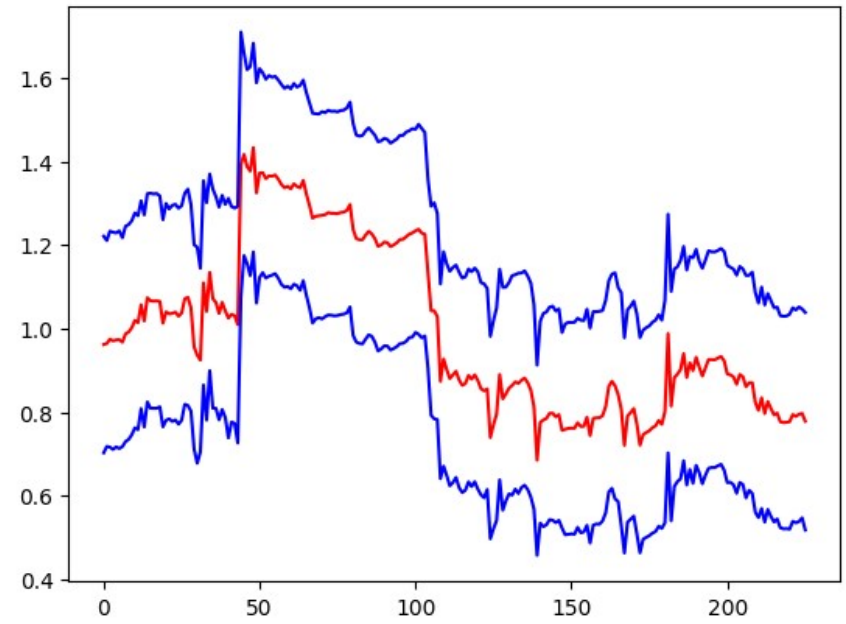


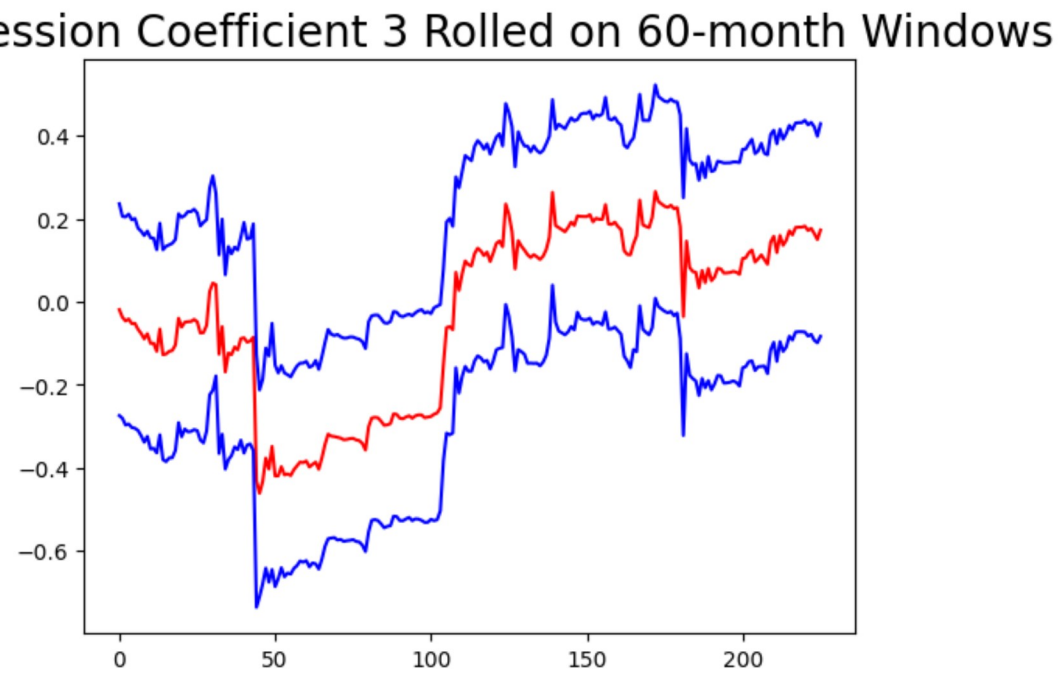

```
Jarque-Bera Test for Normality at 95% Significance:
JB-Statistic = 30.511880758724264; Lower Crit Value = 0.050635615968579795; Upper C
rit Value = 7.377758908227871

----Part m----
Condition Number = inf

----Part n----
With Ramsey RESET test, p-value = 0.8893912405652264

----Part o----
Running rolling regressions...
```

Regression Coefficient 1 Rolled on 60-month Windows



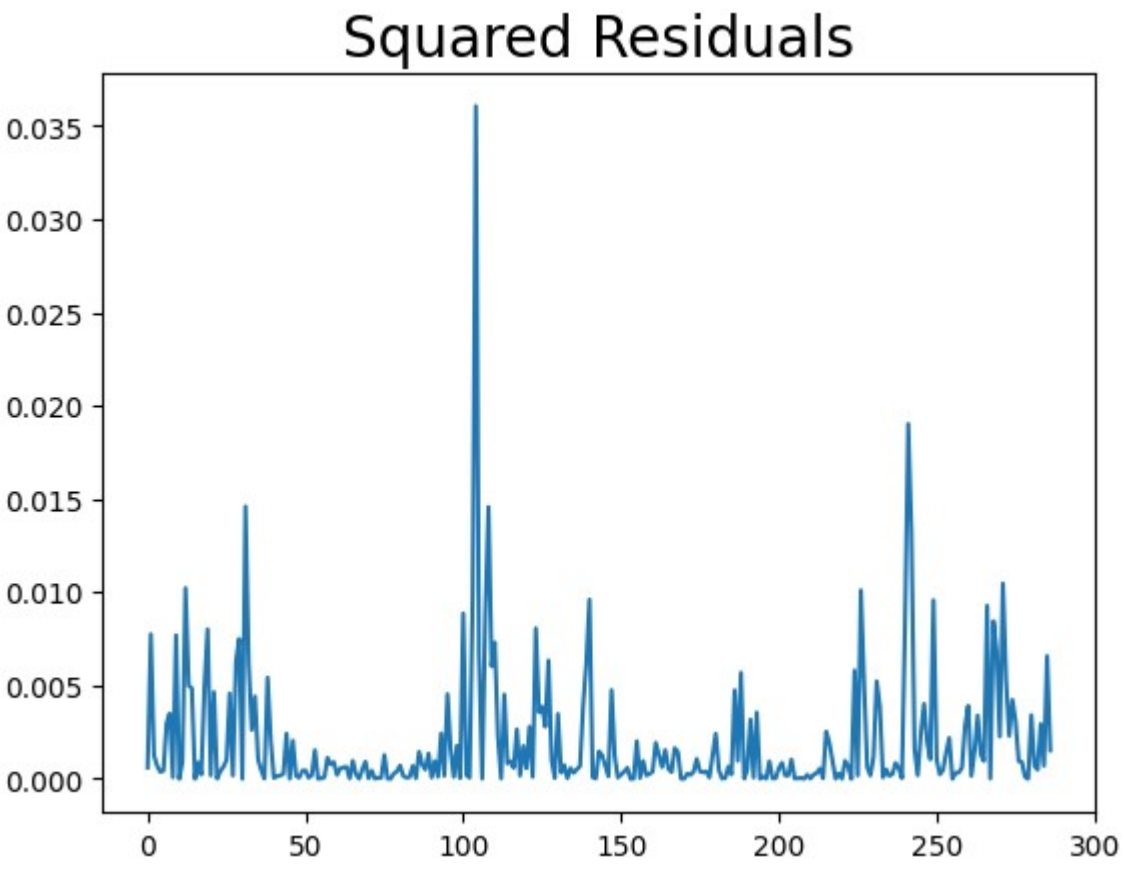
Regression Coefficient 2 Rolled on 60-month Windows



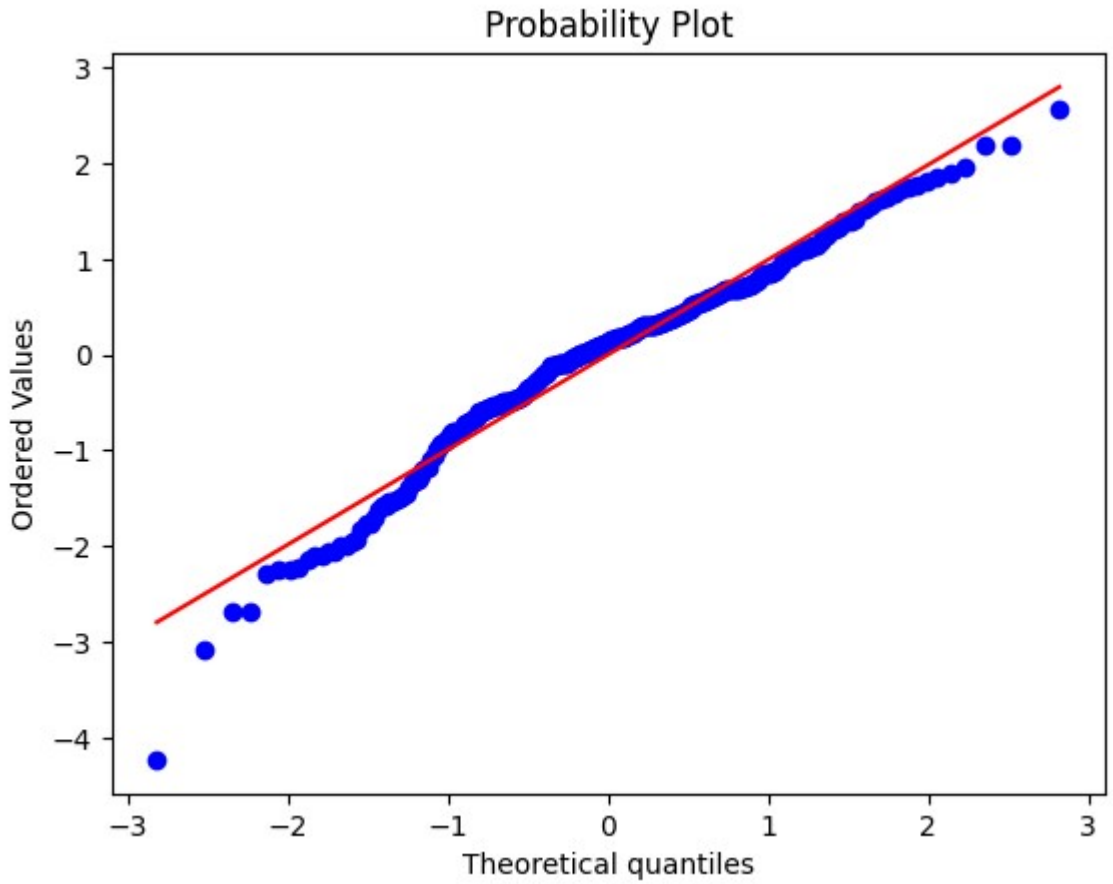


In [40]:
```
# AR(0) for r_t
get_regression_results(Xrt0, yrt0)
```

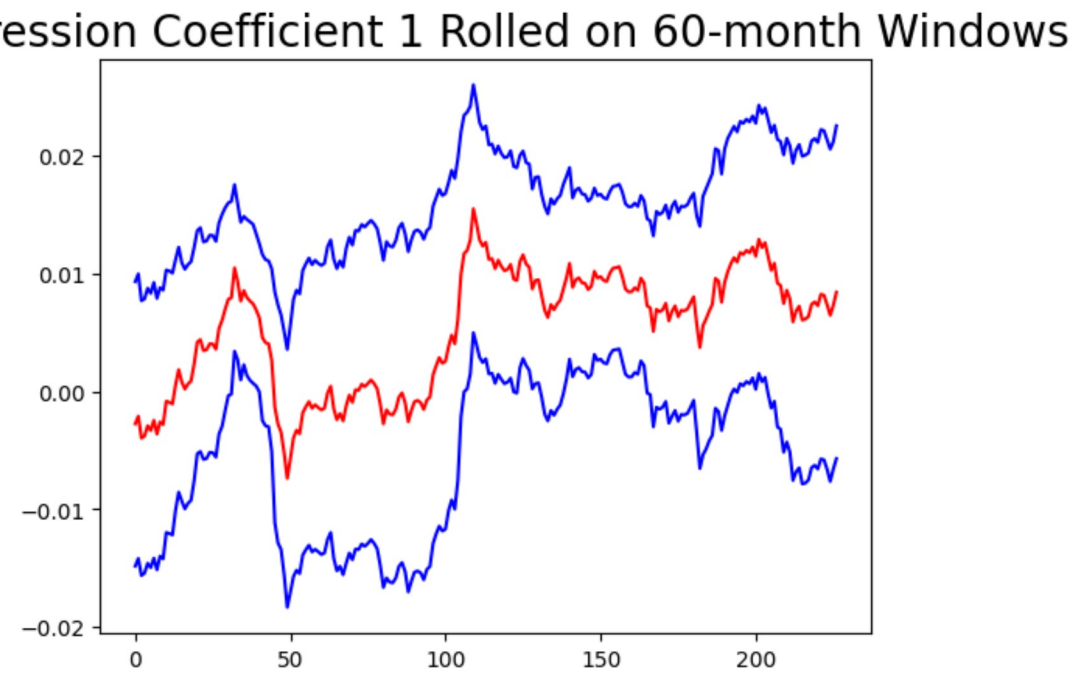
```
----Part a----
Running the regression...
Regression Coefficients Beta = [0.00428503]
R^2 = 0.0
Adjusted R^2 = -0.0035087719298245723

----Part c----
Under homoskedasticity assumption, covariance matrix = [[7.01535369e-06]]

----Part d----
H0: beta = 0; H1: beta != 0
Under homoskedasticity, p-values for regression coefficients = [0.10680465]

----Part e----
Plotting squared residuals to assess homoskedasticity...
```


Squared Residuals
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
50
100
150
200
250
300

```
----Part f----
Under Homoskedasticity assumption,
For coeff 1, Standard Error = 0.002648651295586999 90% CI = (-8.577194652394874e-0
5, 0.008655830940851057) 95% CI = (-0.000928292983432044, 0.009498351977759154) 99%
CI = (-0.0025832638230515086, 0.011153322817378616)

----Part g----
Under heteroskedasticity assumption, covariance matrix = [[7.01535369e-06]]

----Part h----
Under Heteroskedasticity assumption,
For coeff 1, Standard Error = 0.0026486512955869973 90% CI = (-8.577194652394614e-0
5, 0.008655830940851055) 95% CI = (-0.0009282929834320405, 0.00949835197775915) 99%
CI = (-0.0025832638230515043, 0.011153322817378613)

----Part i----
H0: beta = 0; H1: beta != 0
Under heteroskedasticity, p-values for regression coefficients = [0.10680465]

----Part j----
AIC = -2219.4087832501255
BIC = -2215.749301034366
Hannah-Quinn IC = -2217.942118436694

----Part k----
Durbin-Watson Stastistic = 1.9479458816454085
Breusch-Godfrey Statistic = 0.18185074388316322
At 95% Significance, Lower Critical Value for BG Test = 0.0009820691171752583; Uppe
r Critical Value for BG Test = 5.023886187314888

----Part l----
Plotting the QQ plot of residuals...
```

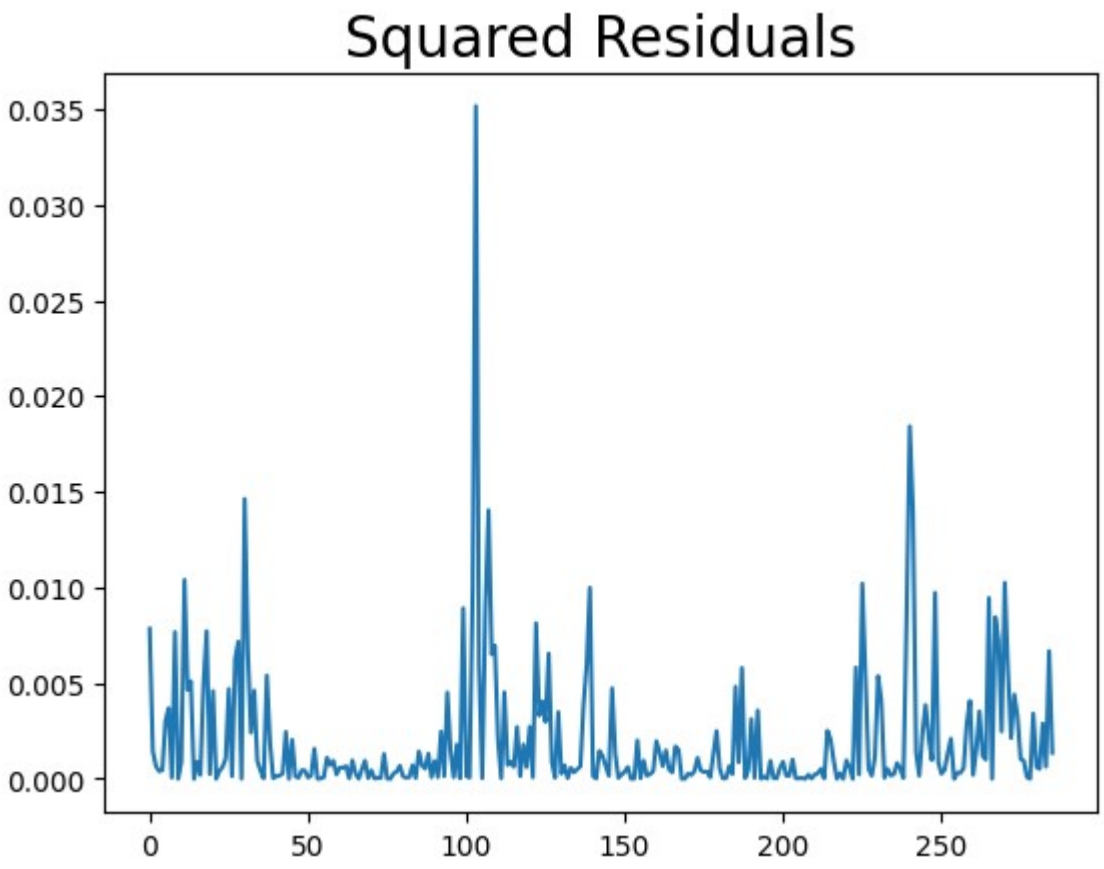



```
Jarque-Bera Test for Normality at 95% Significance:
JB-Statistic = 34.19052355408686; Lower Crit Value = 0.050635615968579795; Upper Cr
it Value = 7.377758908227871

----Part m----
Condition Number = 1.0

----Part n----
With Ramsey RESET test, p-value = 0.9999999664380583

----Part o----
Running rolling regressions...
```

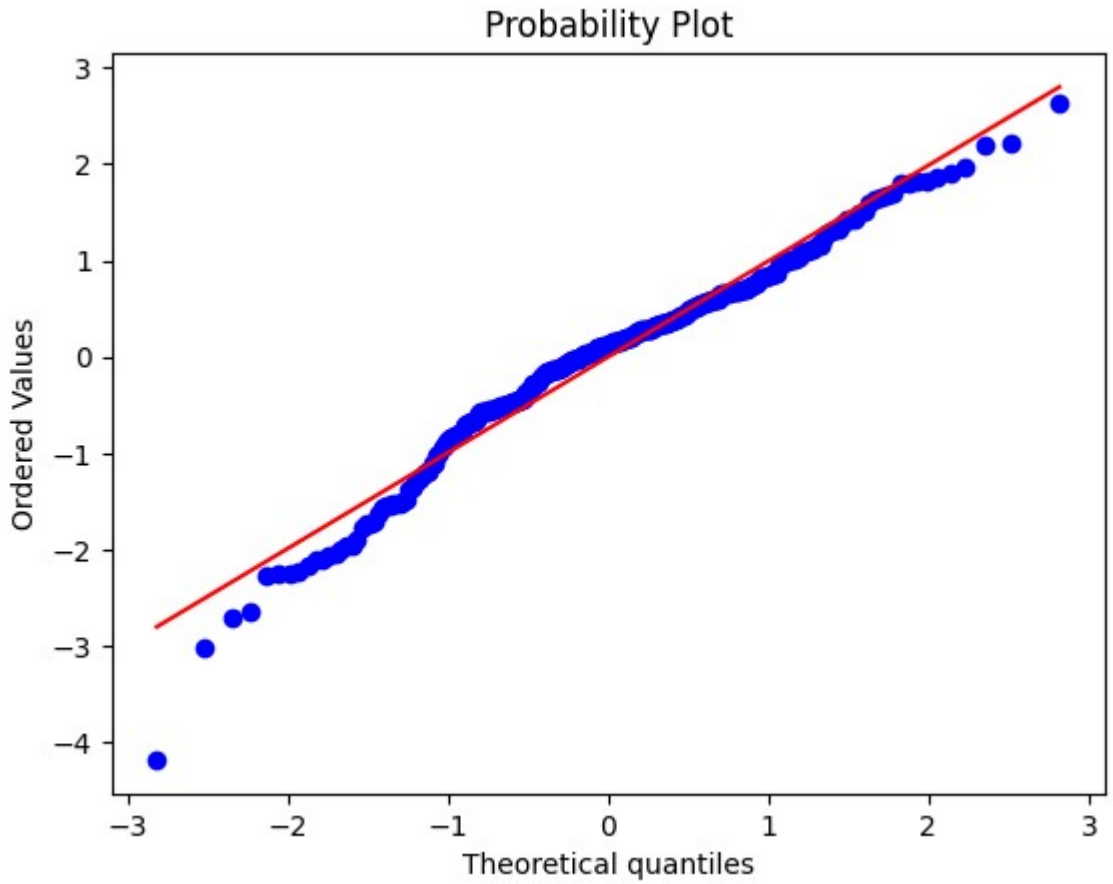


In [41]:
```
# AR(1) for r_t
get_regression_results(Xrt1, yrt1)
```

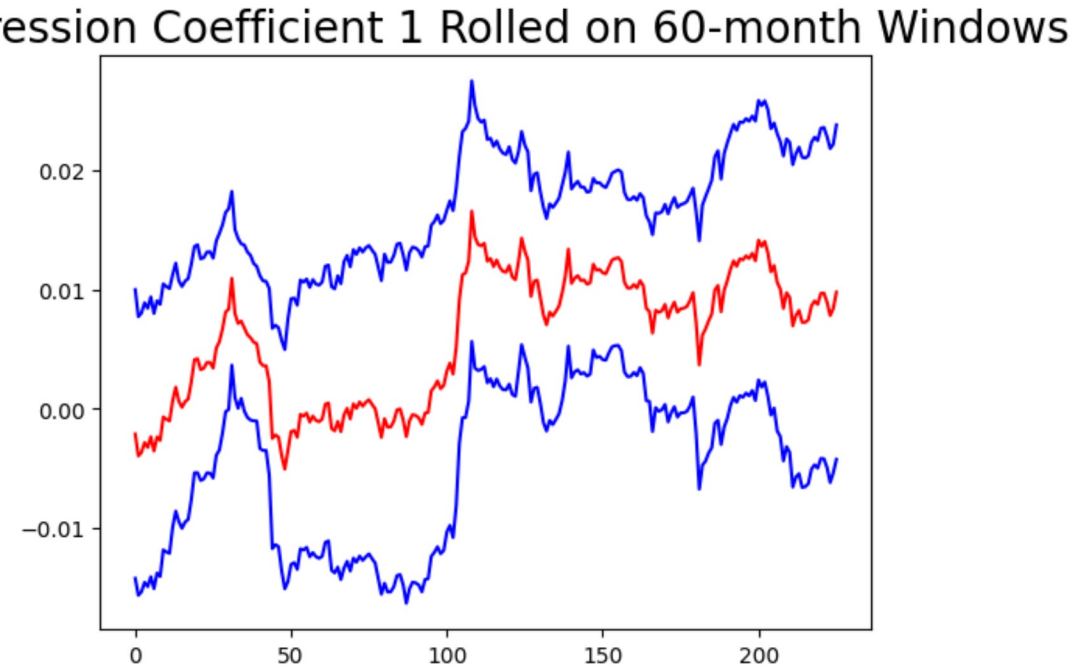
```
----Part a----
Running the regression...
Regression Coefficients Beta = [0.00426616 0.02528023]
R^2 = 0.000638072785554962
Adjusted R^2 = -0.006424555675324628

----Part c----
Under homoskedasticity assumption, covariance matrix = [[ 7.11280975e-06 -1.4519750
1e-05]
 [-1.45197501e-05  3.49984242e-03]]

----Part d----
H0: beta = 0; H1: beta != 0
Under homoskedasticity, p-values for regression coefficients = [0.11079524 0.669467
56]

----Part e----
Plotting squared residuals to assess homoskedasticity...
```

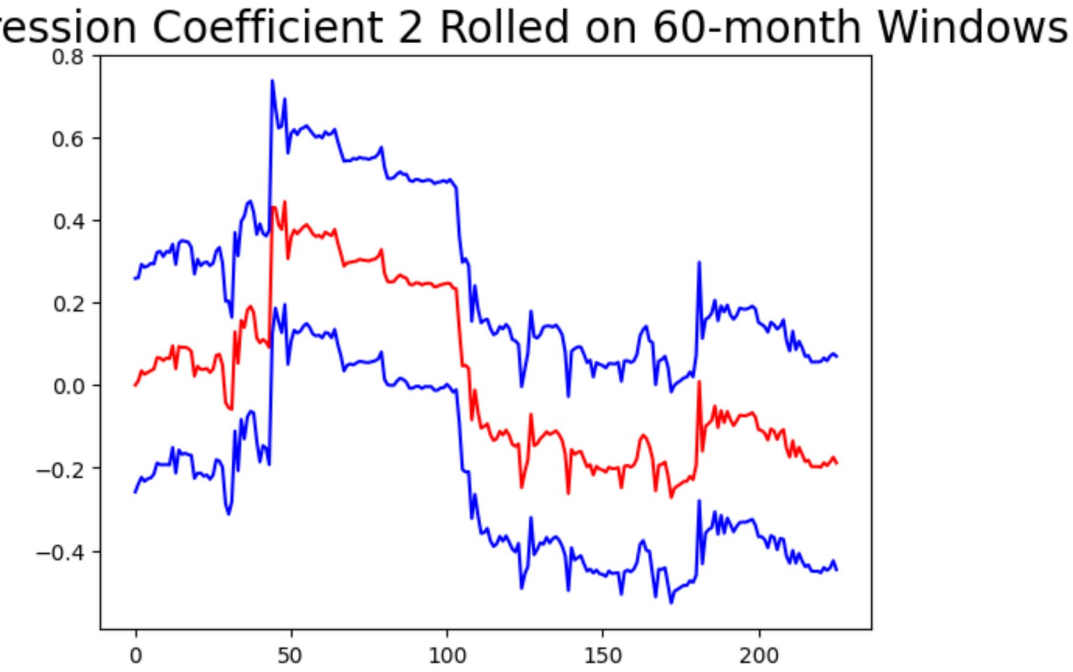

Squared Residuals
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
50
100
150
200
250

```
----Part f----
Under Homoskedasticity assumption,
For coeff 1, Standard Error = 0.0026669851427442485 90% CI = (-0.000134999804268028
27, 0.008667313517024364) 95% CI = (-0.000983409105642547, 0.00951572281839888) 99%
CI = (-0.002650005456670381 6, 0.011182319169426715)
For coeff 2, Standard Error = 0.05915946605283193 90% CI = (-0.07234689258888154,
0.12290734702850205) 95% CI = (-0.09116643547239651, 0.141726889912017) 99% CI =
(-0.12813512670063243, 0.1786955811402529)

----Part g----
Under heteroskedasticity assumption, covariance matrix = [[ 7.11280975e-06 -1.45197
501e-05]
 [-1.45197501e-05  3.49984242e-03]]

----Part h----
Under Heteroskedasticity assumption,
For coeff 1, Standard Error = 0.002666985142744247 90% CI = (-0.000134999804268025
67, 0.00866731351702436) 95% CI = (-0.0009834091056425444, 0.009515722818398879) 99%
CI = (-0.002650005456670378, 0.011182319169426712)
For coeff 2, Standard Error = 0.05915946605283194 90% CI = (-0.07234689258888156,
0.12290734702850208) 95% CI = (-0.09116643547239654, 0.14172688991201704) 99% CI =
(-0.12813512670063246, 0.17869558114025294)

----Part i----
H0: beta = 0; H1: beta != 0
Under heteroskedasticity, p-values for regression coefficients = [0.11079524 0.6694
6756]

----Part j----
AIC = -2208.6386157406832
BIC = -2201.3266321190436
Hannah-Quinn IC = -2205.7077538178096

----Part k----
Durbin-Watson Stastistic = 2.0003016041553976
Breusch-Godfrey Statistic = 0.00858403745571333
At 95% Significance, Lower Critical Value for BG Test = 0.0009820691171752583; Uppe
r Critical Value for BG Test = 5.023886187314888

----Part l----
Plotting the QQ plot of residuals...
```



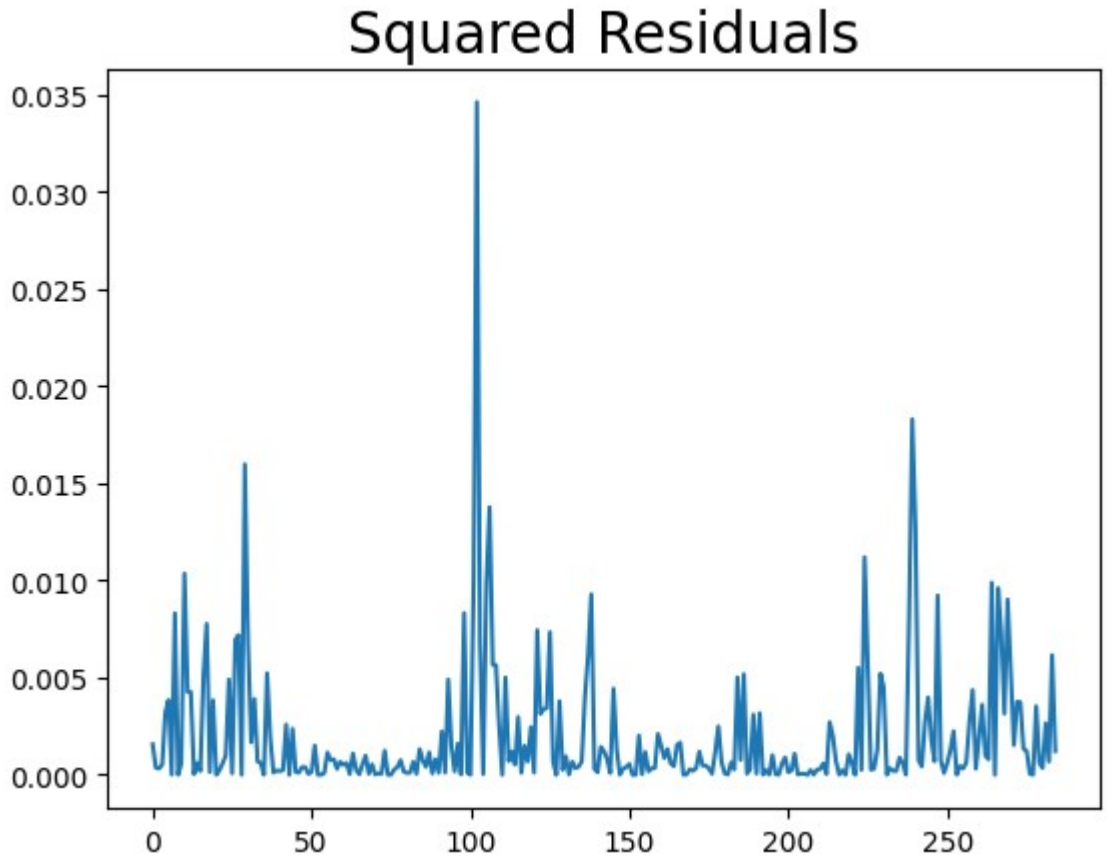

```
Jarque-Bera Test for Normality at 95% Significance:
JB-Statistic = 30.587196059725468; Lower Crit Value = 0.050635615968579795; Upper C
rit Value = 7.377758908227871

----Part m----
Condition Number = 1.276399128362768e+16

----Part n----
With Ramsey RESET test, p-value = 0.7710025938450782

----Part o----
Running rolling regressions...
```



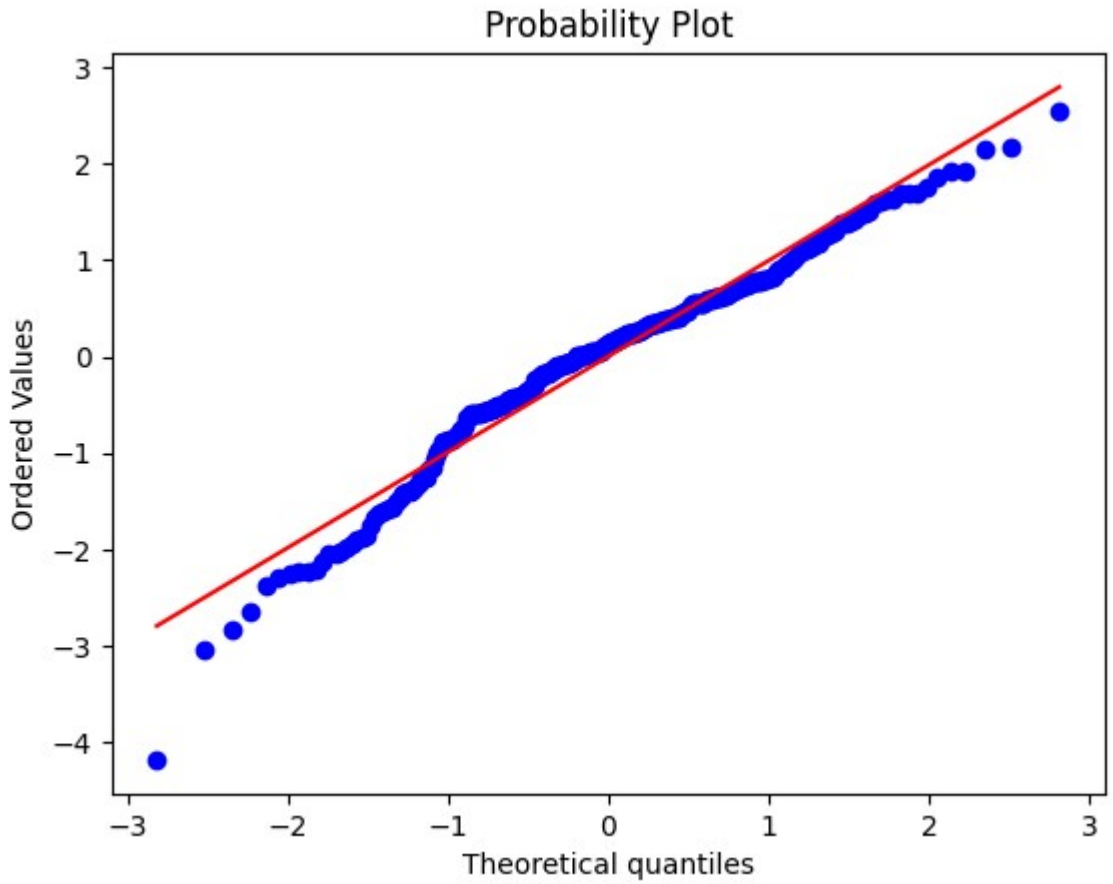


In [42]:
```
# AR(2) for r_t
get_regression_results(Xrt2, yrt2)
```

```
----Part a----
Running the regression...
Regression Coefficients Beta = [ 0.00419484  0.03039451 -0.06758422]
R^2 = 0.0054210927139803165
Adjusted R^2 = -0.005197187434980766

----Part c----
Under homoskedasticity assumption, covariance matrix = [[ 7.08485746e-06 -1.4356049
7e-05 -1.31904087e-05]
 [-1.43560497e-05  3.45327173e-03 -6.90794718e-05]
 [-1.31904087e-05 -6.90794718e-05  3.48981403e-03]]

----Part d----
H0: beta = 0; H1: beta != 0
Under homoskedasticity, p-values for regression coefficients = [0.11615255 0.605404
31 0.25357399]

----Part e----
Plotting squared residuals to assess homoskedasticity...
```

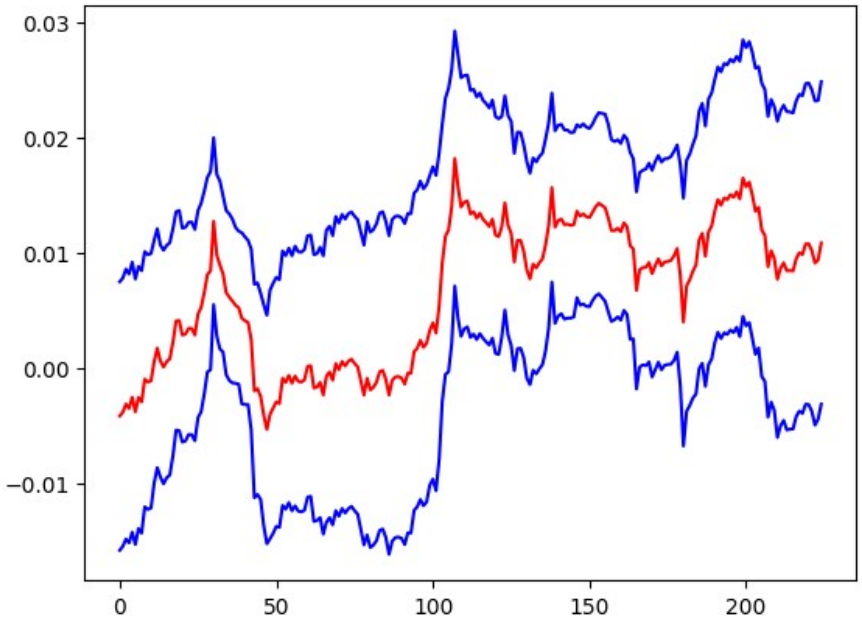


```
----Part f----
Under Homoskedasticity assumption,
For coeff 1, Standard Error = 0.002661739556004666 90% CI = (-0.0001977654320294520
6, 0.00858743888037634) 95% CI = (-0.001044563093899266, 0.009434236542246156) 99%
CI = (-0.0027080534503351383, 0.011097726898682027)
For coeff 2, Standard Error = 0.05876454479773083 90% CI = (-0.06658314573358054,
0.12737216864304637) 95% CI = (-0.0852783178526198, 0.14606734076208563) 99% CI =
(-0.12200401979097943, 0.18279304270044525)
For coeff 3, Standard Error = 0.05907464790684676 90% CI = (-0.1650736322648637, 0.
0299051929076823) 95% CI = (-0.18386745963630946, 0.048699020279128036) 99% CI =
(-0.22078696473258938, 0.08561852537540796)

----Part g----
Under heteroskedasticity assumption, covariance matrix = [[ 7.08485746e-06 -1.43560
497e-05 -1.31904087e-05]
 [-1.43560497e-05  3.45327173e-03 -6.90794718e-05]
 [-1.31904087e-05 -6.90794718e-05  3.48981403e-03]]

----Part h----
Under Heteroskedasticity assumption,
For coeff 1, Standard Error = 0.0026617395560046653 90% CI = (-0.000197765432029451
2, 0.00858743888037634) 95% CI = (-0.0010445630938992643, 0.009434236542246153) 99%
CI = (-0.0027080534503351357, 0.011097726898682025)
For coeff 2, Standard Error = 0.0587645447977308 90% CI = (-0.06658314573358048, 0.
12737216864304632) 95% CI = (-0.08527831785261974, 0.14606734076208558) 99% CI =
(-0.12200401979097934, 0.18279304270044516)
For coeff 3, Standard Error = 0.059074647906846744 90% CI = (-0.1650736322648637,
0.029905192907682274) 95% CI = (-0.18386745963630943, 0.04869902027912801) 99% CI =
(-0.22078696473258935, 0.08561852537540793)

----Part i----
H0: beta = 0; H1: beta != 0
Under heteroskedasticity, p-values for regression coefficients = [0.11615255 0.6054
0431 0.25357399]

----Part j----
AIC = -2207.357690303989
BIC = -2196.400222763183
Hannah-Quinn IC = -2202.9651142378784

----Part k----
Durbin-Watson Stastistic = 1.9764748193212895
Breusch-Godfrey Statistic = 0.04127019151650278
At 95% Significance, Lower Critical Value for BG Test = 0.0009820691171752583; Uppe
r Critical Value for BG Test = 5.023886187314888

----Part l----
Plotting the QQ plot of residuals...
```

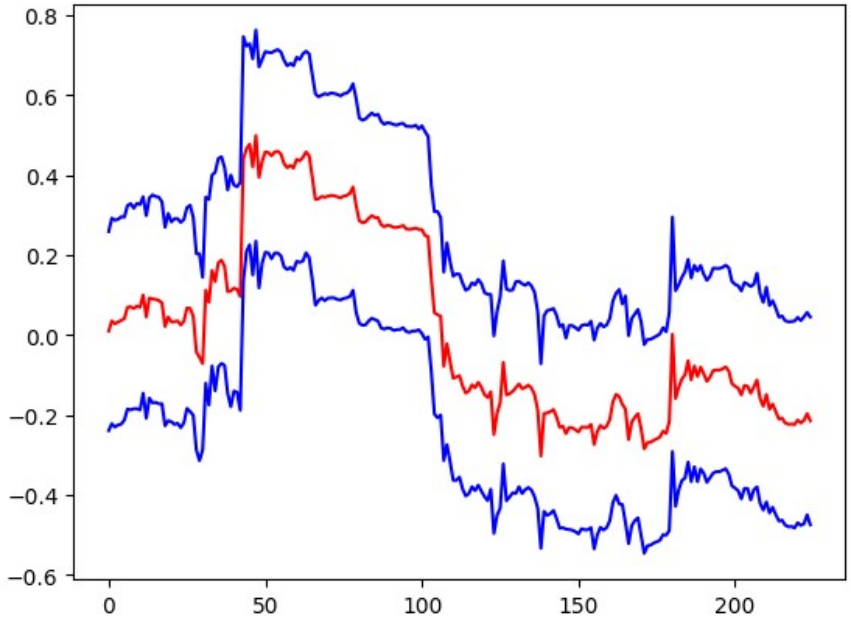


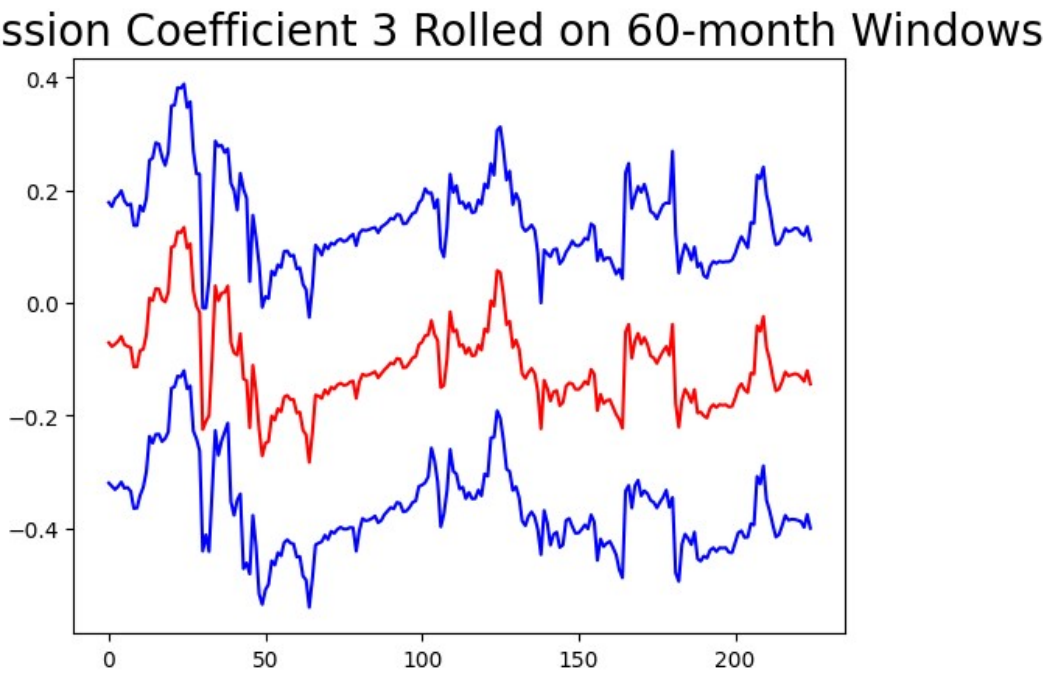

```
Jarque-Bera Test for Normality at 95% Significance:
JB-Statistic = 36.36016849740259; Lower Crit Value = 0.050635615968579795; Upper Cr
it Value = 7.377758908227871

----Part m----
Condition Number = inf

----Part n----
With Ramsey RESET test, p-value = 0.9804297181086428

----Part o----
Running rolling regressions...
```

Regression Coefficient 1 Rolled on 60-month Windows



Regression Coefficient 2 Rolled on 60-month Windows





Part (d): Observations/Remarks

For log prices p_t, $AR(1)$ seems to be the best model. Both R^2 and adjusted-R^2 jump from approximately zero to around 0.99 when we increase the lag from 0 to 1; hence, $AR(1)$ is a better choice than $AR(0)$. Further, even though R^2 and adjusted-R^2 for $AR(1)$ and $AR(2)$ are close in values, BIC reveals that $AR(1)$ is slightly better (more negative value of BIC).

For log returns r_t, we observe similar pattern. The R^2 for $AR(0)$ is very low. Between $AR(1)$ and $AR(2)$, $AR(1)$ has a more negative BIC, hence is somewhat better. However, $AR(1)$ and $AR(2)$ both suffer from multicollinearity, as indicated by their large condition numbers.

Part (e): Observations/Remarks

For log prices p_t, the R^2 and adjusted-R^2 metric increase from 0 to around 0.99 by increasing lag order from 0 to 1. This may be attributed to the underlying growth trend in p_t being explained by considering p_{t-1}. For log returns r_t, however, both R^2 and adjusted-R^2 remain fairly small, indicating relatively bad fit.

The plot of squared residuals for all orders of AR for both p_t and r_t show a very uneven pattern (several valleys and peaks), lending strong evidence against homoskedasticity. The data is most likely heteroskedastic.

The information criteria metrics - AIC, BIC, and $HQIC$ - for both p_t and r_t indicate that $AR(1)$ is the most informative model (more negative values of information criteria).

The Durbin-Watson statistic is close to 2 (except in $AR(0)$ for p_t), indicating that the residuals ϵ_t are not autocorrelated. This makes sense since we use independent random samples from normal distribution to generate the white noise ϵ_t. The Breusch-Godfrey test corroborates the same.

The QQ-plot for p_t $AR(0)$ shows significant departure from normal distribution; that is, the residuals are not normally distributed. For all other cases, the QQ-plots indicate that the residuals are normally distributed for the most part, except for the outliers.

The condition number rises sharply once we increase the order from 0, signalling severe problem of multicollinearity.

The large p-values in the Ramsey test indicate that the linear model specification is indeed the right choice for the model (null hypothesis is not rejected).

The regression coefficients evaluated over 60-month rolling windows fluctuate a lot, signifying that one model may not be used to explain observations for the entire time period studied.