MidTermAssignmentwith8Facts

November 14, 2024

1 Python assignment

[86]:
```
# ---------------------------------------------------------
# File Name: MidTermAssignment.py
# Description:
# Autor: Prat Paul; Gavini Charles; Fournier Justin; Blanc Mathieu
# Creation Date: 2024-10-16
# Version: 1.0
# ---------------------------------------------------------
```

Installing yahoofinance

[87]:
```
#pip install yfinance
```

Installing statsmodels

[88]:
```
#pip install statsmodels
```

[89]:
```
#importations
import numpy as np
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from statsmodels.graphics.tsaplots import plot_acf # import this function from
    ↪this submodule
import statsmodels.api as sm
import scipy.stats as stats
from scipy.stats import gaussian_kde, norm, iqr, skew, kurtosis, jarque_bera,
    ↪kstest, anderson
from statsmodels.stats.diagnostic import lilliefors
import scipy.signal as ss
import pylab
```

2 *First pandas dataframe of Amazon stocks*

[90]:
```
# Importing Amazon stock from yahoo finance
Amazon = yf.download("AMZN", start="1999-01-21", end="2024-10-16")
Amazon.head()
```

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

[90]:

	Open	High	Low	Close	Adj Close	Volume
Date						
1999-01-21	2.612500	2.759375	2.314063	2.650000	2.650000	940964000
1999-01-22	2.487500	3.146875	2.468750	3.075000	3.075000	875316000
1999-01-25	3.037500	3.084375	2.750000	2.809375	2.809375	546476000
1999-01-26	2.815625	3.031250	2.765625	2.877344	2.877344	490696000
1999-01-27	3.353125	3.493750	3.000000	3.140625	3.140625	700452000

[91]:
```
#pip install perfplot
```

[92]:
```
latex_table = Amazon.head().to_latex(index=True)
with open("Latex/table.tex", "w") as file:
    file.write(latex_table)
```

```
/var/folders/5r/ft807c7n1ngd3fpt2_gwsg0m0000gn/T/ipykernel_78356/3304008134.py:1
: FutureWarning: In future versions `DataFrame.to_latex` is expected to utilise
the base implementation of `Styler.to_latex` for formatting and rendering. The
arguments signature may therefore change. It is recommended instead to use
`DataFrame.style.to_latex` which also contains additional functionality.
  latex_table = Amazon.head().to_latex(index=True)
```

3 *Cheking if timestamp is 25 years*

[93]:
```
print('Amazon data range is: ',Amazon.index[0],Amazon.index[-1])

#trying to find gaps

#First create a dataframe for a fullrange of our index, without any gap with␣
 ↪the following formula:
full_range = pd.date_range(start=Amazon.index.min(), end=Amazon.index.max(),␣
 ↪freq='B')

#Then compare to our dataframe:

MissingDays=full_range.difference(Amazon.index)

#Print the count and the detail preview:
print('Missing Days count is: ',len(MissingDays))
print("missing dates",MissingDays)
```

```
print("total size of the 25 years range",len(Amazon.index),"the ratio of␣
↪missing inputs/ total size of the data = ",100*len(MissingDays)/len(Amazon.
↪index))
#We can see that data have ponctual gaps, no issue here we can still use it
```

```
Amazon data range is:  1999-01-21 00:00:00 2024-10-15 00:00:00
Missing Days count is:  238
missing dates DatetimeIndex(['1999-02-15', '1999-04-02', '1999-05-31',
'1999-07-05',
               '1999-09-06', '1999-11-25', '1999-12-24', '2000-01-17',
               '2000-02-21', '2000-04-21',
               …
               '2023-11-23', '2023-12-25', '2024-01-01', '2024-01-15',
               '2024-02-19', '2024-03-29', '2024-05-27', '2024-06-19',
               '2024-07-04', '2024-09-02'],
              dtype='datetime64[ns]', length=238, freq=None)
total size of the 25 years range 6476 the ratio of missing inputs/ total size of
the data =  3.675108091414453
```

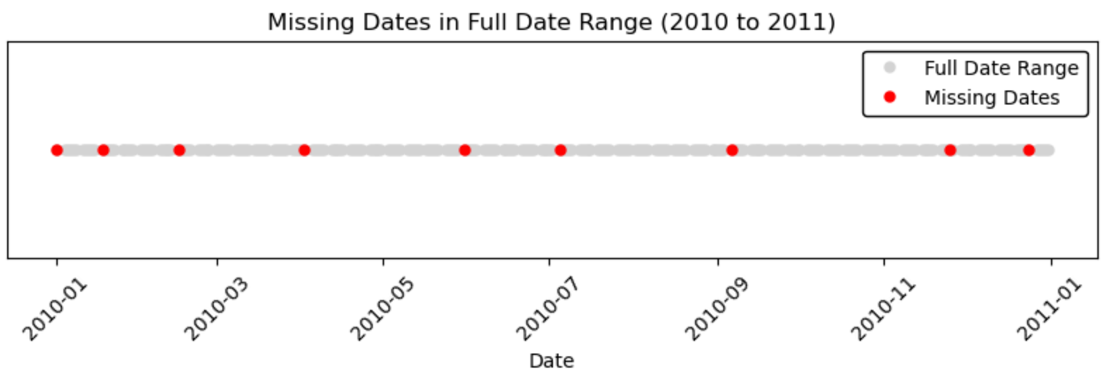
[94]:
```
Amazon.index
#extracting adjusted
Amzn_adj=Amazon['Adj Close']
Amzn_adj.index = Amazon.index

#display first 5 rows, now it is a pandas series instead of a dataframe
Amzn_adj.head()
```

[94]:
```
Date
1999-01-21    2.650000
1999-01-22    3.075000
1999-01-25    2.809375
1999-01-26    2.877344
1999-01-27    3.140625
Name: Adj Close, dtype: float64
```

For the possible gaps in data, we plot them here

[95]:
```
#plot the missing dates
full_data = Amazon.reindex(full_range)

#zoom in over one year
start_date = "2010-01-01"
end_date = "2011-01-01"
filtered_full_range = full_range[(full_range >= start_date) & (full_range <=␣
↪end_date)]
filtered_missing_dates = MissingDays[(MissingDays >= start_date) & (MissingDays␣
↪<= end_date)]
```

```python
plt.figure(figsize=(10, 2))

# Plot all dates in the filtered range with gray dots (showing the full␣
↪timeline for this period)
plt.plot(filtered_full_range, [1] * len(filtered_full_range), 'o',␣
↪color='lightgray', markersize=5, label="Full Date Range")

# Overlay red dots only on the missing dates within the filtered range
plt.plot(filtered_missing_dates, [1] * len(filtered_missing_dates), 'ro',␣
↪markersize=5, label="Missing Dates")

# Customize plot
plt.title("Missing Dates in Full Date Range (2010 to 2011)",color='black')
plt.xlabel("Date",color='black')
plt.yticks([])  # Hide y-axis labels for clarity
plt.xticks(rotation=45,color='black')
plt.legend(facecolor='white', edgecolor='black', framealpha=1, fontsize=10)

#Saving the plot in pdf format
plt.savefig('Latex/Img/MissingDates(2010_to_2011).pdf', format='pdf',␣
↪bbox_inches='tight')

plt.show()
```



4 *PRICES*

[96]:
```python
# extract the closing prices of the Amazon stok (as in lecture)
Pt_d_all = Amazon["Adj Close"]
Pt_d_all.name = 'Pt.d'
# mutate the Index into a DatetimeIndex
Pt_d_all.index = pd.to_datetime(Pt_d_all.index)
Pt_d_all.head()
```

[96]:

```
Date
1999-01-21    2.650000
1999-01-22    3.075000
1999-01-25    2.809375
1999-01-26    2.877344
1999-01-27    3.140625
Name: Pt.d, dtype: float64
```

Compute log price

[97]:

```python
pt_d_all = np.log(Pt_d_all)
pt_d_all.name = 'pt.d'
pt_d_all.head()
```

[97]:

```
Date
1999-01-21    0.974560
1999-01-22    1.123305
1999-01-25    1.032962
1999-01-26    1.056868
1999-01-27    1.144422
Name: pt.d, dtype: float64
```

Compute weekly monthly and yearly

[98]:

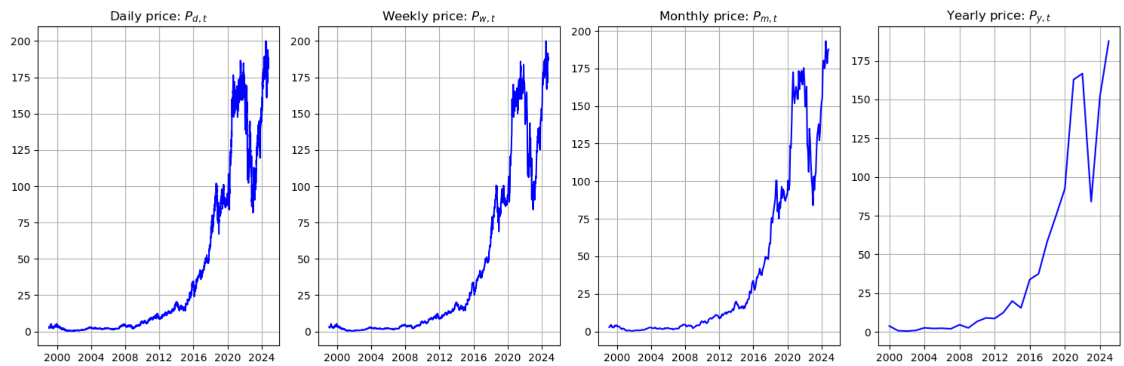
```python
pt_w_all = pt_d_all.resample('W').last()
pt_m_all = pt_d_all.resample('M').last()
pt_y_all = pt_d_all.resample('Y').last()
# and rename them:
pt_w_all.name = 'pt.w.all'
pt_m_all.name = 'pt.m.all'
pt_y_all.name = 'pt.y.all'

#idem for simply prices
Pt_w_all = Pt_d_all.resample('W').last()
Pt_m_all = Pt_d_all.resample('M').last()
Pt_y_all = Pt_d_all.resample('Y').last()
# and rename them:
Pt_w_all.name = 'Pt_w_all'
Pt_m_all.name = 'Pt_m_all'
Pt_y_all.name = 'Pt_y_all'
```

Plot the simple prices

[99]:

```python
# set the 1x4 windows layout
fig, axs = plt.subplots(1, 4, figsize=(15, 5))
# Daily Price
axs[0].plot(Pt_d_all.index, Pt_d_all, color='blue')
axs[0].set_title('Daily price: $P_{d,t}$')
axs[0].grid(True)
```

```
# Weekly price
axs[1].plot(Pt_w_all.index, Pt_w_all, color='blue')
axs[1].set_title('Weekly price: $P_{w,t}$')
axs[1].grid(True)
# Monthly price
axs[2].plot(Pt_m_all.index, Pt_m_all, color='blue')
axs[2].set_title('Monthly price: $P_{m,t}$')
axs[2].grid(True)
#Yearly price
axs[3].plot(Pt_y_all.index, Pt_y_all, color='blue')
axs[3].set_title('Yearly price: $P_{y,t}$')
axs[3].grid(True)

# Manage margings and plot
plt.tight_layout()
plt.savefig('Latex/Img/prices_time.pdf', format='pdf', bbox_inches='tight')
plt.show()
```



Adding python code to the latex document in the appendix part

[100]:
```
#Test for incorporating python code into the appendix section in the latex␣
↪document
code_content = r"""
\section{Appendix: Python Code}
Below is the Python code used in this analysis.

\begin{lstlisting}[language=Python, caption=Python Code for Analysis]
# Python code example
import numpy as np
import pandas as pd

def analyze_data(data):
    mean = np.mean(data)
    std_dev = np.std(data)
```

```
    return mean, std_dev

data = [1, 2, 3, 4, 5]
mean, std_dev = analyze_data(data)
print(f"Mean: {mean}, Standard Deviation: {std_dev}")
\end{lstlisting}
"""

# Write to the 'code_appendix.tex' file
with open("Latex/code_appendix.tex", "w") as file:
    file.write(code_content)
```

5 *Calculating returns*

[101]:
```
#calculating return

#log returns VS simple returns
Rt_d_all_temp = Pt_d_all.pct_change()
rt_d_all_temp = pt_d_all.diff()
rt_d_all_temp, Rt_d_all_temp
```

[101]:
```
(Date
 1999-01-21         NaN
 1999-01-22    0.148745
 1999-01-25   -0.090343
 1999-01-26    0.023906
 1999-01-27    0.087554
                 …
 2024-10-09    0.013319
 2024-10-10    0.007961
 2024-10-11    0.011559
 2024-10-14   -0.006802
 2024-10-15    0.000800
 Name: pt.d, Length: 6476, dtype: float64,
 Date
 1999-01-21         NaN
 1999-01-22    0.160377
 1999-01-25   -0.086382
 1999-01-26    0.024194
 1999-01-27    0.091501
                 …
 2024-10-09    0.013408
 2024-10-10    0.007993
 2024-10-11    0.011626
 2024-10-14   -0.006779
 2024-10-15    0.000800
```

```
Name: Pt.d, Length: 6476, dtype: float64)
```

Compute daily, weekly, and monthly

[102]:
```
rt_d_all = pt_d_all.diff().dropna() #dropna remove the first NaN
rt_w_all = pt_w_all.diff().dropna()
rt_m_all = pt_m_all.diff().dropna()
rt_y_all = pt_y_all.diff().dropna()

Rt_d_all = Pt_d_all.pct_change().dropna() #dropna remove the first NaN
Rt_w_all = Pt_w_all.pct_change().dropna()
Rt_m_all = Pt_m_all.pct_change().dropna()
Rt_y_all = Pt_y_all.pct_change().dropna()

# and rename them:
rt_d_all.name = 'rt_d_all'
rt_w_all.name = 'rt_w_all'
rt_m_all.name = 'rt_m_all'
rt_y_all.name = 'rt_y_all'

Rt_d_all.name = 'Rt_d_all'
Rt_w_all.name = 'Rt_w_all'
Rt_m_all.name = 'Rt_m_all'
Rt_y_all.name = 'Rt_y_all'

rt_d_all.head()
Rt_d_all.head()
```

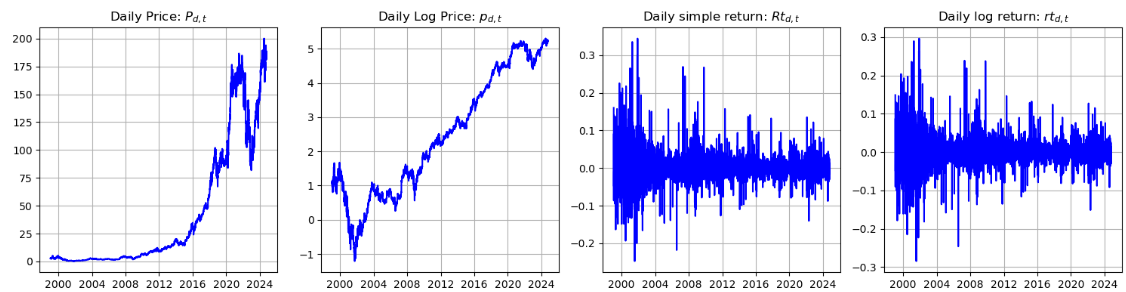
[102]:
```
Date
1999-01-22    0.160377
1999-01-25   -0.086382
1999-01-26    0.024194
1999-01-27    0.091501
1999-01-28   -0.021891
Name: Rt_d_all, dtype: float64
```

The first returns are correctly computed, we have to be careful to the dropna

Let's plot returns

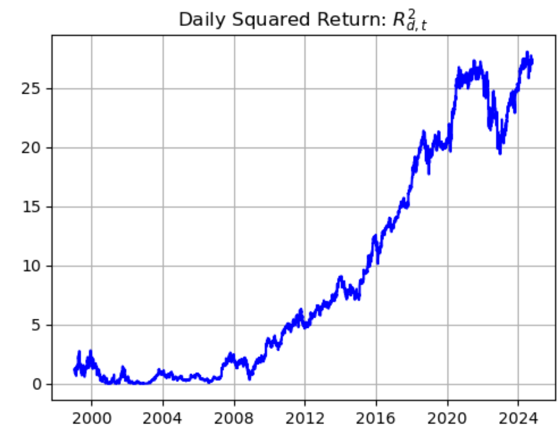
[103]:
```
# set the 1x3 windows layout
fig, axs = plt.subplots(1, 4, figsize=(15, 4))
# Daily Price
axs[0].plot(Pt_d_all.index, Pt_d_all, color='blue')
axs[0].set_title('Daily Price: $P_{d,t}$')
axs[0].grid(True)
# Daily log price
axs[1].plot(pt_d_all.index, pt_d_all, color='blue')
axs[1].set_title('Daily Log Price: $p_{d,t}$')
```

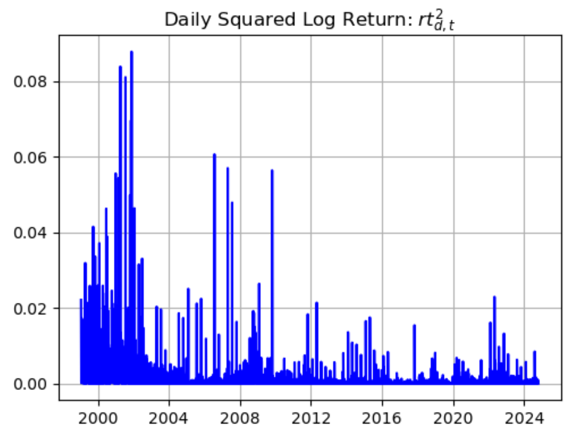
```
axs[1].grid(True)
# Daily simple returns
axs[2].plot(Rt_d_all.index, Rt_d_all, color='blue')
axs[2].set_title('Daily simple return: $Rt_{d,t}$')
axs[2].grid(True)
# Daily log returns
axs[3].plot(rt_d_all.index, rt_d_all, color='blue')
axs[3].set_title('Daily log return: $rt_{d,t}$')
axs[3].grid(True)


plt.tight_layout()
plt.savefig('Latex/Img/log_returns.pdf', format='pdf', bbox_inches='tight')
plt.show()
```



Squared returns

[104]:
```
fig, axs = plt.subplots(1, 2, figsize=(10, 4))

# Daily squared log price
axs[0].plot(pt_d_all.index, pt_d_all**2, color='blue')
axs[0].set_title('Daily Squared Return: $R_{d,t}^2$')
axs[0].grid(True)
# Daily squared log returns
axs[1].plot(rt_d_all.index, rt_d_all**2, color='blue')
axs[1].set_title('Daily Squared Log Return: $rt_{d,t}^2$')
axs[1].grid(True)

plt.tight_layout()
plt.savefig('Latex/Img/squared_log_returns.pdf', format='pdf',␣
  ↪bbox_inches='tight')
plt.show()
```



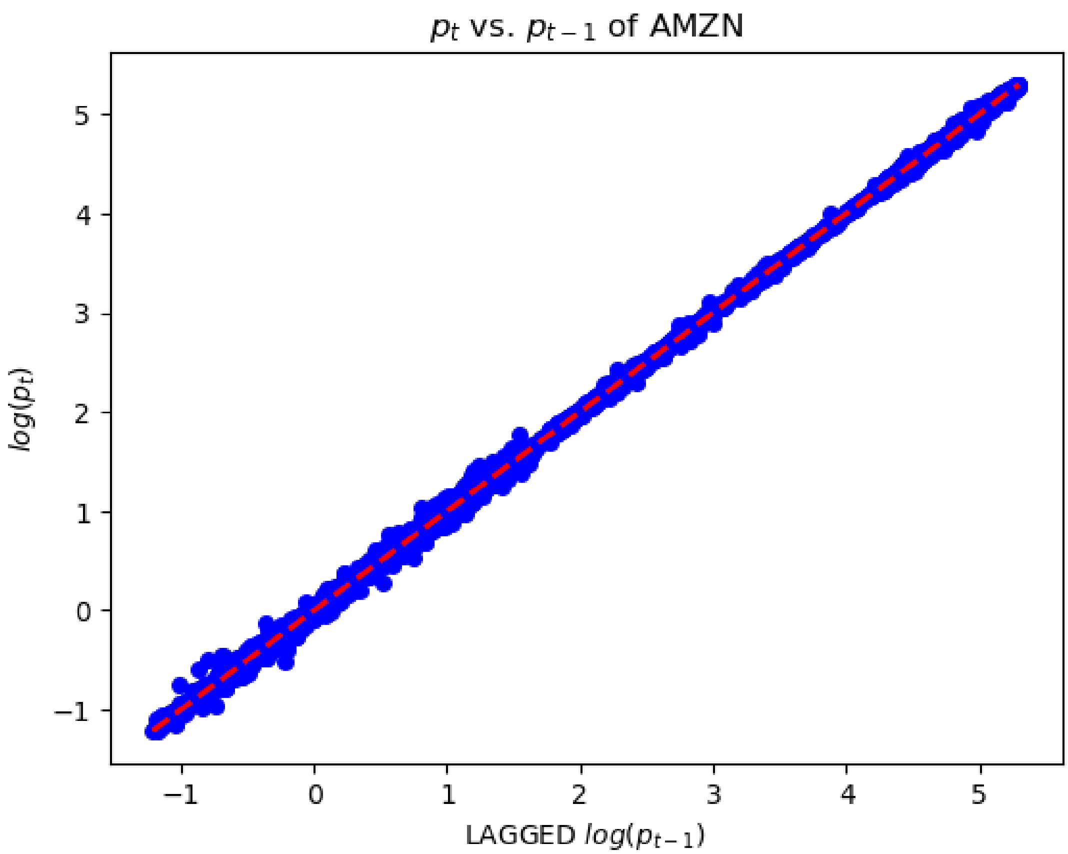


6 *Scatterplot of* $p_t - p_{t-1}$

First define the function for plotting a scatterplot

[105]:
```
def lag1_scatterplot(data, x_label, y_label, title):
    plt.scatter(data.shift(), data, color='blue', s=30) #data.shift is just␣
 ↪pt-1 (if t=0 what happend ?)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.title(title)
    plt.plot([min(data), max(data)], [min(data), max(data)],␣
 ↪linestyle='dashed', linewidth=2, color='red')
#    plt.savefig('Latex/Img/Laggedlog(p_t-1).pdf', format='pdf',␣
 ↪bbox_inches='tight')
```

[106]:
```
lag1_scatterplot(pt_d_all,"LAGGED $log(p_{t-1})$","$log(p_t)$","$p_t$ vs.␣
 ↪$p_{t-1}$ of AMZN")
```



7 Autocorrelation

[107]:
```
"""autocorrelate=pt_d_all.shift().corrwith(pt_d_all, method='pearson')
print(round(autocorrelate,4))"""
```

[107]:
```
"autocorrelate=pt_d_all.shift().corrwith(pt_d_all,
method='pearson')\nprint(round(autocorrelate,4))"
```

[108]:
```
"""
autocorrelate = pt_d_all.shift(1).corrwith(pt_d_all, method='pearson')
print(autocorrelate.round(4))
"""
```

[108]:
```
"\nautocorrelate = pt_d_all.shift(1).corrwith(pt_d_all,
method='pearson')\nprint(autocorrelate.round(4))\n"
```

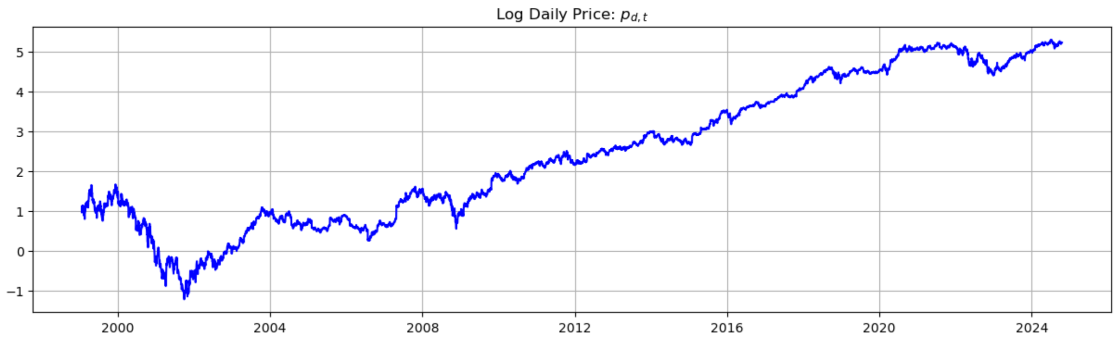
8 *4.1/ Prices are non-stationary*

1. Profile of Log Prices with Time

2. Pt VS P(t-1)

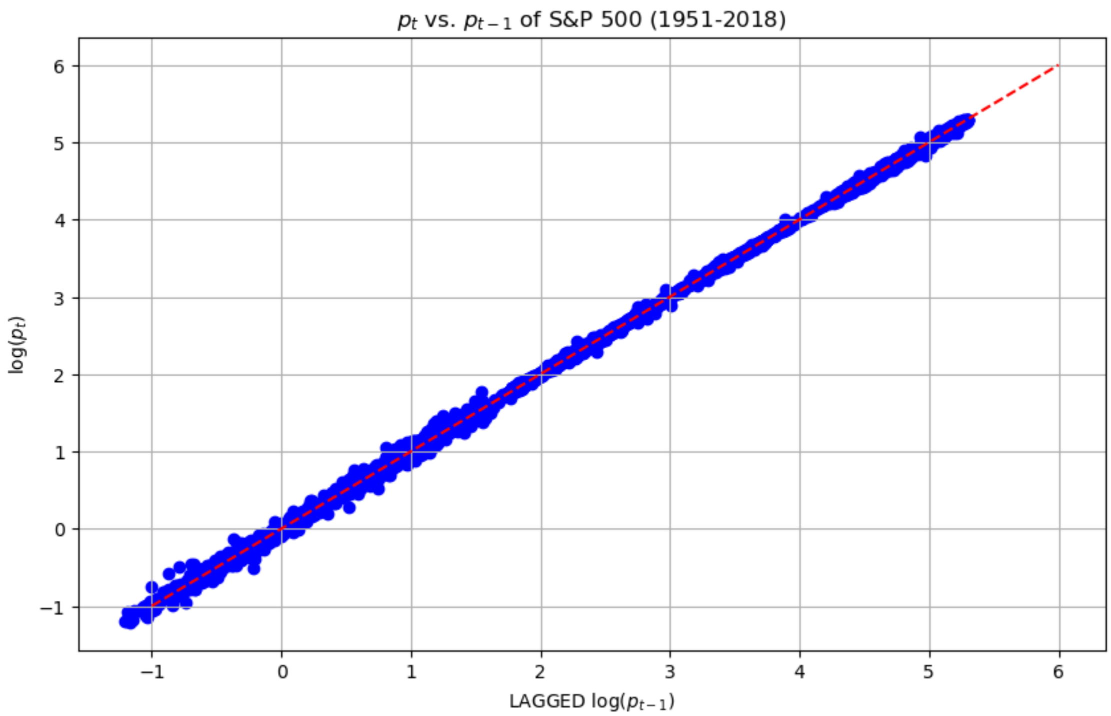
3. Autocorrelation of Daily Prices

[109]:
```python
import matplotlib.pyplot as plt

# Set the layout for 1x3 subplots (though you may not need all subplots)
fig, axs = plt.subplots(1, 1, figsize=(15, 4))

# Plot Daily Price
axs.plot(pt_d_all.index, pt_d_all, color='blue')
axs.set_title('Log Daily Price: $p_{d,t}$')
axs.grid(True)

# Show the plot
plt.show()
```



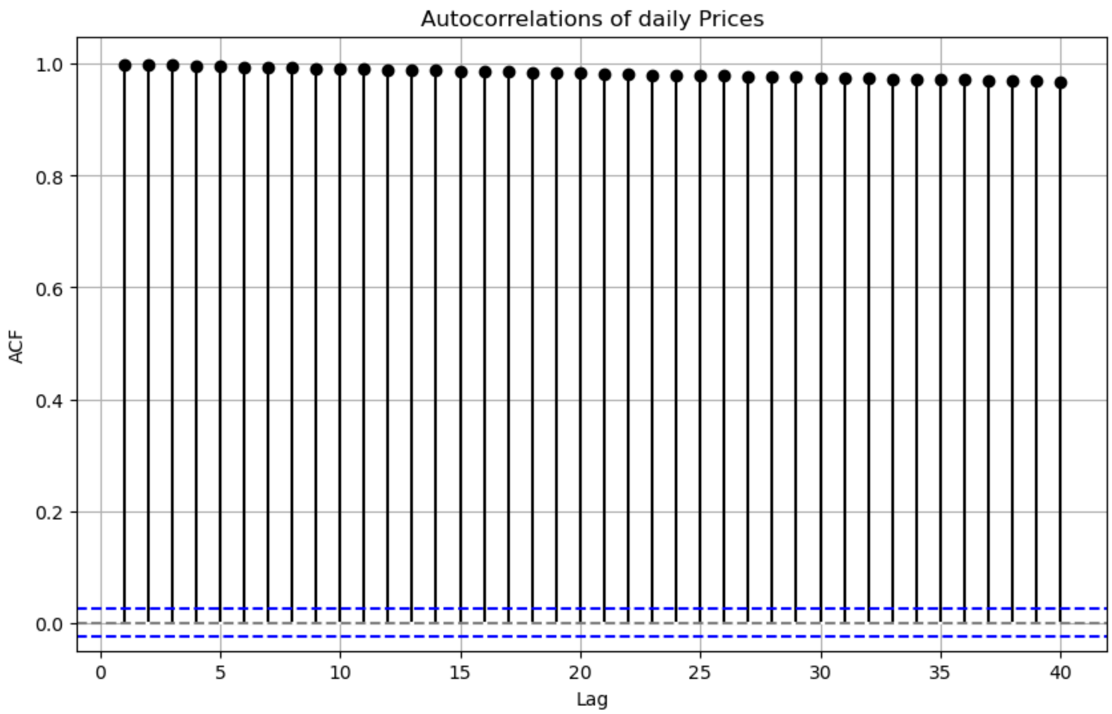
[110]:
```python
import yfinance as yf
import matplotlib.pyplot as plt
import numpy as np

# Estrai i log-prezzi giornalieri
log_price_daily = pt_d_all

# Calcola il log-prezzo al giorno precedente
log_price_previous = log_price_daily.shift(1)

# Creazione dello scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(log_price_previous, log_price_daily, color='blue')
plt.plot([-1, 6], [-1, 6], color='red', linestyle='--')
plt.title('$p_t$ vs. $p_{t-1}$ of S&P 500 (1951-2018)')
plt.xlabel(r'LAGGED $\log(p_{t-1})$')
plt.ylabel(r'$\log(p_t)$')
plt.grid(True)
```

```
# Salva il grafico in formato png
plt.savefig('Latex/Img/log(pt) vs log(pt-1).pdf', format='pdf',␣
↪bbox_inches='tight')

plt.show()
```




```
[111]: from statsmodels.tsa.stattools import acf

# Calculate empirical autocorrelation
lags = 40
acf_values = acf(Pt_d_all, nlags=lags)

# Calculate Bartlett intervals
Bart_Int = 1.96 / np.sqrt(len(Pt_d_all))

# Create the autocorrelation plot with Bartlett intervals
plt.figure(figsize=(10, 6))
plt.stem(np.arange(1, lags + 1), acf_values[1:], linefmt='k-', markerfmt='ko',␣
↪basefmt='w-')
plt.axhline(y=0, color='gray', linestyle='--')
plt.axhline(y=Bart_Int, color='blue', linestyle='--')
plt.axhline(y=-Bart_Int, color='blue', linestyle='--')
```

```
plt.title('Autocorrelations of daily Prices')
plt.xlabel('Lag')
plt.ylabel('ACF')
plt.grid(True)
#plt.savefig('Latex/Autocorrel_daily.pdf', format='pdf', bbox_inches='tight')
plt.show()
```

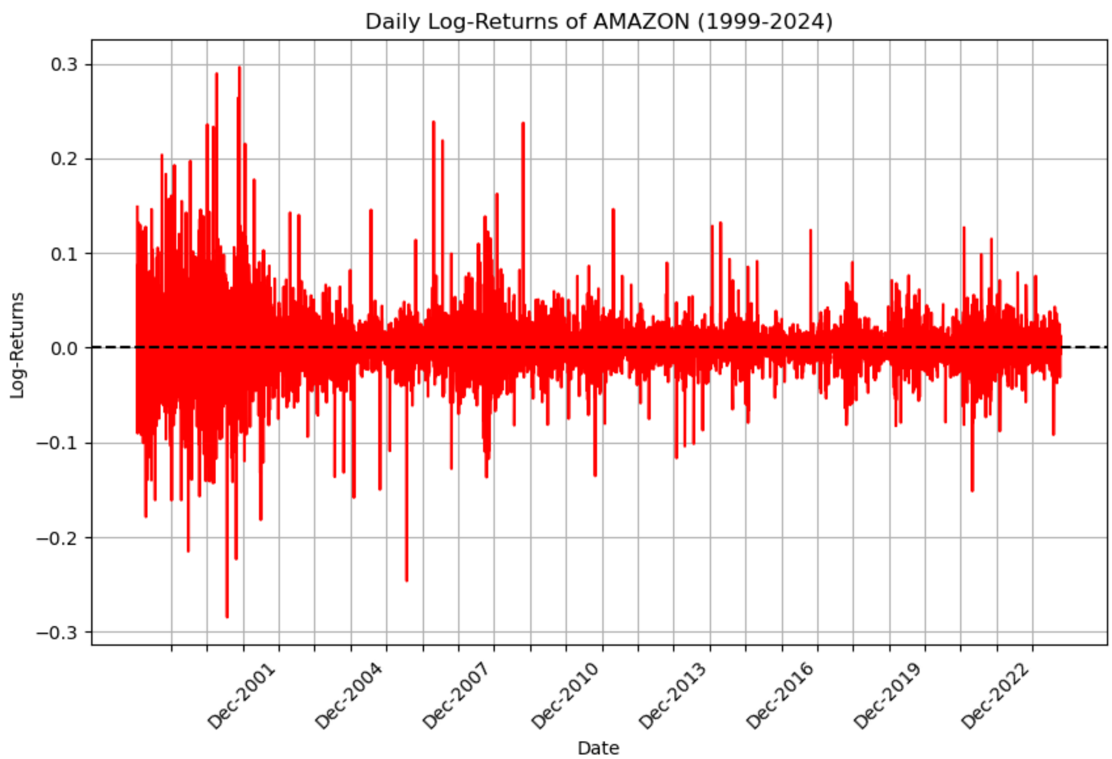


9 *4.2/ Log Returns are Stationary*

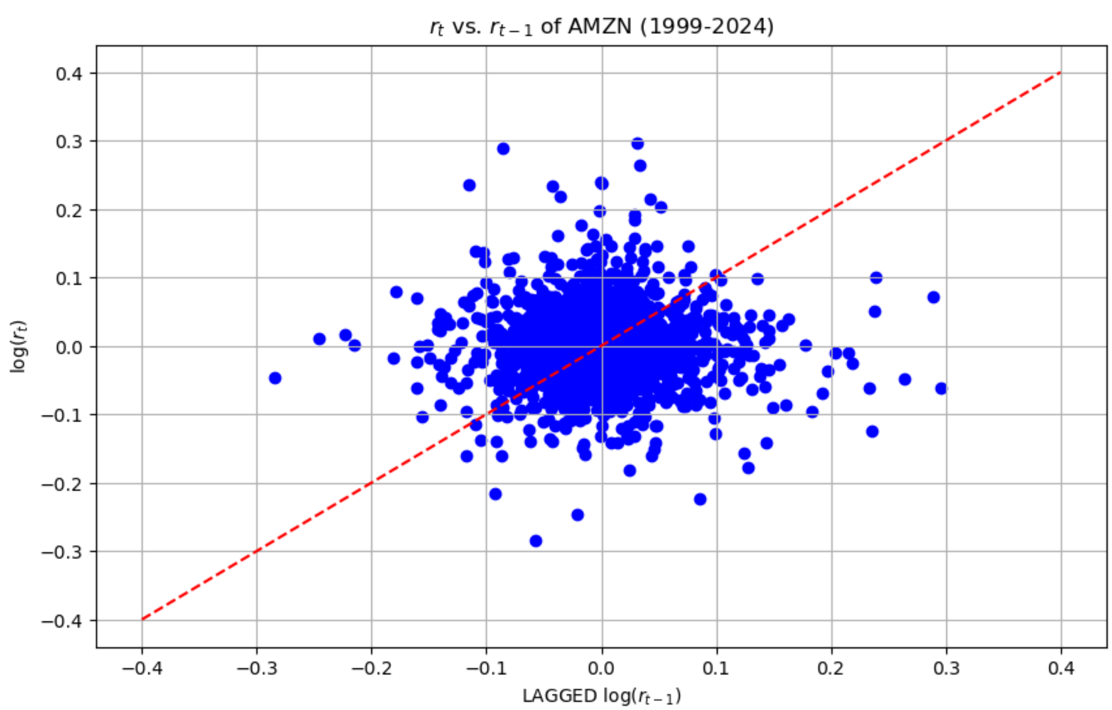
1. Profile of Log Returns with Time
2. Rt VS R(t-1)
3. Autocorrelation of Daily Returns

[112]:
```
import yfinance as yf
import matplotlib.pyplot as plt
import numpy as np

# Calculate daily log returns
log_returns_daily = rt_d_all

# Create the plot of daily log returns with a black horizontal line
plt.figure(figsize=(10, 6))
```

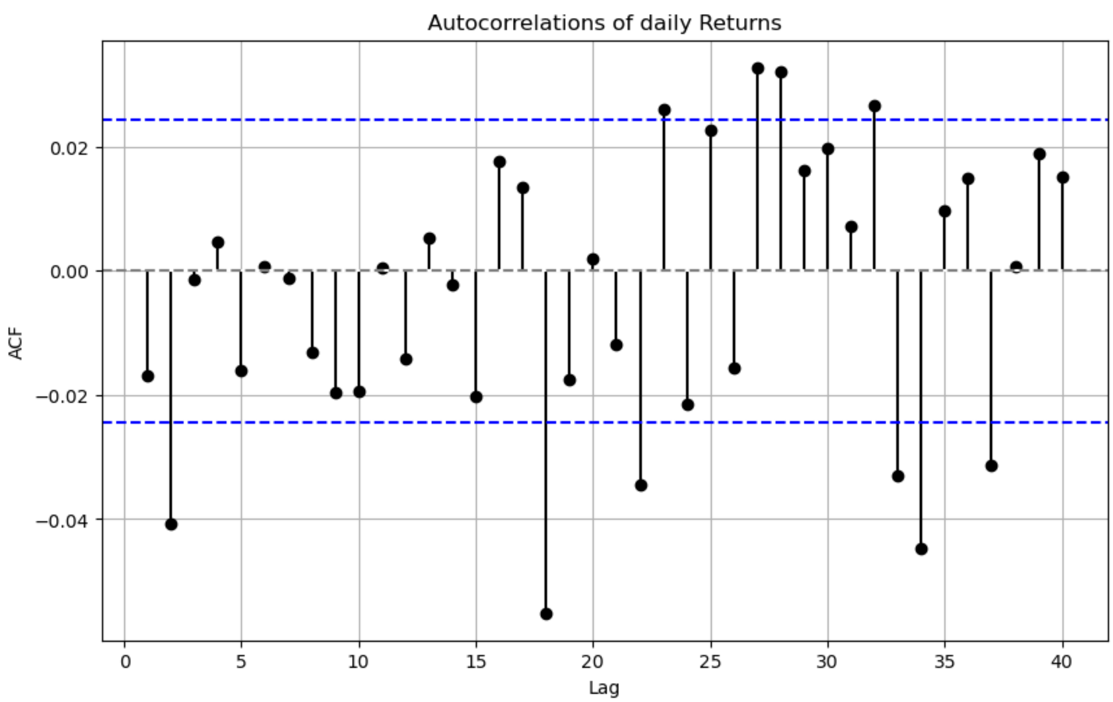
```
plt.plot(log_returns_daily.index, log_returns_daily, color='red')
plt.axhline(y=0, color='black', linestyle='--')
plt.title('Daily Log-Returns of AMAZON (1999-2024)')
plt.xlabel('Date')
plt.ylabel('Log-Returns')
plt.grid(True)

# Customizing x-axis labels for December 31 of each year
date_labels = pd.date_range(start='1999-12-31', end='2023-12-31', freq='A-DEC')
# Show 1 tick every 3 years
formatted_labels = [f'Dec-{date.year}' if date.year % 3 == 0 else '' for date␣
↪in date_labels]
# Add labels and rotate them
plt.xticks(date_labels, formatted_labels, rotation=45)

# Save the plot in png format
plt.savefig('Latex/Img/Daily Log Returns.pdf', format='pdf',␣
↪bbox_inches='tight')

plt.show()
```



[113]:
```python
import yfinance as yf
import matplotlib.pyplot as plt
import numpy as np


# Get the Daily Log Returns
log_return_daily = rt_d_all

# Calculation of the Lagged log returns
log_return_previous = log_return_daily.shift(1)

# Creation of the Scatter Plot
plt.figure(figsize=(10, 6))
plt.scatter(log_return_previous, log_return_daily, color='blue')
plt.plot([-0.4, 0.4], [-0.4, 0.4], color='red', linestyle='--')
plt.title('$r_t$ vs. $r_{t-1}$ of AMZN (1999-2024)')
plt.xlabel(r'LAGGED $\log(r_{t-1})$')
plt.ylabel(r'$\log(r_t)$')
plt.grid(True)

# Saving the Image
plt.savefig('Latex/Img/LogReturns_vs_LaggedLogReturns.pdf', format='png',␣
 ↪bbox_inches='tight')

plt.show()
```



[114]:
```python
from statsmodels.tsa.stattools import acf

# Calculate empirical autocorrelation
lags = 40
acf_values = acf(Rt_d_all, nlags=lags)

# Calculate Bartlett intervals
Bart_Int = 1.96 / np.sqrt(len(Rt_d_all))

# Create the autocorrelation plot with Bartlett intervals
plt.figure(figsize=(10, 6))
plt.stem(np.arange(1, lags + 1), acf_values[1:], linefmt='k-', markerfmt='ko',
    basefmt='w-')
plt.axhline(y=0, color='gray', linestyle='--')
plt.axhline(y=Bart_Int, color='blue', linestyle='--')
plt.axhline(y=-Bart_Int, color='blue', linestyle='--')
plt.title('Autocorrelations of daily Returns')
plt.xlabel('Lag')
plt.ylabel('ACF')
plt.grid(True)
#plt.savefig('Latex/Autocorrel_Returns_daily.pdf', format='pdf',
    bbox_inches='tight')
plt.show()
```

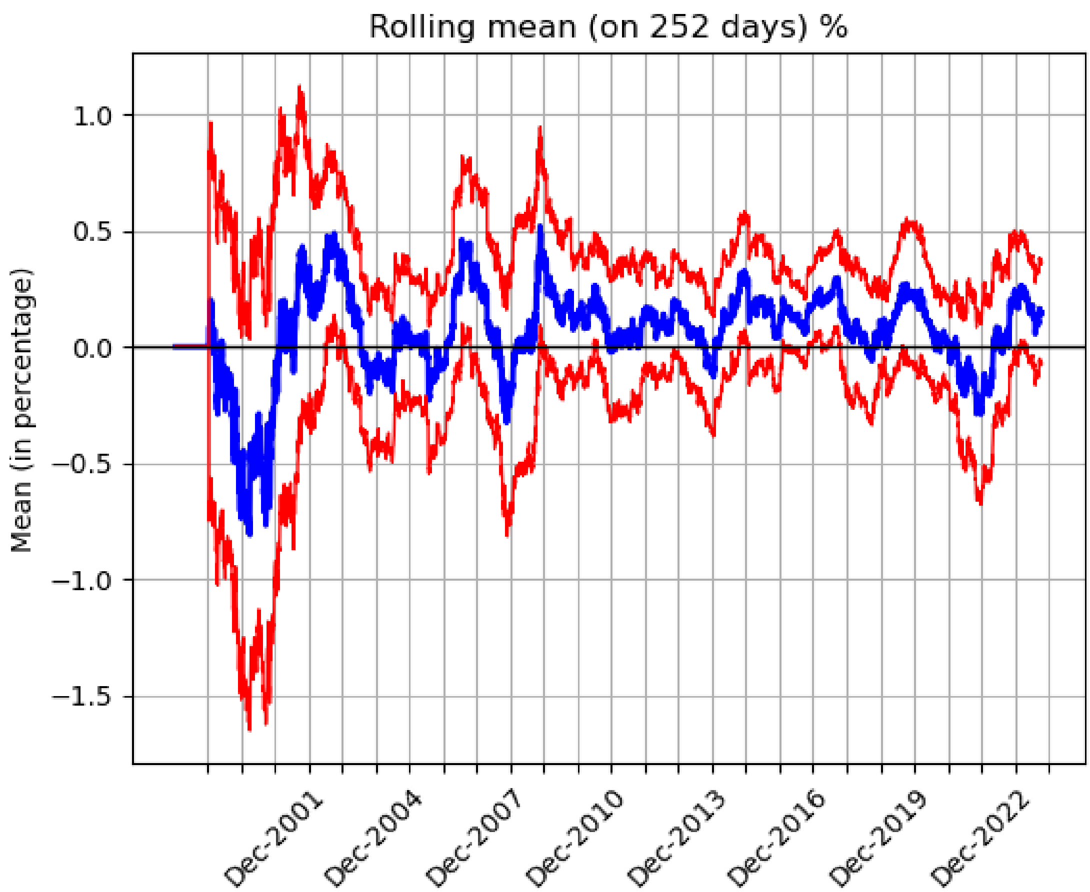


10 *4.3/ Are Log Returns Asymmetric ?*

1. Rolling Mean
2. Rolling Standard Deviation
3. Rolling Skewness
4. Current Skewness and Interpretation

[161]:
```python
import yfinance as yf
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import skew, kurtosis
import pandas as pd

# Compute daily log-returns
log_returns_daily = rt_d_all

# set the rolling window equal to 252 days
window_length = 252
T = log_returns_daily.shape[0]

# Create an empty matrix to store data
roll_mom_manual = np.zeros((T, 5))

# Run a for loop to fill the matrix with moments
for i in range(window_length, T):
    est_window = np.arange(i - window_length + 1, i + 1)

    # Use .iloc to select rows by integer positions, not labels
    y = log_returns_daily.iloc[est_window]

    # Compute the moments for each
    roll_mom_manual[i, 0] = np.mean(y)
    roll_mom_manual[i, 1] = np.std(y, ddof=1)
    roll_mom_manual[i, 2] = skew(y)
    roll_mom_manual[i, 3] = kurtosis(y)
    roll_mom_manual[i, 4] = np.mean((y - np.mean(y))**4)


# Plot results of manually computed rolling mean
mean_plot_man = roll_mom_manual[:, 0]
mean_plot_man_ub = mean_plot_man + 1.96 * roll_mom_manual[:, 1] / np.
 ↪sqrt(window_length)
mean_plot_man_lb = mean_plot_man - 1.96 * roll_mom_manual[:, 1] / np.
 ↪sqrt(window_length)
```

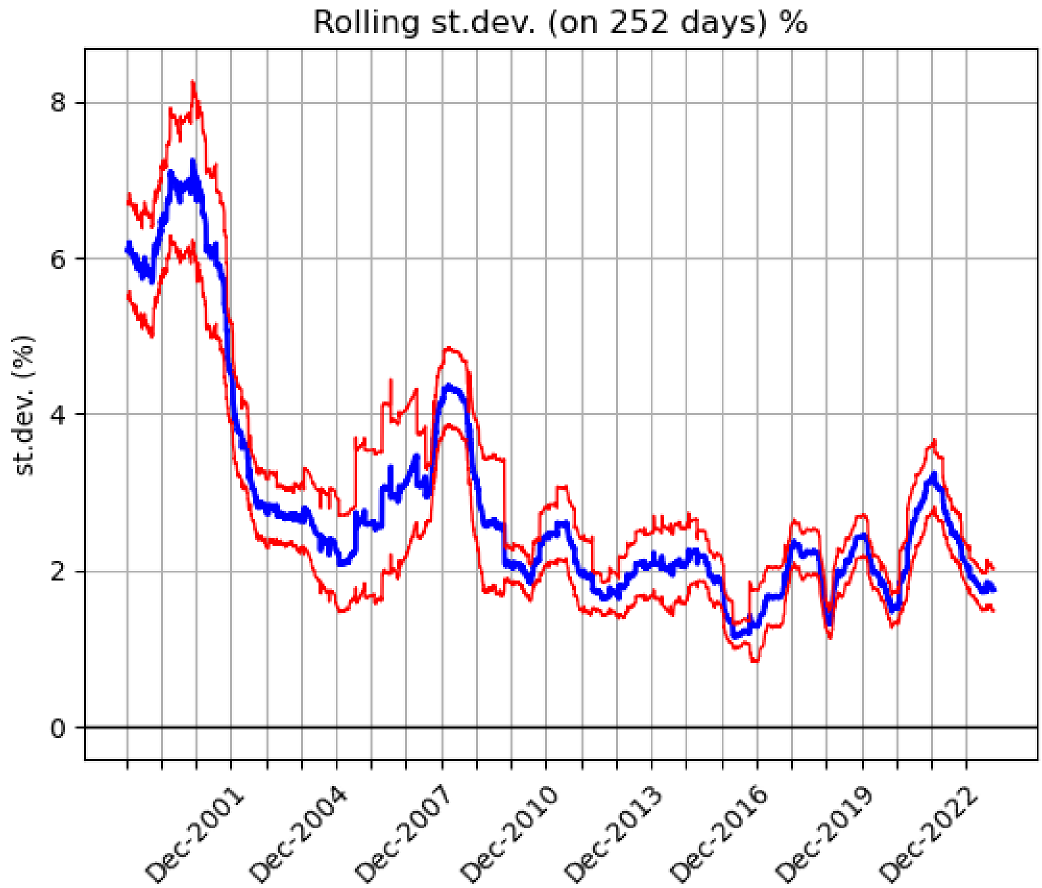
```
data2plot_na = np.column_stack((mean_plot_man, mean_plot_man_lb,␣
 ↪mean_plot_man_ub))

data_index = log_returns_daily.index

data2plot_na = pd.DataFrame({'Mean': mean_plot_man, 'LowerBound':␣
 ↪mean_plot_man_lb, 'UpperBound': mean_plot_man_ub},
                               index=data_index)

# Select only rows without missing values
data2plot = data2plot_na.dropna()
# retrieve the data index
data2plot

# Customizing x-axis labels for December 31 of each year
date_labels = pd.date_range(start='1999-12-31', end='2024-12-31', freq='A-DEC')
# Show 1 tick every 3 years
formatted_labels = [f'Dec-{date.year}' if date.year % 3 == 0 else '' for date␣
 ↪in date_labels]
# Add labels and rotate them
plt.xticks(date_labels, formatted_labels, rotation=45)


# Plot the data
plt.plot(data2plot.index, data2plot["Mean"] * 100, color='blue', linestyle='-',␣
 ↪linewidth=2)
plt.plot(data2plot.index, data2plot["LowerBound"] * 100, color='red',␣
 ↪linestyle='-', linewidth=1)
plt.plot(data2plot.index, data2plot["UpperBound"] * 100, color='red',␣
 ↪linestyle='-', linewidth=1)
plt.grid(True)
plt.xlabel('')
plt.ylabel('Mean (in percentage)')
plt.title('Rolling mean (on 252 days) %')
plt.axhline(0, linestyle='-', color='black', linewidth=1)  # Add a zero line


plt.savefig('Latex/Img/AMZN_MEAN_rolling_1999_2024.pdf', format='pdf',␣
 ↪bbox_inches='tight')

plt.show()
```



[162]:
```
# extract the Std Dev from roll_mom_manual
sd_plot = roll_mom_manual[:,1]
mu4 = roll_mom_manual[:,4]
sd_plot_ub = roll_mom_manual[:,1]+1.96*(1/(2*sd_plot)*np.sqrt(mu4-sd_plot**4))/
↪np.sqrt(window_length)
sd_plot_lb = roll_mom_manual[:,1]-1.96*(1/(2*sd_plot)*np.sqrt(mu4-sd_plot**4))/
↪np.sqrt(window_length)

data2plot_na = np.column_stack((sd_plot, sd_plot_lb, sd_plot_ub))

data_index = log_returns_daily.index

data2plot_na = pd.DataFrame({'StD': sd_plot, 'LowerBound': sd_plot_lb,␣
↪'UpperBound': sd_plot_ub},
                            index=data_index)

# Select only rows without missing values
data2plot = data2plot_na.dropna()
```

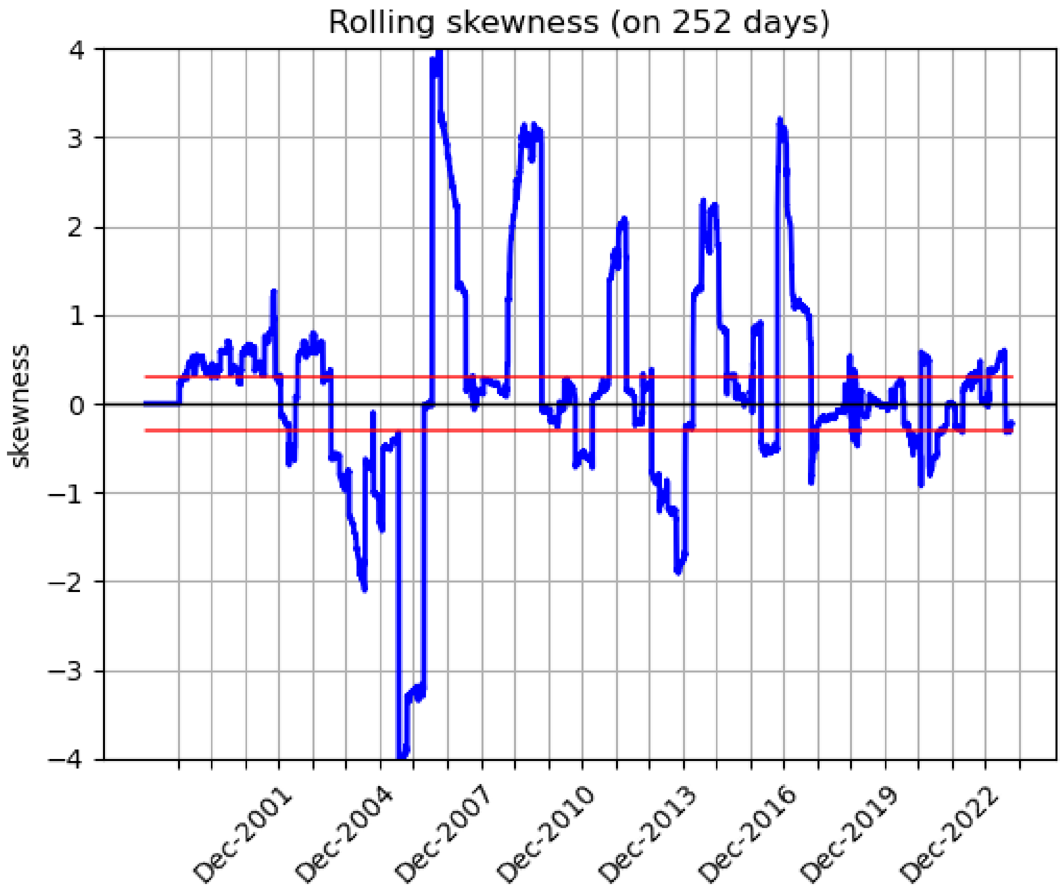
```python
# retrieve the data index
data2plot

# Customizing x-axis labels for December 31 of each year
date_labels = pd.date_range(start='1999-01-01', end='2024-10-01', freq='A-DEC')
# Show 1 tick every 3 years
formatted_labels = [f'Dec-{date.year}' if date.year % 3 == 0 else '' for date␣
↪in date_labels]
# Add labels and rotate them
plt.xticks(date_labels, formatted_labels, rotation=45)

# Plot the data
plt.plot(data2plot.index, data2plot["StD"] * 100, color='blue', linestyle='-',␣
↪linewidth=2)
plt.plot(data2plot.index, data2plot["LowerBound"] * 100, color='red',␣
↪linestyle='-', linewidth=1)
plt.plot(data2plot.index, data2plot["UpperBound"] * 100, color='red',␣
↪linestyle='-', linewidth=1)
plt.xlabel('')
plt.grid(True)
plt.ylabel('st.dev. (%)')
plt.title('Rolling st.dev. (on 252 days) %')
plt.axhline(0, linestyle='-', color='black', linewidth=1)  # Add a zero line


plt.savefig('Latex/Img/Fact7_AMZN_rolling_stdev.pdf', format='pdf',␣
↪bbox_inches='tight')
plt.show()
```

```
/var/folders/5r/ft807c7n1ngd3fpt2_gwsg0m0000gn/T/ipykernel_78356/1880708915.py:4
: RuntimeWarning: divide by zero encountered in true_divide
  sd_plot_ub = roll_mom_manual[:,1]+1.96*(1/(2*sd_plot)*np.sqrt(mu4-
sd_plot**4))/np.sqrt(window_length)
/var/folders/5r/ft807c7n1ngd3fpt2_gwsg0m0000gn/T/ipykernel_78356/1880708915.py:4
: RuntimeWarning: invalid value encountered in multiply
  sd_plot_ub = roll_mom_manual[:,1]+1.96*(1/(2*sd_plot)*np.sqrt(mu4-
sd_plot**4))/np.sqrt(window_length)
/var/folders/5r/ft807c7n1ngd3fpt2_gwsg0m0000gn/T/ipykernel_78356/1880708915.py:5
: RuntimeWarning: divide by zero encountered in true_divide
  sd_plot_lb = roll_mom_manual[:,1]-1.96*(1/(2*sd_plot)*np.sqrt(mu4-
sd_plot**4))/np.sqrt(window_length)
/var/folders/5r/ft807c7n1ngd3fpt2_gwsg0m0000gn/T/ipykernel_78356/1880708915.py:5
: RuntimeWarning: invalid value encountered in multiply
  sd_plot_lb = roll_mom_manual[:,1]-1.96*(1/(2*sd_plot)*np.sqrt(mu4-
sd_plot**4))/np.sqrt(window_length)
```



[116]:
```
# Skewess
skew_plot = roll_mom_manual[:,2]
skew_plot_ub = np.full(skew_plot.shape[0],+1.96*np.sqrt(6)/np.
 ↪sqrt(window_length))
skew_plot_lb = np.full(skew_plot.shape[0],-1.96*np.sqrt(6)/np.
 ↪sqrt(window_length))

data2plot_na = np.column_stack((skew_plot, skew_plot_lb, skew_plot_ub))

data_index = log_returns_daily.index

data2plot_na = pd.DataFrame({'Skewness': skew_plot, 'LowerBound': skew_plot_lb,␣
 ↪'UpperBound': skew_plot_ub},
                               index=data_index)

# Select only rows without missing values
data2plot = data2plot_na.dropna()
# retrieve the data index
```

```
data2plot

# Customizing x-axis labels for December 31 of each year
date_labels = pd.date_range(start='1999-12-31', end='2024-12-31', freq='A-DEC')
# Show 1 tick every 3 years
formatted_labels = [f'Dec-{date.year}' if date.year % 3 == 0 else '' for date in date_labels]
# Add labels and rotate them
plt.xticks(date_labels, formatted_labels, rotation=45)

# Plot the data
plt.plot(data2plot.index, data2plot["Skewness"], color='blue', linestyle='-', linewidth=2)
plt.plot(data2plot.index, data2plot["LowerBound"], color='red', linestyle='-', linewidth=1)
plt.plot(data2plot.index, data2plot["UpperBound"], color='red', linestyle='-', linewidth=1)
plt.ylim(-4,4)
plt.grid(True)
plt.xlabel('')
plt.ylabel('skewness')
plt.title('Rolling skewness (on 252 days)')
plt.axhline(0, linestyle='-', color='black', linewidth=1)  # Add a zero line


plt.savefig('Latex/Img/AMZN_skew_rolling_1999_2024.pdf', format='pdf', bbox_inches='tight')

plt.show()
```



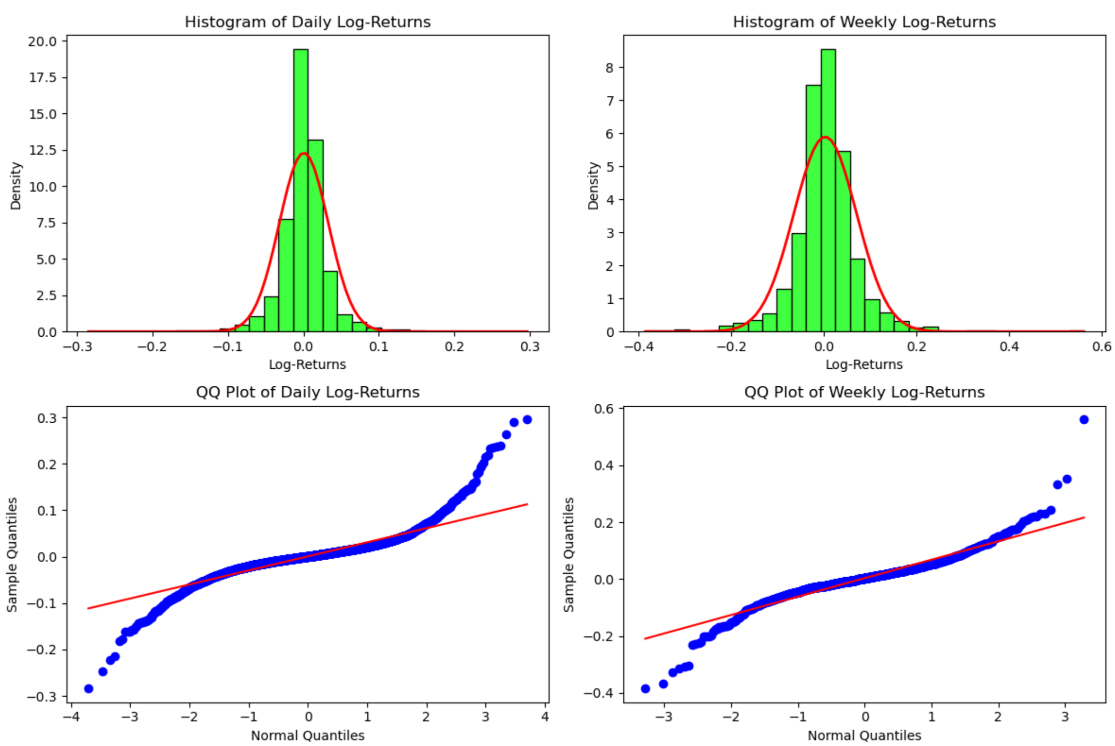
11 *4.4/ Heavy Tailed Distribution for the Daily Log Returns ?*

1. Comparison of the Normal Distribution vs our actual Values
2. Excess Kurtosis of our data
3. Interpretation

[117]:
```
import yfinance as yf
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats
import seaborn as sns

# Extract daily log-returns
log_price_daily = pt_d_all  # Ensure this is a pandas DataFrame or Series
log_returns_daily = rt_d_all  # Ensure this is a pandas DataFrame or Series
```

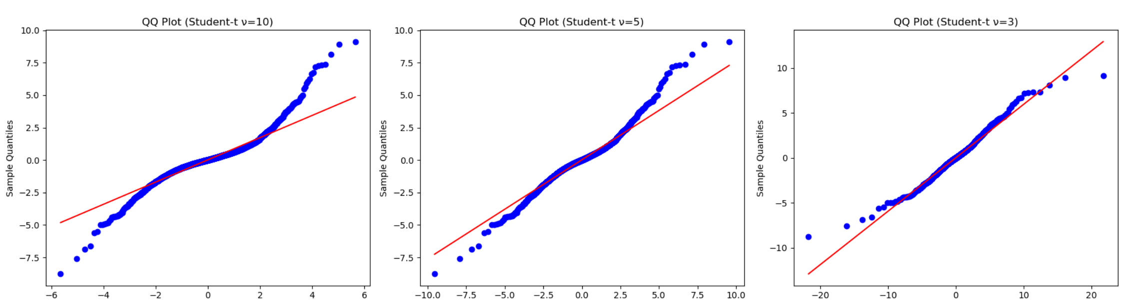
```
# If log_returns_daily is a DataFrame, convert it to a 1D array (assuming 'column_name' is the name of the column)
log_returns_daily = log_returns_daily.values.flatten()  # Ensure it's 1D array

# Calculate monthly log-returns
log_price_weekly = pt_w_all  # Ensure this is a pandas DataFrame or Series
log_returns_weekly = rt_w_all  # Ensure this is a pandas DataFrame or Series

# If log_returns_monthly is a DataFrame, convert it to a 1D array (assuming 'column_name' is the name of the column)
log_returns_weekly = log_returns_weekly.values.flatten()  # Ensure it's 1D array

# Create the figure with four subplots
fig, axs = plt.subplots(2, 2, figsize=(12, 8))

# Plot histogram of daily log-returns
sns.histplot(log_returns_daily, bins=30, color='lime', edgecolor='black', kde_kws={'color': 'red'}, ax=axs[0, 0], stat='density')
axs[0, 0].plot(np.linspace(log_returns_daily.min(), log_returns_daily.max(), 100),
               stats.norm.pdf(np.linspace(log_returns_daily.min(), log_returns_daily.max(), 100),
                              log_returns_daily.mean(), log_returns_daily.std()), color='red', linewidth=2)
axs[0, 0].set_title('Histogram of Daily Log-Returns')
axs[0, 0].set_xlabel('Log-Returns')
axs[0, 0].set_ylabel('Density')

# Plot histogram of monthly log-returns
sns.histplot(log_returns_weekly, bins=30, color='lime', edgecolor='black', kde_kws={'color': 'red'}, ax=axs[0, 1], stat='density')
axs[0, 1].plot(np.linspace(log_returns_weekly.min(), log_returns_weekly.max(), 100),
               stats.norm.pdf(np.linspace(log_returns_weekly.min(), log_returns_weekly.max(), 100),
                              log_returns_weekly.mean(), log_returns_weekly.std()), color='red', linewidth=2)
axs[0, 1].set_title('Histogram of Weekly Log-Returns')
axs[0, 1].set_xlabel('Log-Returns')
axs[0, 1].set_ylabel('Density')

# QQ plot of daily log-returns
stats.probplot(log_returns_daily, dist="norm", plot=axs[1, 0])
axs[1, 0].set_title('QQ Plot of Daily Log-Returns')
axs[1, 0].set_xlabel('Normal Quantiles')
axs[1, 0].set_ylabel('Sample Quantiles')
```

```
# QQ plot of monthly log-returns
stats.probplot(log_returns_weekly, dist="norm", plot=axs[1, 1])
axs[1, 1].set_title('QQ Plot of Weekly Log-Returns')
axs[1, 1].set_xlabel('Normal Quantiles')
axs[1, 1].set_ylabel('Sample Quantiles')

# Adjust spacing between plots
plt.tight_layout()

# Save the plot in pdf format
plt.savefig('Latex/Img/QQplot_daily_weekly_AMZN.pdf', format='pdf',␣
↪bbox_inches='tight')

# Show the plot
plt.show()
```



[118]:
```
import yfinance as yf
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats

# Extract daily log-returns
```

```python
log_returns_daily = rt_d_all.values.flatten()

# Create three side-by-side QQ plots
fig, axs = plt.subplots(1, 3, figsize=(18, 5))

# Normalize the data to have zero mean and unit variance
log_returns_daily_normalized = log_returns_daily / np.std(log_returns_daily)

# QQ plot against Student-t distribution with   = 10
stats.probplot(log_returns_daily_normalized, dist=stats.t, sparams=(10,), plot=axs[0])
axs[0].set_title('QQ Plot (Student-t  =10)')
axs[0].set_xlabel('Theoretical Quantiles')
axs[0].set_ylabel('Sample Quantiles')

# QQ plot against Student-t distribution with   = 5
stats.probplot(log_returns_daily_normalized, dist=stats.t, sparams=(5,), plot=axs[1])
axs[1].set_title('QQ Plot (Student-t  =5)')
axs[1].set_xlabel('Theoretical Quantiles')
axs[1].set_ylabel('Sample Quantiles')

# QQ plot against Student-t distribution with   = 3
stats.probplot(log_returns_daily_normalized, dist=stats.t, sparams=(3,), plot=axs[2])
axs[2].set_title('QQ Plot (Student-t  =3)')
axs[2].set_xlabel('Theoretical Quantiles')
axs[2].set_ylabel('Sample Quantiles')

# Adjust spacing between QQ plots
plt.tight_layout()

# Save the plot in png format
plt.savefig('Latex/Img/qqplt_tstudents_AMZNdaily.pdf', format='pdf', bbox_inches='tight')

plt.show()
```



Kurtosis of the sample

[119]:
```
from scipy.stats import kurtosis
exc_kurt = kurtosis(Rt_d_all) - 3
print("Excess Kurtosis = ", exc_kurt)
```

```
Excess Kurtosis =  10.51221657872248
```

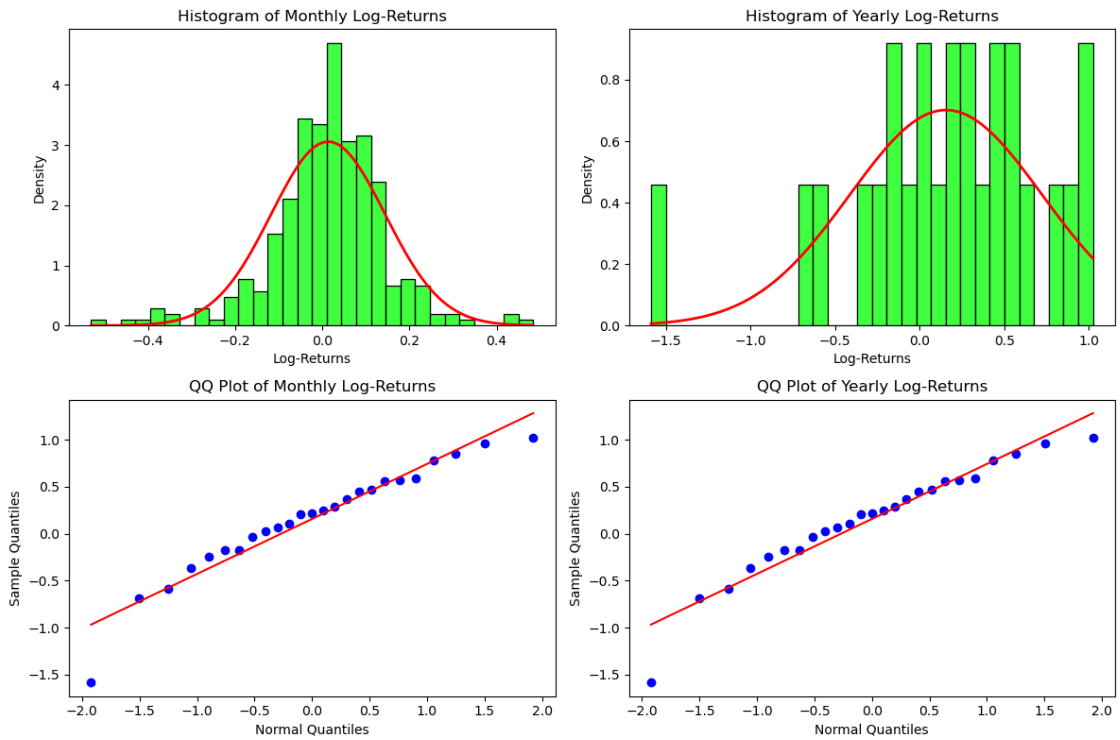
The sample of data defined by the simple returns of AMZN stock is HEAVY TAILED

12 *4.5/High Frequecy non-Gaussianity*

1. Overall Shapes
2. Lilliefors Test

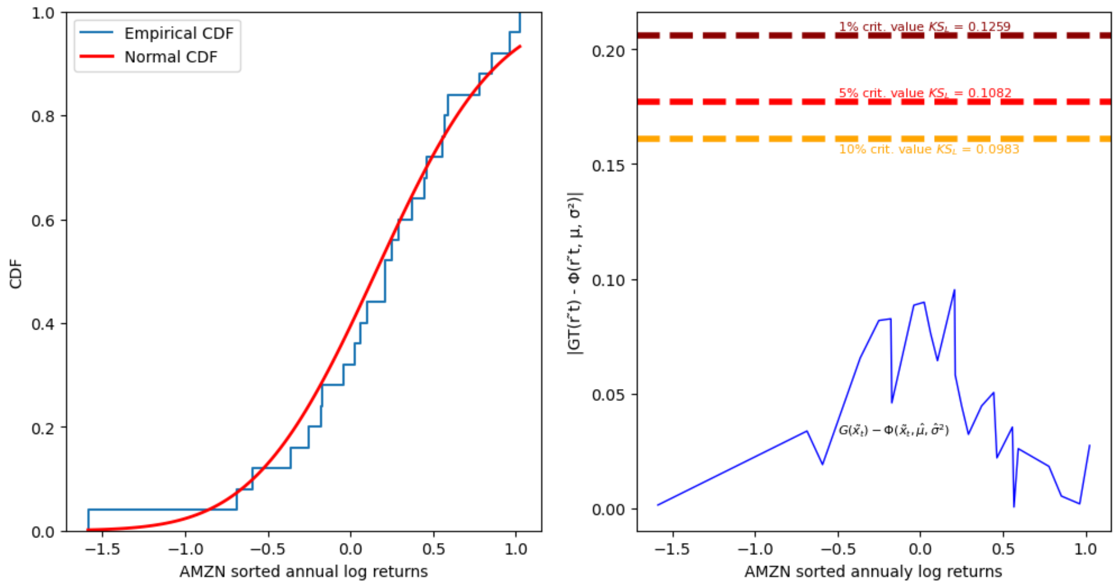
[120]:
```
import yfinance as yf
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats
import seaborn as sns

# Extract daily log-returns
log_price_mothly = pt_m_all  # Ensure this is a pandas DataFrame or Series
log_returns_monthly = rt_m_all  # Ensure this is a pandas DataFrame or Series

# If log_returns_daily is a DataFrame, convert it to a 1D array (assuming␣
↪'column_name' is the name of the column)
log_returns_monthly = log_returns_monthly.values.flatten()  # Ensure it's 1D␣
↪array

# Calculate monthly log-returns
log_price_yearly = pt_y_all  # Ensure this is a pandas DataFrame or Series
log_returns_yearly = rt_y_all  # Ensure this is a pandas DataFrame or Series

# If log_returns_monthly is a DataFrame, convert it to a 1D array (assuming␣
↪'column_name' is the name of the column)
log_returns_yearly = log_returns_yearly.values.flatten()  # Ensure it's 1D array

# Create the figure with four subplots
fig, axs = plt.subplots(2, 2, figsize=(12, 8))

# Plot histogram of daily log-returns
sns.histplot(log_returns_monthly, bins=30, color='lime', edgecolor='black',␣
↪kde_kws={'color': 'red'}, ax=axs[0, 0], stat='density')
```

```
axs[0, 0].plot(np.linspace(log_returns_monthly.min(), log_returns_monthly.
 ↪max(), 100),
               stats.norm.pdf(np.linspace(log_returns_monthly.min(),␣
 ↪log_returns_monthly.max(), 100),
                              log_returns_monthly.mean(), log_returns_monthly.
 ↪std()), color='red', linewidth=2)
axs[0, 0].set_title('Histogram of Monthly Log-Returns')
axs[0, 0].set_xlabel('Log-Returns')
axs[0, 0].set_ylabel('Density')

# Plot histogram of monthly log-returns
sns.histplot(log_returns_yearly, bins=30, color='lime', edgecolor='black',␣
 ↪kde_kws={'color': 'red'}, ax=axs[0, 1], stat='density')
axs[0, 1].plot(np.linspace(log_returns_yearly.min(), log_returns_yearly.max(),␣
 ↪100),
               stats.norm.pdf(np.linspace(log_returns_yearly.min(),␣
 ↪log_returns_yearly.max(), 100),
                              log_returns_yearly.mean(), log_returns_yearly.
 ↪std()), color='red', linewidth=2)
axs[0, 1].set_title('Histogram of Yearly Log-Returns')
axs[0, 1].set_xlabel('Log-Returns')
axs[0, 1].set_ylabel('Density')

# QQ plot of daily log-returns
stats.probplot(log_returns_yearly, dist="norm", plot=axs[1, 0])
axs[1, 0].set_title('QQ Plot of Monthly Log-Returns')
axs[1, 0].set_xlabel('Normal Quantiles')
axs[1, 0].set_ylabel('Sample Quantiles')

# QQ plot of monthly log-returns
stats.probplot(log_returns_yearly, dist="norm", plot=axs[1, 1])
axs[1, 1].set_title('QQ Plot of Yearly Log-Returns')
axs[1, 1].set_xlabel('Normal Quantiles')
axs[1, 1].set_ylabel('Sample Quantiles')

# Adjust spacing between plots
plt.tight_layout()

# Save the plot in pdf format
plt.savefig('Latex/Img/QQplot_monthly_yearly_AMZN.pdf', format='pdf',␣
 ↪bbox_inches='tight')

# Show the plot
plt.show()
```



[121]:
```
import yfinance as yf
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats
import seaborn as sns

# Compute annual log-returns
log_returns_weekly = rt_w_all

log_returns_yearly = rt_y_all
# Compute mean and std
mean_data = log_returns_yearly.mean()
sd_data = log_returns_yearly.std()
samp_size = len(log_returns_yearly)
seq_ind = np.arange(1, samp_size + 1, 1)
emp_cdf = seq_ind / samp_size
emp_cdf_2 = (seq_ind - 1) / samp_size
my_data_ordered = np.sort(log_returns_yearly)
theor_cdf = stats.norm.cdf(my_data_ordered, mean_data, sd_data)

# Set the layout
fig, axs = plt.subplots(1, 2, figsize=(12, 6))
```

```python
# Left panel: empirical and Normal cdf's
sns.ecdfplot(log_returns_yearly, ax=axs[0], label='Empirical CDF')
axs[0].plot(np.linspace(log_returns_yearly.min(), log_returns_yearly.max(), 100),
            stats.norm.cdf(np.linspace(log_returns_yearly.min(), log_returns_yearly.max(), 100),
                           mean_data, sd_data),
            color='red', linewidth=2, label='Normal CDF')
axs[0].set_xlabel('AMZN sorted annual log returns')
axs[0].set_ylabel('CDF')
axs[0].set_title('')
axs[0].legend()

# Right panel: Lilliefors test
KS_L_stat1 = np.max(np.abs(emp_cdf - theor_cdf))
KS_L_stat2 = np.max(np.abs(emp_cdf_2 - theor_cdf))
KS_L_stat = max(KS_L_stat1, KS_L_stat2)
axs[1].plot(my_data_ordered, np.abs(emp_cdf_2 - theor_cdf), color='blue', linewidth=1)
axs[1].axhline(y=0.805/np.sqrt(samp_size), color='orange', linewidth=4, linestyle='--')
axs[1].axhline(y=0.886/np.sqrt(samp_size), color='red', linewidth=4, linestyle='--')
axs[1].axhline(y=1.031/np.sqrt(samp_size), color='darkred', linewidth=4, linestyle='--')
axs[1].text(-0.5, 0.805/np.sqrt(samp_size)-0.006, '10% crit. value $KS_L$ = 0.0983', fontsize=8, color='orange')
axs[1].text(-0.5, 0.886/np.sqrt(samp_size)+0.002, '5% crit. value $KS_L$ = 0.1082', fontsize=8, color='red')
axs[1].text(-0.5, 1.031/np.sqrt(samp_size)+0.002, '1% crit. value $KS_L$ = 0.1259', fontsize=8, color='darkred')
axs[1].text(-0.5, 0.032, '$ G(\\tilde{x}_t)-\\Phi(\\tilde{x}_t, \\hat{\\mu}, \\hat{\\sigma}^2)$', fontsize=8)
axs[1].set_xlabel('AMZN sorted annualy log returns')
axs[1].set_ylabel('|GT(r˜t) - Φ(r˜t,  ,  ²)|')
axs[1].set_title('')

# Set the space within plots
#plt.tight_layout()

# Save the figure in png format
plt.savefig('Latex/Img/lillie_test_AMZNannualy.pdf', format='pdf', bbox_inches='tight')

plt.show()
```

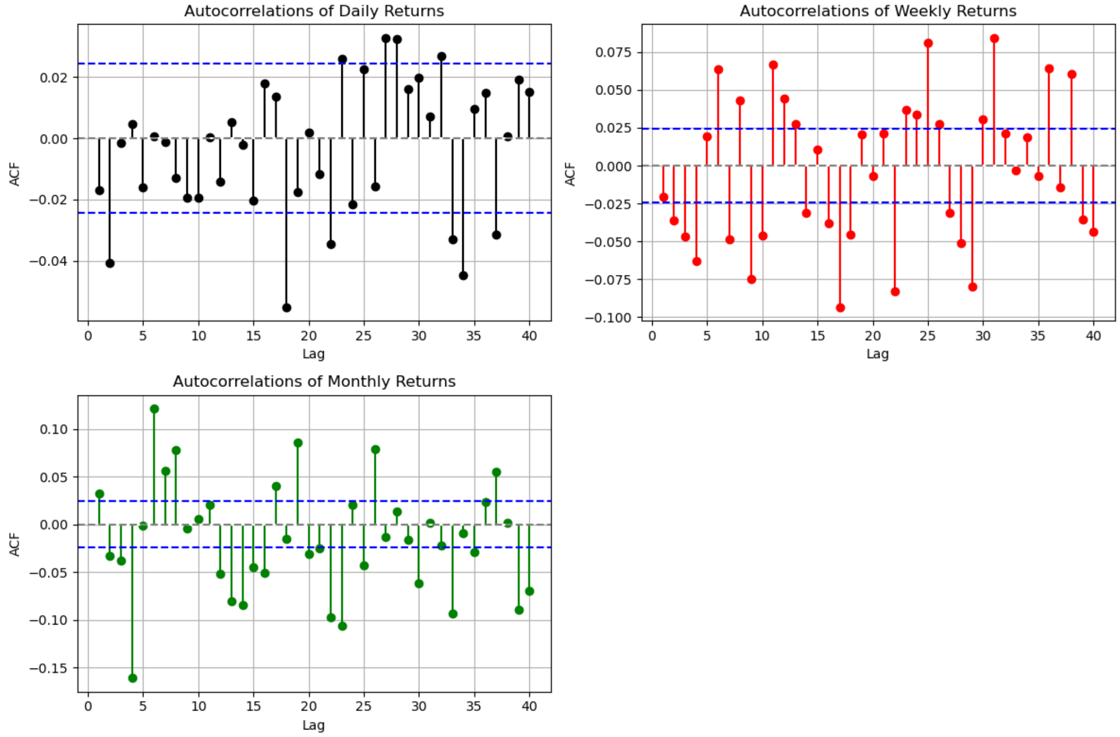


13 *4.6/ Returns are not autocorrelated*

1. Daily, Weekly and Monthly Autocorrelations

[122]:
```
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import acf

# Calculate empirical autocorrelations for daily, weekly, monthly, and yearly␣
↪returns
lags = 40

# Daily ACF
acf_daily_values = acf(Rt_d_all, nlags=lags)

# Weekly ACF
acf_weekly_values = acf(Rt_w_all, nlags=lags)

# Monthly ACF
acf_monthly_values = acf(Rt_m_all, nlags=lags)

# Calculate Bartlett intervals
Bart_Int = 1.96 / np.sqrt(len(Rt_d_all))

# Create the autocorrelation plot with Bartlett intervals for each time frame
plt.figure(figsize=(12, 8))
```

```python
# Plot daily autocorrelations
plt.subplot(2, 2, 1)
plt.stem(np.arange(1, lags + 1), acf_daily_values[1:], linefmt='k-',
  markerfmt='ko', basefmt='w-')
plt.axhline(y=0, color='gray', linestyle='--')
plt.axhline(y=Bart_Int, color='blue', linestyle='--')
plt.axhline(y=-Bart_Int, color='blue', linestyle='--')
plt.title('Autocorrelations of Daily Returns')
plt.xlabel('Lag')
plt.ylabel('ACF')
plt.grid(True)

# Plot weekly autocorrelations
plt.subplot(2, 2, 2)
plt.stem(np.arange(1, lags + 1), acf_weekly_values[1:], linefmt='r-',
  markerfmt='ro', basefmt='w-')
plt.axhline(y=0, color='gray', linestyle='--')
plt.axhline(y=Bart_Int, color='blue', linestyle='--')
plt.axhline(y=-Bart_Int, color='blue', linestyle='--')
plt.title('Autocorrelations of Weekly Returns')
plt.xlabel('Lag')
plt.ylabel('ACF')
plt.grid(True)

# Plot monthly autocorrelations
plt.subplot(2, 2, 3)
plt.stem(np.arange(1, lags + 1), acf_monthly_values[1:], linefmt='g-',
  markerfmt='go', basefmt='w-')
plt.axhline(y=0, color='gray', linestyle='--')
plt.axhline(y=Bart_Int, color='blue', linestyle='--')
plt.axhline(y=-Bart_Int, color='blue', linestyle='--')
plt.title('Autocorrelations of Monthly Returns')
plt.xlabel('Lag')
plt.ylabel('ACF')
plt.grid(True)

# Adjust layout and show plot
plt.tight_layout()
plt.show()
```




```
[123]: """autocorrelate = pt_d_all.shift(1).corrwith(pt_d_all, method='pearson')
       print(autocorrelate.round(4))"""
```

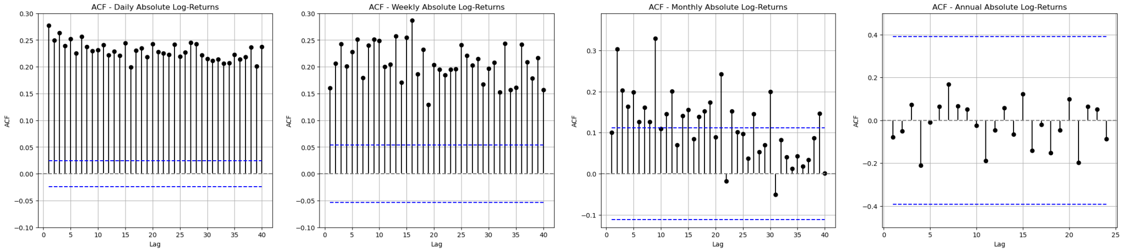
```
[123]: "autocorrelate = pt_d_all.shift(1).corrwith(pt_d_all,
       method='pearson')\nprint(autocorrelate.round(4))"
```

14 *4.7/ Returns feature volatility clustering long run range dependence of squared returns*

```
[124]: # Extract daily log-returns
       log_returns_daily = rt_d_all

       # Parameter for the empirical autocorrelation
       lags = 40

       # Creation of the three side-by-side graphs
       fig, axs = plt.subplots(1, 4, figsize=(30, 6))

       # ACF of daily log-returns with confidence bands
       acf_values_daily = acf(abs(log_returns_daily), nlags=lags)
       confint = 1.96 / np.sqrt(len(log_returns_daily))
       confint_upper = np.full(lags, confint)
```

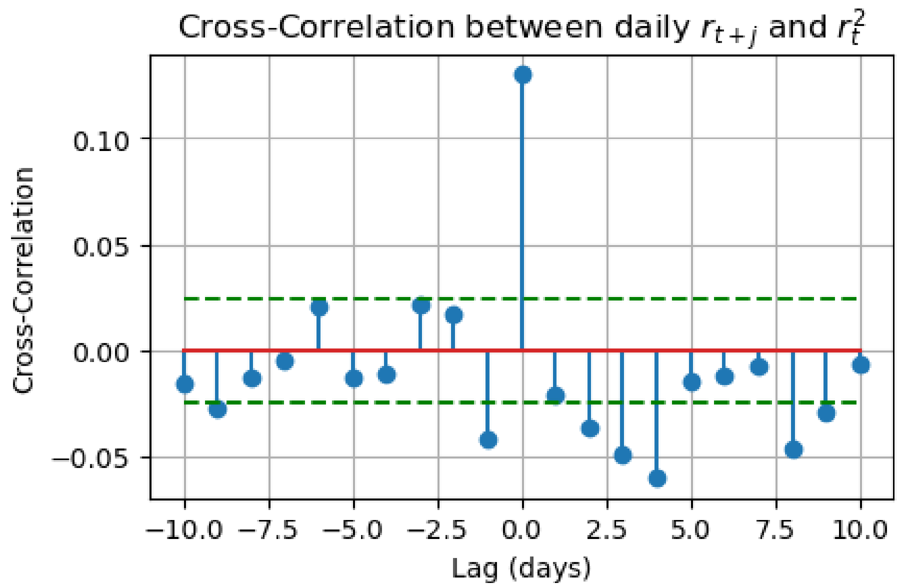
```
confint_lower = -np.full(lags, confint)

axs[0].stem(np.arange(1, lags + 1), acf_values_daily[1:], linefmt='k-',␣
↪markerfmt='ko', basefmt='w-')
axs[0].axhline(y=0, color='gray', linestyle='--')
axs[0].plot(np.arange(1, lags + 1), confint_upper, color='blue',␣
↪linestyle='dashed')
axs[0].plot(np.arange(1, lags + 1), confint_lower, color='blue',␣
↪linestyle='dashed')
axs[0].set_ylim(-0.1, 0.3)
axs[0].set_title('ACF - Daily Absolute Log-Returns')
axs[0].set_xlabel('Lag')
axs[0].set_ylabel('ACF')
axs[0].grid(True)

# ACF of weekly log-returns with confidence bands
acf_values_weekly = acf(abs(log_returns_weekly), nlags=lags)
confint_weekly = 1.96 / np.sqrt(len(log_returns_weekly))
confint_weekly_upper = np.full(lags, confint_weekly)
confint_weekly_lower = -np.full(lags, confint_weekly)

axs[1].stem(np.arange(1, lags + 1), acf_values_weekly[1:], linefmt='k-',␣
↪markerfmt='ko', basefmt='w-')
axs[1].axhline(y=0, color='gray', linestyle='--')
axs[1].plot(np.arange(1, lags + 1), confint_weekly_upper, color='blue',␣
↪linestyle='dashed')
axs[1].plot(np.arange(1, lags + 1), confint_weekly_lower, color='blue',␣
↪linestyle='dashed')
axs[1].set_ylim(-0.1, 0.3)
axs[1].set_title('ACF - Weekly Absolute Log-Returns')
axs[1].set_xlabel('Lag')
axs[1].set_ylabel('ACF')
axs[1].grid(True)

# ACF of monthly log-returns with confidence bands
acf_values_monthly = acf(abs(log_returns_monthly), nlags=lags)
confint_monthly = 1.96 / np.sqrt(len(log_returns_monthly))
confint_monthly_upper = np.full(lags, confint_monthly)
confint_monthly_lower = -np.full(lags, confint_monthly)

axs[2].stem(np.arange(1, lags + 1), acf_values_monthly[1:], linefmt='k-',␣
↪markerfmt='ko', basefmt='w-')
axs[2].axhline(y=0, color='gray', linestyle='--')
axs[2].plot(np.arange(1, lags + 1), confint_monthly_upper, color='blue',␣
↪linestyle='dashed')
```

```
axs[2].plot(np.arange(1, lags + 1), confint_monthly_lower, color='blue', linestyle='dashed')
axs[2].set_ylim(-0.13, 0.39)
axs[2].set_title('ACF - Monthly Absolute Log-Returns')
axs[2].set_xlabel('Lag')
axs[2].set_ylabel('ACF')
axs[2].grid(True)

# ACF of annual log-returns with confidence bands
lags = 24

acf_values_yearly = acf(abs(log_returns_yearly), nlags=lags)
confint = 1.96 / np.sqrt(len(log_returns_yearly))
confint_upper = np.full(lags, confint)
confint_lower = -np.full(lags, confint)

axs[3].stem(np.arange(1, lags + 1), acf_values_yearly[1:], linefmt='k-', markerfmt='ko', basefmt='w-')
axs[3].axhline(y=0, color='gray', linestyle='--')
axs[3].plot(np.arange(1, lags + 1), confint_upper, color='blue', linestyle='dashed')
axs[3].plot(np.arange(1, lags + 1), confint_lower, color='blue', linestyle='dashed')
axs[3].set_ylim(-0.5, 0.5)
axs[3].set_title('ACF - Annual Absolute Log-Returns')
axs[3].set_xlabel('Lag')
axs[3].set_ylabel('ACF')
axs[3].grid(True)

# Adjusting the spacing between graphs
#plt.tight_layout()

# Save the graphic in png format
plt.savefig('Latex/Img/Fact7_AbsoluteLogReturns.pdf', format='pdf', bbox_inches='tight')

plt.show()
```



15 *4.8/ Leverage Effect*

[ ]:
```python
# Define a function
def ccf(x, y, lag_max = 100):
    # Compute correlation
    result = ss.correlate(y - np.mean(y), x - np.mean(x), method='direct') /
 ↪(np.std(y) * np.std(x) * len(y))
    # Define the length
    length = (len(result) - 1) // 2
    lo = length - lag_max
    hi = length + (lag_max + 1)
    return result[lo:hi]

# Choose the max lag and execute the function
lag_max = 10
log_returns_daily = np.array(log_returns_daily)
cross_corr = ccf(log_returns_daily,log_returns_daily**2,lag_max=lag_max)

# Plot results
lags = np.arange(-lag_max, lag_max + 1)

# ACF of monthly log-returns with confidence bands
confint_daily = 1.96 / np.sqrt(len(log_returns_daily))
confint_daily_upper = np.full(len(lags), confint_daily)
confint_daily_lower = -np.full(len(lags), confint_daily)

plt.figure(figsize=(5, 3))
plt.stem(lags, cross_corr)
plt.plot(lags, confint_daily_upper, color='green', linestyle='dashed')
plt.plot(lags, confint_daily_lower, color='green', linestyle='dashed')
plt.xlabel('Lag (days)')
plt.ylabel('Cross-Correlation')
plt.title('Cross-Correlation between daily $r_{t+j}$ and $r_t^2$')
plt.grid(True)

# Add the bartlet intervals

plt.savefig('Latex/Img/Fact8_CrossCorr_r_r2.pdf', format='pdf',
 ↪bbox_inches='tight')

plt.show()
```



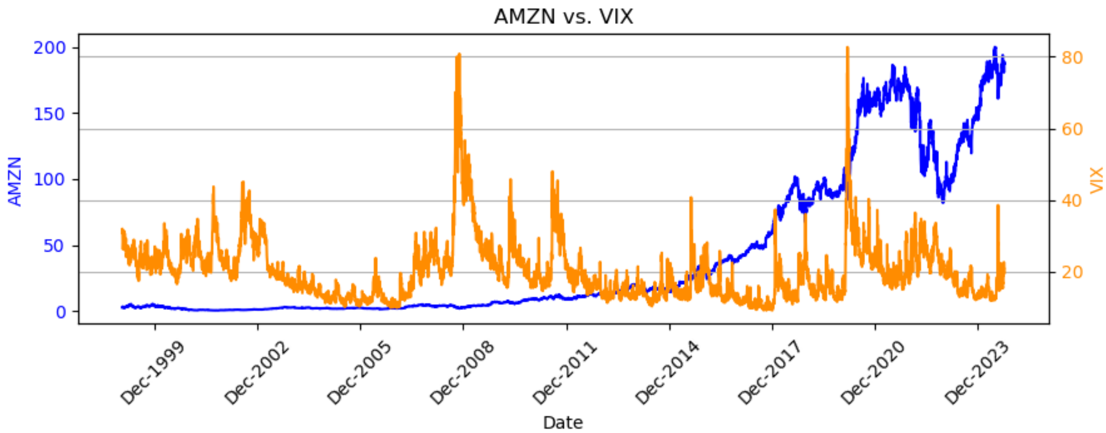
[126]:
```
print(Pt_d_all, type)
```

```
Date
1999-01-21      2.650000
1999-01-22      3.075000
1999-01-25      2.809375
1999-01-26      2.877344
1999-01-27      3.140625
                  …
2024-10-09    185.169998
2024-10-10    186.649994
2024-10-11    188.820007
2024-10-14    187.539993
2024-10-15    187.690002
Name: Pt.d, Length: 6476, dtype: float64 <class 'type'>
```

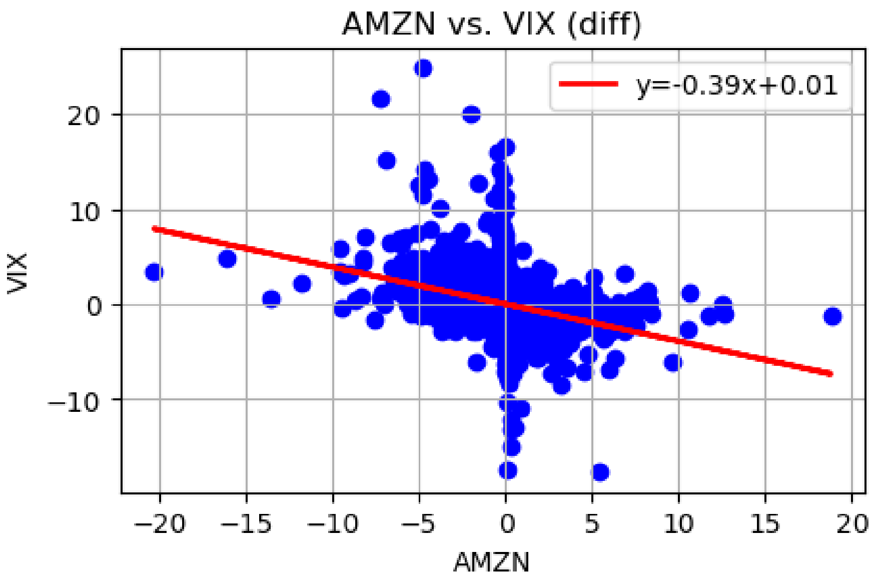
[127]:
```
#Get the starting and ending date of our stock
start_date = Pt_d_all.index.min()
end_date = Pt_d_all.index.max()

# Get VIX data
VIX = yf.download("^VIX", start=start_date, end=end_date)

# Extract and Rename the adjusted closing prices
VIX_d = VIX["Adj Close"]
VIX_d.name = 'VIX.d'
```

```python
# Mutate the Index into a DatetimeIndex
VIX_d.index = pd.to_datetime(VIX_d.index)

# Merge the two datasets and rename columns
merged_df = pd.merge(Pt_d_all, VIX_d, on='Date', how='outer') # outer: only␣
↪commond indexes (dates)
merged_df.head()

# Compute changes in pt and VIX compared to previous period (NaN are kept)
diff_df = merged_df.diff()
diff_df.head()

# Remove from the price dataframe
merged_df = merged_df.dropna()
# And from the second one
diff_df = diff_df.dropna()

# Define the figure parameters
fig, ax1 = plt.subplots(figsize=(10, 3))

# Customizing x-axis labels for December of each year
date_labels = pd.date_range(start=start_date, end=end_date, freq='3Y')
formatted_labels = [f'Dec-{date.year}' for date in date_labels]
# Add label and rotate them
plt.xticks(date_labels, formatted_labels, rotation=45)

# Work on the first y-axis: S&P
ax1.plot(merged_df.index, merged_df['Pt.d'], label="AMZN" + ' Prices',␣
↪color='blue')
ax1.set_xlabel('Date')
ax1.set_ylabel("AMZN", color='blue')
ax1.tick_params(axis='y', labelcolor='blue')

# Work on the second y-axis: VIX
ax2 = ax1.twinx()
ax2.plot(merged_df.index, merged_df['VIX.d'], label='VIX', color='darkorange')
ax2.set_ylabel('VIX', color='darkorange')
ax2.tick_params(axis='y', labelcolor='darkorange')

# Adjust the figure
plt.title("AMZN" + ' vs. VIX')
plt.grid(True)

# Save the figure
plt.savefig('Latex/Img/Fact8.pdf', format='png', bbox_inches='tight')
plt.show()
```

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



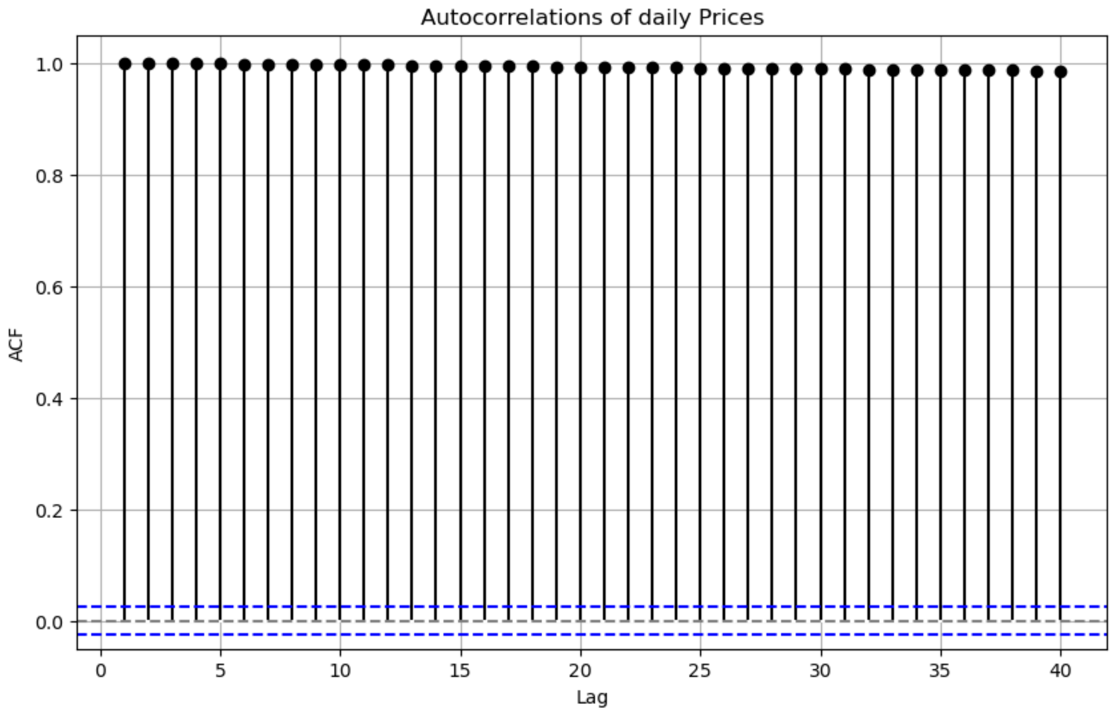
[128]:
```
plt.figure(figsize=(5, 3))
plt.scatter(diff_df['Pt.d'], diff_df['VIX.d'], color='blue', marker='o')

# Add labels and title
plt.xlabel("AMZN")
plt.ylabel('VIX')
plt.title("AMZN" + ' vs. VIX (diff)')
plt.grid(True)

# Add regression line
coefficients = np.polyfit(diff_df['Pt.d'], diff_df['VIX.d'], 1)
regression_line = np.polyval(coefficients, diff_df['Pt.d'])
plt.plot(diff_df['Pt.d'], regression_line, color='red', linewidth=2,␣
  ↪label='y='+str(round(coefficients[0],2))+'x+'+str(round(coefficients[1],2)))
plt.legend()

plt.savefig('Latex/Img/Fact_8_3'+"AMZN"+'_.pdf', format='pdf',␣
  ↪bbox_inches='tight')

# Show plot
plt.show()
```



15.0.1 Other Material

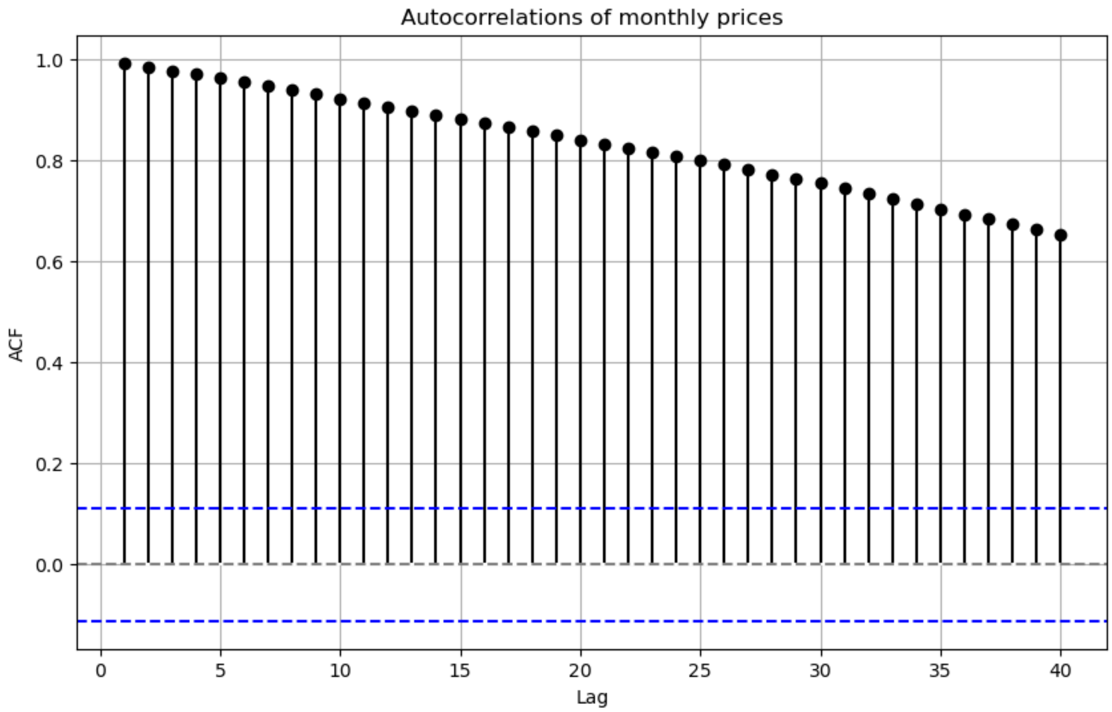
took from stylized facts 1

[129]:
```
from statsmodels.tsa.stattools import acf

# Calculate empirical autocorrelation
lags = 40
acf_values = acf(pt_d_all, nlags=lags)

# Calculate Bartlett intervals
Bart_Int = 1.96 / np.sqrt(len(pt_d_all))

# Create the autocorrelation plot with Bartlett intervals
plt.figure(figsize=(10, 6))
plt.stem(np.arange(1, lags + 1), acf_values[1:], linefmt='k-', markerfmt='ko',␣
  ↪basefmt='w-')
plt.axhline(y=0, color='gray', linestyle='--')
plt.axhline(y=Bart_Int, color='blue', linestyle='--')
plt.axhline(y=-Bart_Int, color='blue', linestyle='--')
plt.title('Autocorrelations of daily Prices')
plt.xlabel('Lag')
plt.ylabel('ACF')
plt.grid(True)
#plt.savefig('Latex/Autocorrel_daily.pdf', format='pdf', bbox_inches='tight')
plt.show()
```



ACF with monthly data

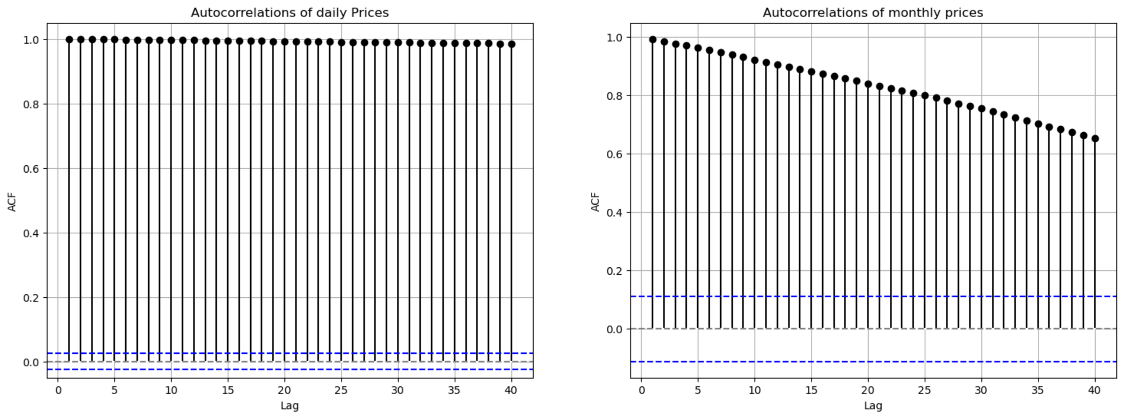
[130]:
```
from statsmodels.tsa.stattools import acf

# Calculate empirical autocorrelation
lags = 40
acf_values = acf(pt_m_all, nlags=lags)

# Calculate Bartlett intervals
Bart_Int = 1.96 / np.sqrt(len(pt_m_all))

# Create the autocorrelation plot with Bartlett intervals
plt.figure(figsize=(10, 6))
plt.stem(np.arange(1, lags + 1), acf_values[1:], linefmt='k-', markerfmt='ko',
    basefmt='w-')
plt.axhline(y=0, color='gray', linestyle='--')
plt.axhline(y=Bart_Int, color='blue', linestyle='--')
plt.axhline(y=-Bart_Int, color='blue', linestyle='--')
plt.title('Autocorrelations of monthly prices')
plt.xlabel('Lag')
plt.ylabel('ACF')
plt.grid(True)
#plt.savefig('Latex/Autocorrel_monthly.pdf', format='pdf', bbox_inches='tight')
plt.show()
```



15.0.2 The two figures subplotted

[131]:
```
fig, axs = plt.subplots(1, 2, figsize=(18, 6))
#first fig
# Calculate empirical autocorrelation
lags = 40
acf_values = acf(pt_d_all, nlags=lags)

# Calculate Bartlett intervals
Bart_Int = 1.96 / np.sqrt(len(pt_d_all))

axs[0].stem(np.arange(1, lags + 1), acf_values[1:], linefmt='k-',␣
 ↪markerfmt='ko', basefmt='w-')
axs[0].axhline(y=0, color='gray', linestyle='--')
axs[0].axhline(y=Bart_Int, color='blue', linestyle='--')
axs[0].axhline(y=-Bart_Int, color='blue', linestyle='--')
axs[0].set_title('Autocorrelations of daily Prices')
axs[0].set_xlabel('Lag')
axs[0].set_ylabel('ACF')
axs[0].grid(True)
#second fig
# Calculate empirical autocorrelation
lags = 40
acf_values = acf(pt_m_all, nlags=lags)
```

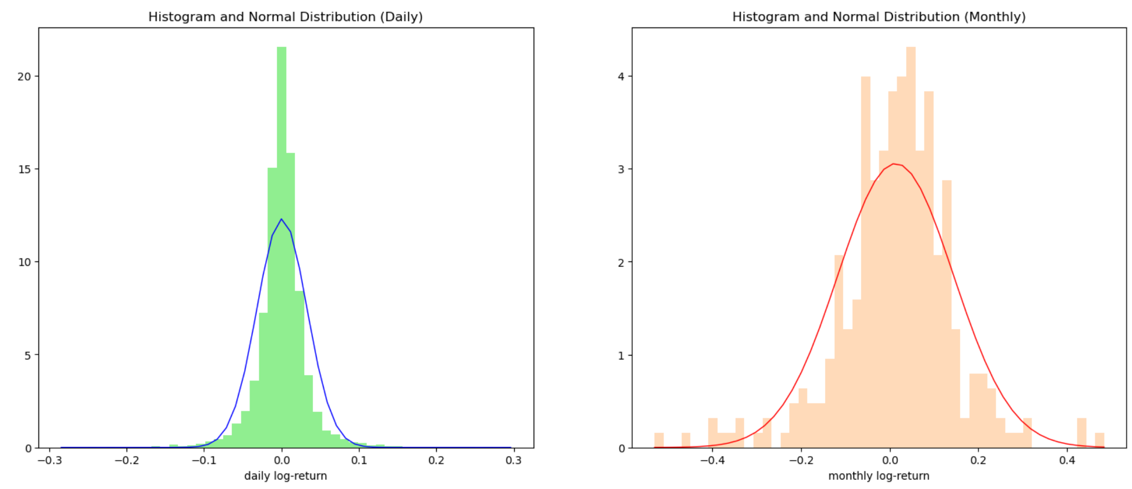
```
# Calculate Bartlett intervals
Bart_Int = 1.96 / np.sqrt(len(pt_m_all))

axs[1].stem(np.arange(1, lags + 1), acf_values[1:], linefmt='k-',␣
 ↪markerfmt='ko', basefmt='w-')
axs[1].axhline(y=0, color='gray', linestyle='--')
axs[1].axhline(y=Bart_Int, color='blue', linestyle='--')
axs[1].axhline(y=-Bart_Int, color='blue', linestyle='--')
axs[1].set_title('Autocorrelations of monthly prices')
axs[1].set_xlabel('Lag')
axs[1].set_ylabel('ACF')
axs[1].grid(True)

plt.savefig('Latex/Img/Autocorrel_daily_monthly.pdf', format='pdf',␣
 ↪bbox_inches='tight')

plt.show()
```



Histogram of daily prices and normal density

[132]:
```
# Set up the subplots
fig, axs = plt.subplots(1, 2, figsize=(18, 7))

# Histogram and Normal Distribution (Daily)
axs[0].hist(rt_d_all, bins=50, density=True, color="lightgreen")
norm_y = stats.norm.pdf(np.linspace(rt_d_all.min(), rt_d_all.max()), loc=np.
 ↪mean(rt_d_all), scale=np.std(rt_d_all))
axs[0].plot(np.linspace(rt_d_all.min(), rt_d_all.max()), norm_y, color="blue",␣
 ↪linewidth=1)
axs[0].set_xlabel("daily log-return")
axs[0].set_title("Histogram and Normal Distribution (Daily)")
```

```
# Histogram and Normal Distribution (Monthly)
axs[1].hist(rt_m_all, bins=50, density=True, color="peachpuff")
norm_y = stats.norm.pdf(np.linspace(rt_m_all.min(), rt_m_all.max()), loc=np.
    ↪mean(rt_m_all), scale=np.std(rt_m_all))
axs[1].plot(np.linspace(rt_m_all.min(), rt_m_all.max()), norm_y, color="red",␣
    ↪linewidth=1)
axs[1].set_xlabel("monthly log-return")
axs[1].set_title("Histogram and Normal Distribution (Monthly)")

# Adjust layout and display the plot
#plt.tight_layout()
plt.savefig('Latex/Img/Hitogram_and_normal_distrib_daily.pdf', format='pdf',␣
    ↪bbox_inches='tight')
plt.show()
```

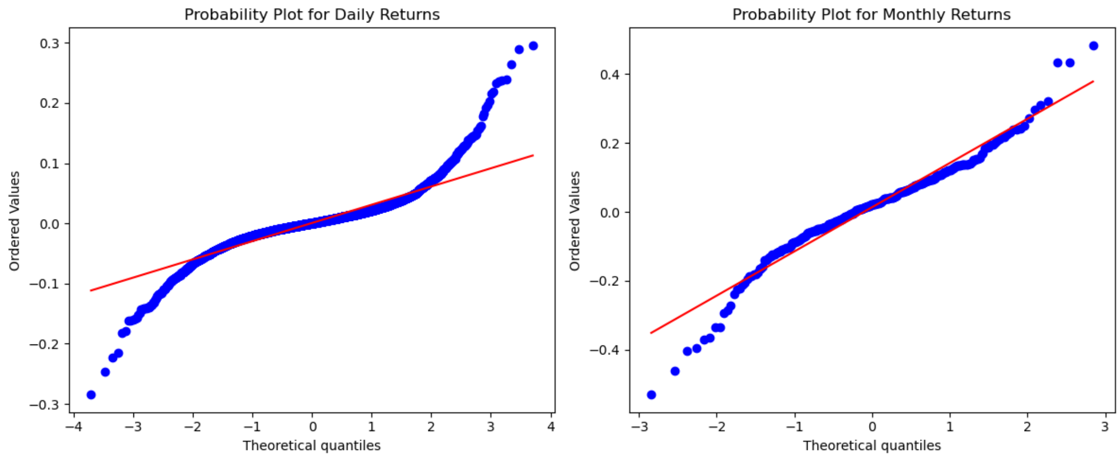


15.0.3 QQ-plot (Normal distribution)

[133]:

```
# Set up the subplots
fig, axs = plt.subplots(1, 2, figsize=(12, 5))

# Probability Plot for Daily Returns
stats.probplot(rt_d_all, dist="norm", plot=axs[0])
axs[0].set_title("Probability Plot for Daily Returns")

# Probability Plot for Monthly Returns
stats.probplot(rt_m_all, dist="norm", plot=axs[1])
axs[1].set_title("Probability Plot for Monthly Returns")

# Adjust layout and display the plot
```

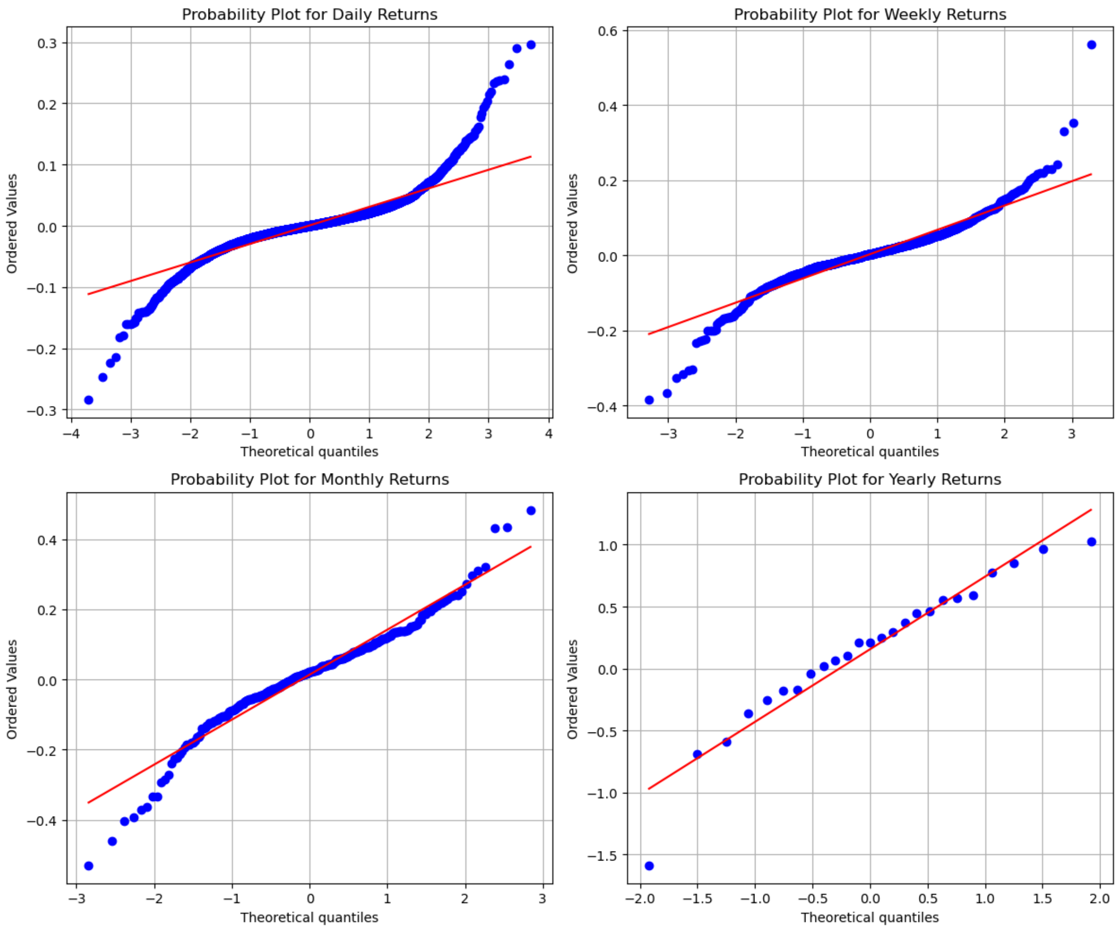
```
plt.tight_layout()
#plt.savefig('Latex/QQ-plot.pdf', format='pdf', bbox_inches='tight')
plt.show()
```



Check how the QQ-plots change aggregating data

[134]:
```
# Set up the subplots
fig, axs = plt.subplots(2, 2, figsize=(12, 10))

# Probability Plot for Daily Returns
stats.probplot(rt_d_all, dist="norm", plot=axs[0, 0])
axs[0, 0].set_title("Probability Plot for Daily Returns")
axs[0, 0].grid(True)

# Probability Plot for Weekly Returns
stats.probplot(rt_w_all, dist="norm", plot=axs[0, 1])
axs[0, 1].set_title("Probability Plot for Weekly Returns")
axs[0, 1].grid(True)

# Probability Plot for Monthly Returns
stats.probplot(rt_m_all, dist="norm", plot=axs[1, 0])
axs[1, 0].set_title("Probability Plot for Monthly Returns")
axs[1, 0].grid(True)

# Probability Plot for Yearly Returns
stats.probplot(rt_y_all, dist="norm", plot=axs[1, 1])
axs[1, 1].set_title("Probability Plot for Yearly Returns")
axs[1, 1].grid(True)

# Adjust layout and display the plot
plt.tight_layout()
```

```
plt.show()
```

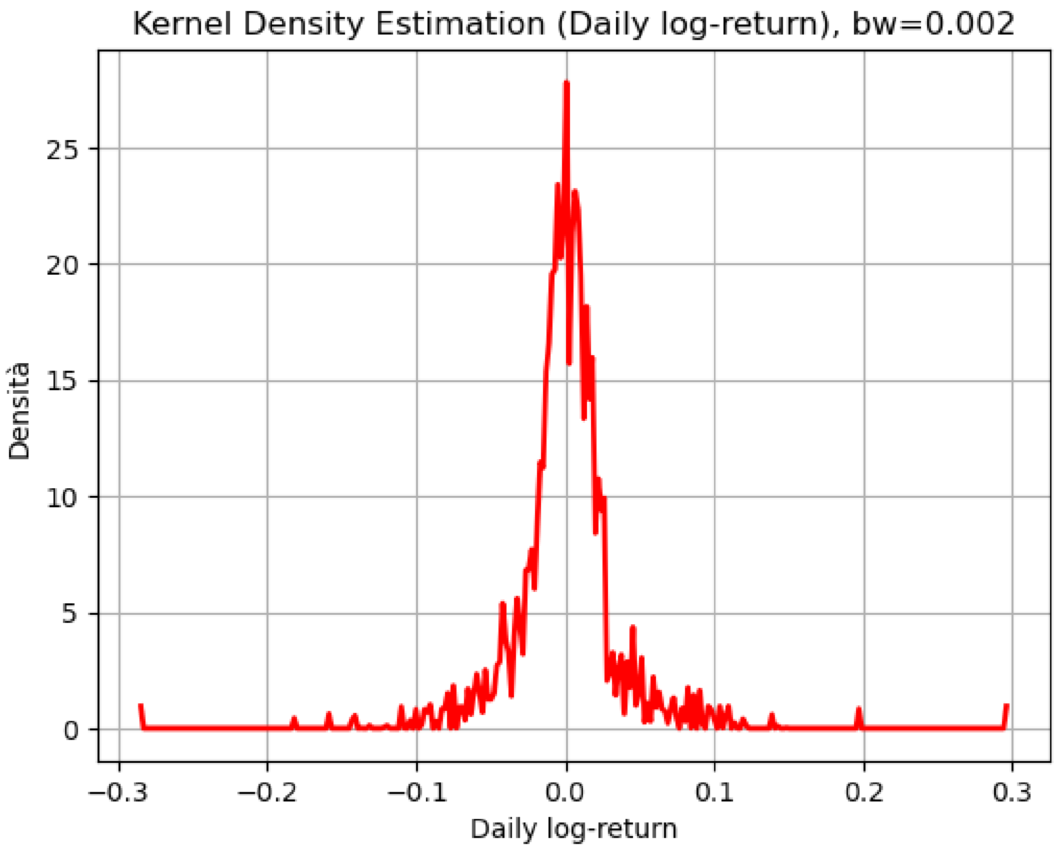


15.0.4 Kernel density

It is similar to a smooth histogram!

[135]:
```
## Compute the kernel density: daily returns
# divide the interval between the min and max returns into 300 segments
density_eval_points = np.linspace(rt_d_all.min(), rt_d_all.max(), num=300)
# estimate the kernel density of our returns
kde = gaussian_kde(rt_d_all, bw_method=0.002)
# and evaluate in the interval defined above
density_estimation = kde(density_eval_points)

# Plotting
plt.plot(density_eval_points, density_estimation, color='red', lw=2,␣
 ↪label='Kernel density')
plt.xlabel("Daily log-return")
```

```python
plt.ylabel("Densità")
plt.title("Kernel Density Estimation (Daily log-return), bw=0.002")
plt.grid(True)
plt.show()
```

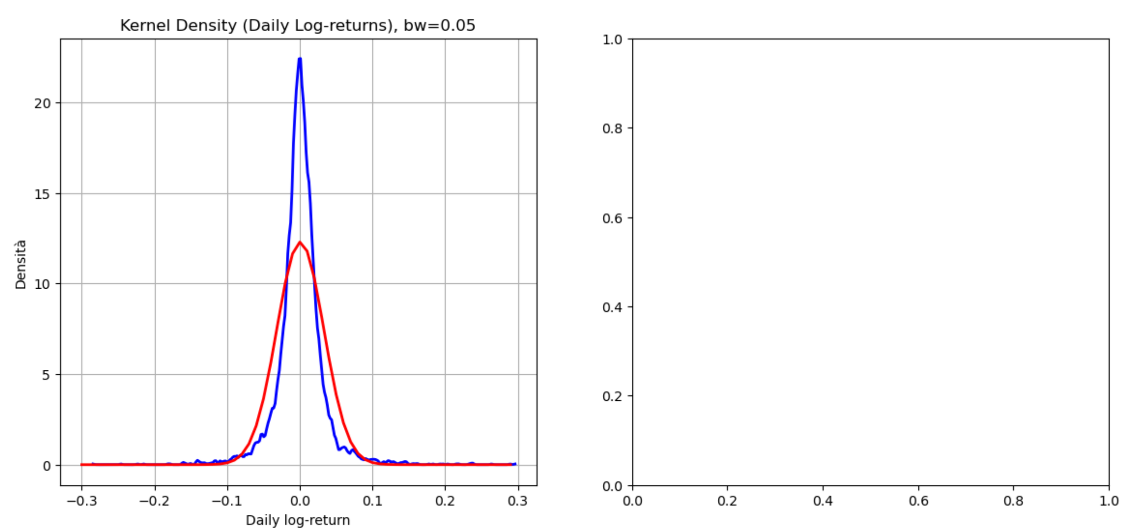


bw_method defines the *bandwidth parameter*:

⇒ the larger the bandwidth, the smoother the histogram:

[136]:
```python
## Compute the kernel density: daily returns
# divide the interval between the min and max returns into 300 segments
density_eval_points = np.linspace(rt_d_all.min(), rt_d_all.max(), num=300)
# estimate the kernel density of our returns
kde = gaussian_kde(rt_d_all, bw_method=0.05)
# and evaluate in the interval defined above
density_estimation = kde(density_eval_points)

# Empirical mean and std
mean_empirical= log_returns_daily.mean()
std_empirical= log_returns_daily.std()
```

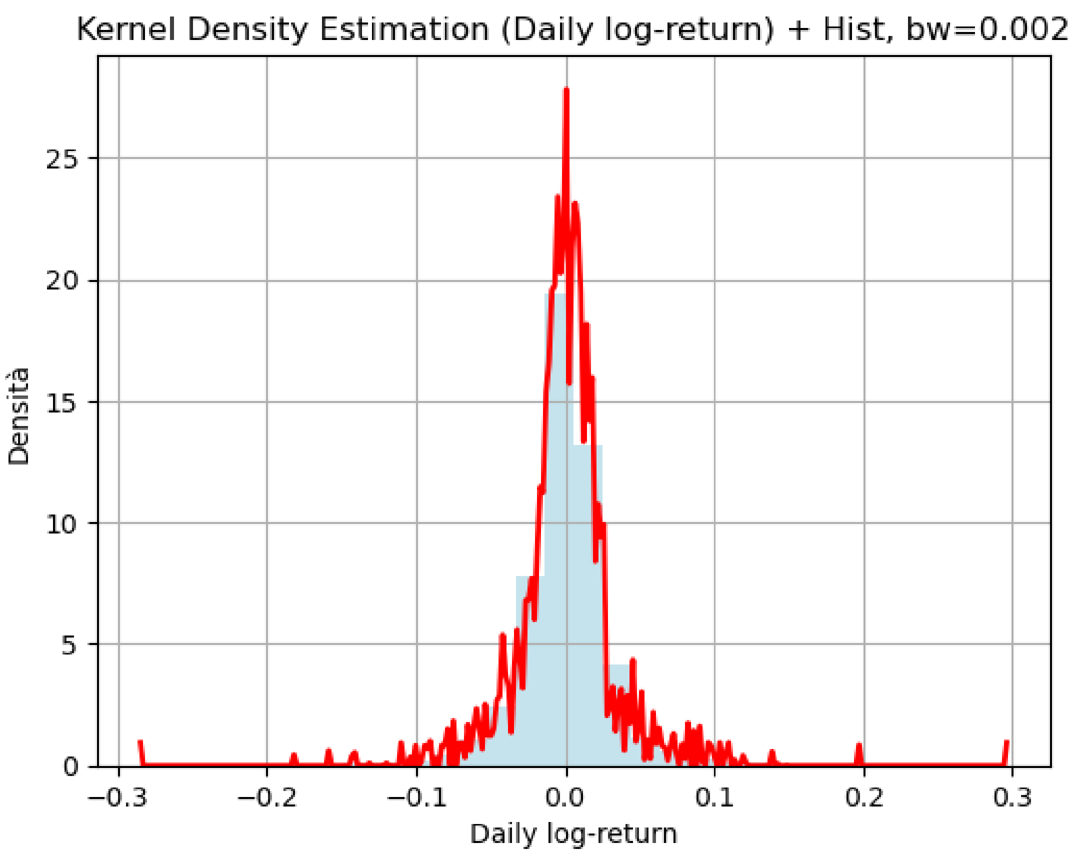
```python
x=np.arange(-0.3,0.3,0.01)


fig,axs=plt.subplots(1,2,figsize=(14,6))
# Plotting

#sns.kdeplot(log_returns_daily, color='blue', ax=axs[0])

# 1rst plot is kernel density daily log returns, compared to the standard normal
axs[0].plot(density_eval_points, density_estimation, color='blue', lw=2, label='Kernel density')
axs[0].plot(x, stats.norm.pdf(x, mean_empirical, std_empirical), color='red', linewidth=2)
axs[0].set_xlabel("Daily log-return")
axs[0].set_ylabel("Densità")
axs[0].set_title("Kernel Density (Daily Log-returns), bw=0.05")
axs[0].grid(True)

"""sns.histplot(log_returns_daily, bins=60, color='lime', edgecolor='black', kde_kws={'color': 'red'}, stat='density', ax=axs[1])
axs[1].plot(stats.norm.pdf(np.linspace(log_returns_daily.min(), log_returns_daily.max(), 100),log_returns_daily.mean(), log_returns_daily.std()),color='red', linewidth=2)
"""
plt.show()
```



Above, there was an issue with a histogramm, but we did not used it in the interpretation

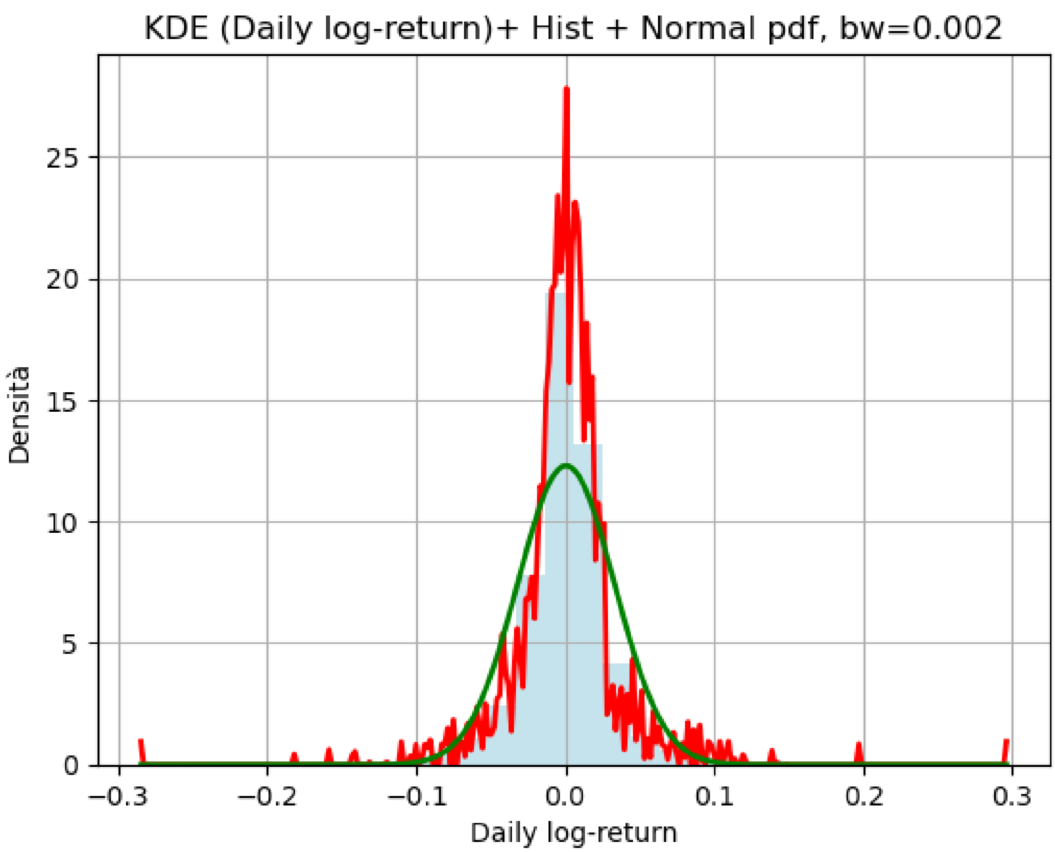
[137]:

```
## Compute the kernel density: daily returns
# divide the interval between the min and max returns into 300 segments
density_eval_points = np.linspace(rt_d_all.min(), rt_d_all.max(), num=300)
# estimate the kernel density of our returns
kde = gaussian_kde(rt_d_all, bw_method=0.002)
# and evaluate in the interval defined above
density_estimation = kde(density_eval_points)

# Plotting
plt.hist(rt_d_all, bins=30, density=True, alpha=0.7, color='lightblue')
plt.plot(density_eval_points, density_estimation, color='red', lw=2,␣
 ↪label='Kernel density')
plt.xlabel("Daily log-return")
plt.ylabel("Densità")
plt.title("Kernel Density Estimation (Daily log-return) + Hist, bw=0.002")
plt.grid(True)
plt.show()
```



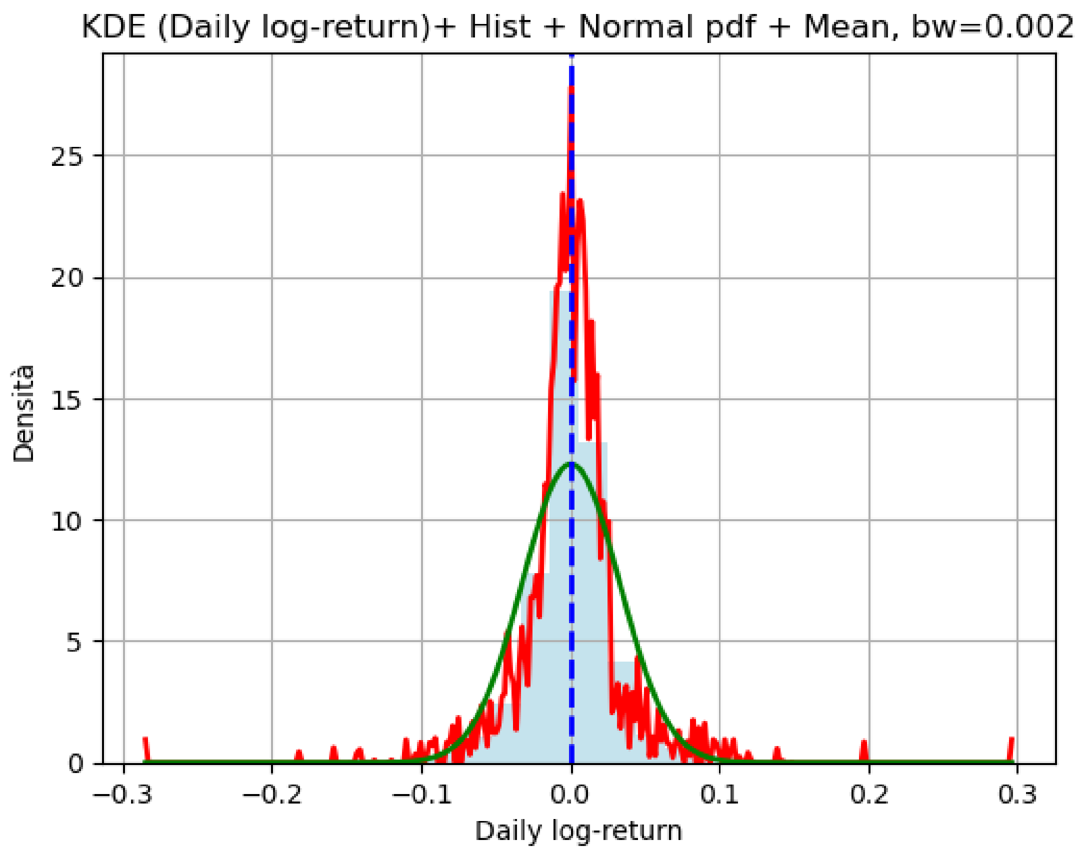
[138]:

```
## Compute the kernel density: daily returns
# divide the interval between the min and max returns into 300 segments
density_eval_points = np.linspace(rt_d_all.min(), rt_d_all.max(), num=300)
# estimate the kernel density of our returns
kde = gaussian_kde(rt_d_all, bw_method=0.002)
# and evaluate in the interval defined above
density_estimation = kde(density_eval_points)
# on the same interval, we evaluate a Normal pdf
pdf_theoretical = norm.pdf(density_eval_points, np.mean(rt_d_all), np.
 ↪std(rt_d_all))

# Plotting
plt.hist(rt_d_all, bins=30, density=True, alpha=0.7, color='lightblue')
plt.plot(density_eval_points, density_estimation, color='red', lw=2,␣
 ↪label='Kernel density')
plt.plot(density_eval_points, pdf_theoretical, color='green', lw=2, label='PDF␣
 ↪Teorica (Normale)')
plt.xlabel("Daily log-return")
plt.ylabel("Densità")
plt.title("KDE (Daily log-return)+ Hist + Normal pdf, bw=0.002")
plt.grid(True)
plt.show()
```



[139]:

```
## Compute the kernel density: daily returns
# divide the interval between the min and max returns into 300 segments
density_eval_points = np.linspace(rt_d_all.min(), rt_d_all.max(), num=300)
# estimate the kernel density of our returns
kde = gaussian_kde(rt_d_all, bw_method=0.002)
# and evaluate in the interval defined above
density_estimation = kde(density_eval_points)
# on the same interval, we evaluate a Normal pdf
pdf_theoretical = norm.pdf(density_eval_points, np.mean(rt_d_all), np.
 ↪std(rt_d_all))
# compute the mean
mean_data = np.mean(rt_d_all)

# Plotting
plt.hist(rt_d_all, bins=30, density=True, alpha=0.7, color='lightblue')
plt.plot(density_eval_points, density_estimation, color='red', lw=2,␣
 ↪label='Kernel density')
plt.plot(density_eval_points, pdf_theoretical, color='green', lw=2, label='PDF␣
 ↪Teorica (Normale)')
plt.axvline(mean_data, color='blue', linestyle='dashed', linewidth=2,␣
 ↪label='Media')

plt.xlabel("Daily log-return")
plt.ylabel("Densità")
plt.title("KDE (Daily log-return)+ Hist + Normal pdf + Mean, bw=0.002")
plt.grid(True)
plt.show()
```



15.1 Summary Statistics

There is a function which compute some summary statistics...not really the ones we want called describe:

[140]:
```
rt_d_all.describe()
```

[140]:
```
count    6475.000000
mean        0.000658
std         0.032465
min        -0.284568
25%        -0.012598
50%         0.000413
75%         0.013969
max         0.296181
Name: rt_d_all, dtype: float64
```

15.1.1 Skewness & Kurtosis

We use the fucntions which came from scipy.stats:

from scipy.stats import gaussian_kde, norm, iqr, skew, kurtosis, jarque_bera, kstest, anderson

These functions replicate the formulas you find on slides.

[141]:
```
rt_d_skew = skew(rt_d_all, nan_policy='omit')
rt_d_kurt = kurtosis(rt_d_all, nan_policy='omit')

print("The skewness is:", rt_d_skew)
print("The kurtosis is:", rt_d_kurt)

# NOTE: There are several formulas to compute skewness and kurtosis.
#       These functions both divide the summations of the estimators by 1/T
```

```
The skewness is: 0.4304836875303971
The kurtosis is: 11.128625063290052
```

Aggregational Kurtosis We compute the kurtosis of the daily, weekly, monthly, and annual returns:

[142]:
```
rt_d_kurt = kurtosis(rt_d_all, nan_policy='omit')
rt_w_kurt = kurtosis(rt_w_all, nan_policy='omit')
rt_m_kurt = kurtosis(rt_m_all, nan_policy='omit')
rt_y_kurt = kurtosis(rt_y_all, nan_policy='omit')

print("Daily: ", round(rt_d_kurt,3))
print("Weekly: ", round(rt_w_kurt,3))
print("Monthly: ", round(rt_m_kurt,3))
print("Annual: ", round(rt_y_kurt,3))
```

```
Daily:  11.129
Weekly:  7.605
Monthly:  2.604
Annual:  1.461
```

15.1.2 Normality Tests

Compute Normality tests and sample summary statistics

Jarque-Bera Test

[143]:
```
JB_rt_d = jarque_bera(rt_d_all)
# first position (0): statistic
print("JB Stat: ", round(JB_rt_d[0],3))
# second position (1): p-value
print("JB p-value: ", JB_rt_d[1])
```

```
JB Stat:  33612.686
JB p-value:  0.0
```

Check the Aggregational Normality:

[144]:
```
print("JB p-value", "daily", "returns:", jarque_bera(rt_d_all)[1])
print("JB p-value", "weekly", "returns:", jarque_bera(rt_w_all)[1])
print("JB p-value", "monthly", "returns:", jarque_bera(rt_m_all)[1])
print("JB p-value", "yearly", "returns:", jarque_bera(rt_y_all)[1])
```

```
JB p-value daily returns: 0.0
JB p-value weekly returns: 0.0
JB p-value monthly returns: 0.0
JB p-value yearly returns: 0.04262486815078237
```

We can also compute the p-value. The JB Stats follows a χ^2_2 distribution. So:

[145]:
```
p_value = 1 - stats.chi2.cdf(STATISTIC, df=2)
print("The associated p-value is:",p_value)
```

```
The associated p-value is: 0.04262486815078237
```

15.1.3 Other normality tests:

Lilliefors test:

[146]:
```
lill_rt_y = lilliefors(rt_y_all)
print("Stat:",lill_rt_y[0])
print("p-val:",lill_rt_y[1])
```

```
Stat: 0.0952201438460884
p-val: 0.7974329823750796
```

Kolmogorov-Smirnov test:

[147]:
```
ks_rt_y = kstest(rt_y_all, 'norm')
print("Stat:",ks_rt_y[0])
print("p-val:",ks_rt_y[1])
```

```
Stat: 0.24100976414208733
p-val: 0.0919870853397472
```

Anderson-Darling test:

[148]:
```
ad_rt_y = anderson(rt_y_all, 'norm')
print("Stat:",ad_rt_y[0])
print("critical val:",ad_rt_y[1])
print("sign level:",ad_rt_y[2])
```

```
Stat: 0.34577306424633036
critical val: [0.514 0.586 0.703 0.82  0.975]
sign level: [15.  10.   5.   2.5  1. ]
```

15.2 Generates table exactly equal to the one in slide n.91

Personalized table of summary statistics.

[149]:
```
# X contains returns at different frequencies
X = {
    'daily': rt_d_all,
    'weekly': rt_w_all,
    'monthly': rt_m_all,
    'annual': rt_y_all
}
```

[150]:
```
def multi_fun(x):
    stat_tab = {
        'Mean': round(np.mean(x) * 100,5),
        'St.Deviation': round(np.std(x) * 100,5),
        'Diameter.C.I.Mean': round(1.96 * np.sqrt(np.var(x) / len(x)) * 100,5),
        'Skewness': round(skew(x),5),
        'Kurtosis': round(kurtosis(x),5),
        'Excess.Kurtosis': round(kurtosis(x) - 3,5),
        'Min': round(np.min(x) * 100,5),
        'Quant5': round(np.quantile(x, 0.05) * 100,5),
        'Quant25': round(np.quantile(x, 0.25) * 100,5),
        'Median': round(np.quantile(x, 0.50) * 100,5),
        'Quant75': round(np.quantile(x, 0.75) * 100,5),
        'Quant95': round(np.quantile(x, 0.95) * 100,5),
        'Max': round(np.max(x) * 100,5),
        'Jarque.Bera.stat': round(jarque_bera(x)[0],5),
        'Jarque.Bera.pvalue.X100': round(jarque_bera(x)[1] *100,5),
        'Lillie.test.stat': round(lilliefors(x)[0],5),
        'Lillie.test.pvalue.X100': round(lilliefors(x)[1] * 100,5),
        'N.obs': len(x)
    }
    return stat_tab
```

1. Define a new dictionary to store the stats:
    a. key will contains the key (i.e., daily, weekly, …)
    b. data will contains the returns
2. Apply *multi_fun* to each data series
3. Define a DataFrame with the stats results
4. Print the dictionary

[151]:
```
# 1.
statistics_dict = {}

# 2.
statistics_dict = {
    key: multi_fun(data.iloc[1:])
```

```
    for key, data in X.items()
}
# apply multi_fun to each returns ("series" in pandas)
# which is located in one of the four key of our dictionary X
# 3.
statistics_df = pd.DataFrame(statistics_dict)

# 4.
print(statistics_df)
```

```
                              daily       weekly     monthly      annual
Mean                        0.06351      0.31014     1.32164    22.85692
St.Deviation                3.24131      6.76830    13.06275    45.28052
Diameter.C.I.Mean           0.07896      0.36213     1.45887    18.11598
Skewness                    0.41992      0.05008    -0.45895    -0.15137
Kurtosis                   11.15158      7.60655     2.59462    -0.64793
Excess.Kurtosis             8.15158      4.60655    -0.40538    -3.64793
Min                       -28.45678    -38.51804   -53.02674   -68.54809
Quant5                     -4.61051     -9.74288   -20.16713   -55.72274
Quant25                    -1.25994     -2.64062    -4.98163    -7.23999
Median                      0.04108      0.30519     2.09626    23.07665
Quant75                     1.39659      3.40897     8.45973    55.96192
Quant95                     4.47118     10.67416    20.90661    94.77653
Max                        29.61811     56.11507    48.35221   102.44636
Jarque.Bera.stat        33735.75720   3235.87866    97.20696     0.51147
Jarque.Bera.pvalue.X100     0.00000      0.00000     0.00000    77.43487
Lillie.test.stat            0.10194      0.09591     0.08194     0.06494
Lillie.test.pvalue.X100     0.10000      0.10000     0.10000    99.00000
N.obs                    6474.00000   1342.00000   308.00000    24.00000
```

Export it as a latex table

[152]:
```
latex_table = statistics_df.to_latex(index=True)
with open("Latex/8stylized.tex", "w") as file:
    file.write(latex_table)
```

```
/var/folders/5r/ft807c7n1ngd3fpt2_gwsg0m0000gn/T/ipykernel_78356/805359341.py:1:
FutureWarning: In future versions `DataFrame.to_latex` is expected to utilise
the base implementation of `Styler.to_latex` for formatting and rendering. The
arguments signature may therefore change. It is recommended instead to use
`DataFrame.style.to_latex` which also contains additional functionality.
  latex_table = statistics_df.to_latex(index=True)
```

[153]:
```
#skewness & kurtosis dict

def skewness_dict(x):
    stat_tab = {
        'Skewness': round(skew(x),5),
        'Kurtosis': round(kurtosis(x),5),
```

```
    }
    return stat_tab
```

[154]:
```
# Y contains returns at different frequencies
Y = {
    'daily': Rt_d_all,
    'weekly': Rt_w_all,
    'monthly': Rt_m_all,
    'annual': Rt_y_all
}
```

Creating a table with isolated skewness and kurtosis

[155]:
```
statistics_dict_sk = {}



statistics_dict_sk_log = {
    key: skewness_dict(data.iloc[1:])
    for key, data in X.items()
}

statistics_dict_sk_simple = {
    key: skewness_dict(data.iloc[1:])
    for key, data in Y.items()
}
#printing results

print("Log returns",pd.DataFrame(statistics_dict_sk_log))
print("Simple returns",pd.DataFrame(statistics_dict_sk_simple))
```

```
Log returns              daily   weekly  monthly   annual
Skewness   0.41992  0.05008 -0.45895 -0.15137
Kurtosis  11.15158  7.60655  2.59462 -0.64793
Simple returns              daily    weekly  monthly   annual
Skewness   1.07310   1.16787  0.40997  0.69139
Kurtosis  13.54753  13.70696  2.93127 -0.30325
```

Export it as a Latex table

[156]:
```
latex_table = pd.DataFrame(statistics_dict_sk_log).to_latex(index=True)
with open("Latex/table_Skewness_kurtosis.tex", "w") as file:
    file.write(latex_table)
```

```
/var/folders/5r/ft807c7n1ngd3fpt2_gwsg0m0000gn/T/ipykernel_78356/3564160307.py:1
: FutureWarning: In future versions `DataFrame.to_latex` is expected to utilise
the base implementation of `Styler.to_latex` for formatting and rendering. The
arguments signature may therefore change. It is recommended instead to use
`DataFrame.style.to_latex` which also contains additional functionality.
```

```
latex_table = pd.DataFrame(statistics_dict_sk_log).to_latex(index=True)
```

[157]:

```
# Compute Box Pierce and Ljung Box tests

for log_returns in [log_returns_daily, log_returns_monthly]:
  my_max_lag = 25
  lags_all = np.arange(1, my_max_lag + 1)
  my_acf = sm.tsa.acf(log_returns, nlags=my_max_lag)
  my_acf_diameter = 1.96 / np.sqrt(len(log_returns))
  my_acf_tstat_0 = (my_acf[1:] - 0) / np.sqrt(1 / len(log_returns))
  my_LjungBox = sm.stats.diagnostic.acorr_ljungbox(log_returns, lags=lags_all,␣
 ↪boxpierce=False)
  my_BoxPierce = sm.stats.diagnostic.acorr_ljungbox(log_returns, lags=lags_all,␣
 ↪boxpierce=True)
  crit_value_5_BP = stats.chi2.ppf(0.95,lags_all)

my_table = np.column_stack((
    lags_all,
    my_acf[1:],
    np.full(my_max_lag, my_acf_diameter),
    my_acf_tstat_0,
    my_BoxPierce['bp_stat'],
    my_BoxPierce['bp_pvalue'],
    my_LjungBox['lb_stat'],
    my_LjungBox['lb_pvalue'],
    np.full(my_max_lag, crit_value_5_BP)
))
column_names = ["lag", "acf", "acf diam.", "acf test", "B-P stat", "B-P pval",␣
 ↪"L-B stat", "L-B pval", "crit"]
my_table_df = pd.DataFrame(data=my_table, columns=column_names)

# Reducing the selection of lags
my_table_df = my_table_df.iloc[[0,4,14,24]]

# Print the rounded table
my_table_df = my_table_df.round(3)
my_table_df['lag'] = my_table_df['lag'].astype(int)
print(my_table_df)
```

```
    lag    acf  acf diam.  acf test  B-P stat  B-P pval  L-B stat  L-B pval  \
0     1  0.053      0.112     0.930     0.865     0.352     0.874     0.350
4     5  0.021      0.112     0.376     7.654     0.176     7.796     0.168
14   15 -0.049      0.112    -0.865    22.975     0.085    23.673     0.071
24   25 -0.048      0.112    -0.846    36.101     0.070    37.859     0.048

      crit
0    3.841
```

```
4   11.070
14  24.996
24  37.652
```

Export it as a Latex Table

[158]:
```
latex_table = my_table_df.to_latex(index=True)
with open("Latex/LB_BP.tex", "w") as file:
    file.write(latex_table)
```

```
/var/folders/5r/ft807c7n1ngd3fpt2_gwsg0m0000gn/T/ipykernel_78356/4088390180.py:1
: FutureWarning: In future versions `DataFrame.to_latex` is expected to utilise
the base implementation of `Styler.to_latex` for formatting and rendering. The
arguments signature may therefore change. It is recommended instead to use
`DataFrame.style.to_latex` which also contains additional functionality.
  latex_table = my_table_df.to_latex(index=True)
```