

## Question 3 Antartica

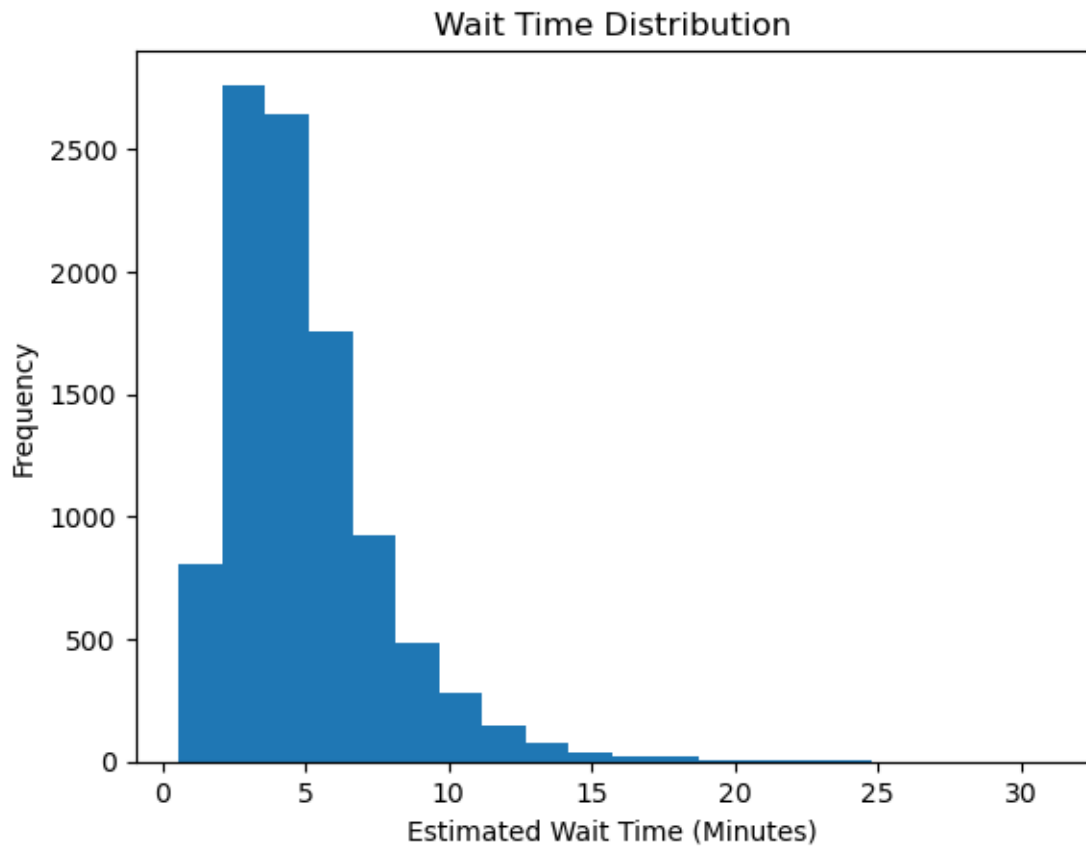
May 23, 2023

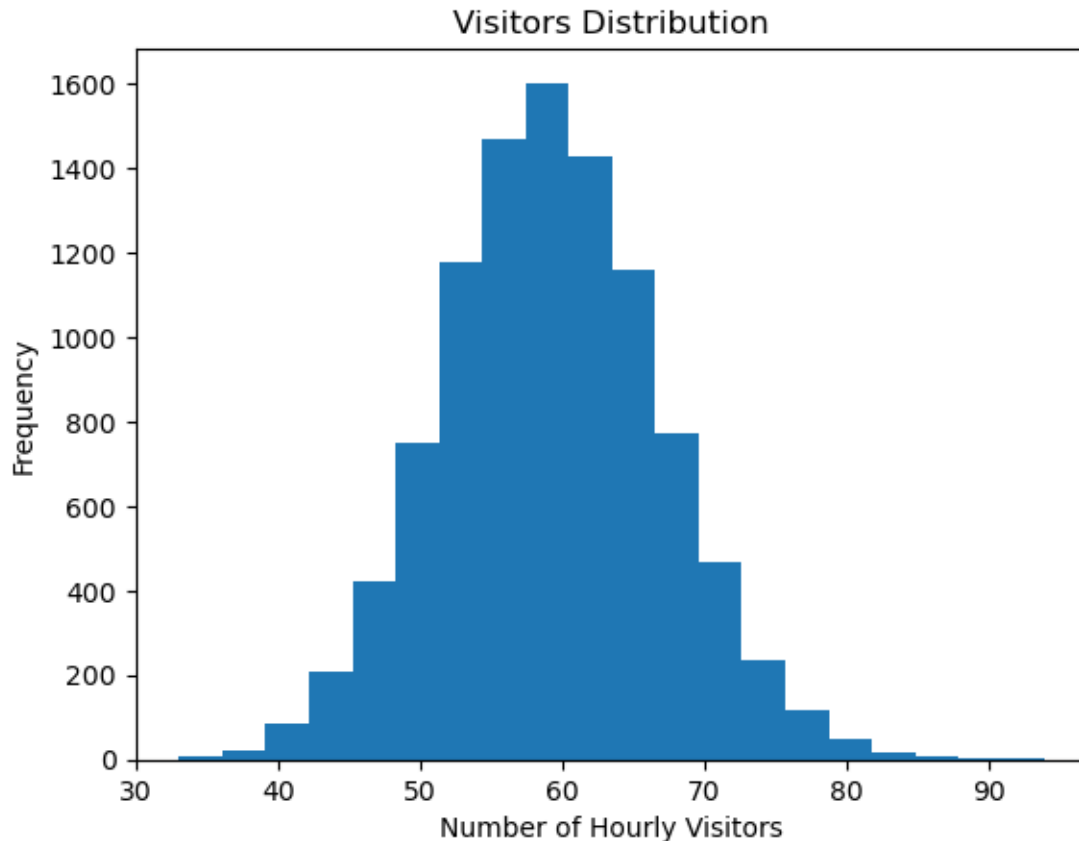
```
[1]: import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats
import statsmodels.api as sm

# Load data
df = pd.read_excel('data.xlsx', sheet_name='website data')

# plot histogram of estimated wait time
plt.hist(df['estimated wait time (mins)'], bins=20)
plt.title('Wait Time Distribution')
plt.xlabel('Estimated Wait Time (Minutes)')
plt.ylabel('Frequency')
plt.show()

# plot histogram of hourly visitors
plt.hist(df['number of visitors'], bins=20)
plt.title('Visitors Distribution')
plt.xlabel('Number of Hourly Visitors')
plt.ylabel('Frequency')
plt.show()
```





The data shows estimated wait times ranging from 0 to 31 minutes and number of hourly visitors ranging from 33 to 94.

Looking at the histograms, we can derive the following information:

- **Estimated Wait Time Distribution:** The histogram of estimated wait time is skewed to the right, with a majority of values between 0 and 10 minutes and a long tail of higher wait times. This type of distribution is often observed in real-world scenarios where the wait time cannot be negative but can have extreme values. Based on this, we can conclude that the estimated wait time follows a positively skewed distribution such as a Weibull or a Lognormal distribution.
- **Number of Hourly Visitors Distribution:** The histogram of the number of hourly visitors is skewed to the right, with a majority of values between 0 and 100 visitors and a long tail of higher visitor counts. This type of distribution is often observed in real-world scenarios where the values cannot be negative but can have extreme values. Based on this, we can conclude that the number of hourly visitors follows a positively skewed distribution such as a Negative Binomial distribution.

```
[2]: # check if normally distributed
      # perform Shapiro-Wilk test for Estimated Wait Time
      stat, p = stats.shapiro(df['estimated wait time (mins)'])
```

```

print('Shapiro-Wilk test for Estimated Wait Time:')
print('stat=%.3f, p=%.2E' % (stat, p))
alpha = 0.05
if p > alpha:
    print('Estimated Wait Time is normally distributed (fail to reject H0)')
else:
    print('Estimated Wait Time is not normally distributed (reject H0)')

# perform Kolmogorov-Smirnov test for Estimated Wait Time
stat, p = stats.kstest(df['estimated wait time (mins)'], 'norm')
print('Kolmogorov-Smirnov test for Estimated Wait Time:')
print('stat=%.3f, p=%.2E' % (stat, p))
alpha = 0.05
if p > alpha:
    print('Estimated Wait Time is normally distributed (fail to reject H0)')
else:
    print('Estimated Wait Time is not normally distributed (reject H0)')

# perform Shapiro-Wilk test for number of visitors
stat, p = stats.shapiro(df['number of visitors'])
print('Shapiro-Wilk test for number of visitors:')
print('stat=%.3f, p=%.2E' % (stat, p))
alpha = 0.05
if p > alpha:
    print('number of visitors is normally distributed (fail to reject H0)')
else:
    print('number of visitors is not normally distributed (reject H0)')

# perform Kolmogorov-Smirnov test for number of visitors
stat, p = stats.kstest(df['number of visitors'], 'norm')
print('Kolmogorov-Smirnov test for number of visitors:')
print('stat=%.3f, p=%.2E' % (stat, p))
alpha = 0.05
if p > alpha:
    print('number of visitors is normally distributed (fail to reject H0)')
else:
    print('number of visitors is not normally distributed (reject H0)')

```

Shapiro-Wilk test for Estimated Wait Time:  
stat=0.871, p=0.00E+00  
Estimated Wait Time is not normally distributed (reject H0)  
Kolmogorov-Smirnov test for Estimated Wait Time:  
stat=0.920, p=0.00E+00  
Estimated Wait Time is not normally distributed (reject H0)  
Shapiro-Wilk test for number of visitors:  
stat=0.997, p=1.43E-12  
number of visitors is not normally distributed (reject H0)

```

Kolmogorov-Smirnov test for number of visitors:
stat=1.000, p=0.00E+00
number of visitors is not normally distributed (reject H0)

/Applications/anaconda3/lib/python3.9/site-
packages/scipy/stats/_morestats.py:1800: UserWarning: p-value may not be
accurate for N > 5000.
  warnings.warn("p-value may not be accurate for N > 5000.")

```

```

[3]: from scipy.optimize import minimize

# Define the negative log-likelihood function for the Negative Binomial
↪distribution
def neg_log_likelihood(params, x):
    r = params[0]
    p = params[1]
    return -np.sum(stats.nbinom.logpmf(x, r, p))

# Fit Negative Binomial distribution to the number of visitors using MLE
x = df['number of visitors']
params_init = [1, 0.5] # Initial parameter values
result = minimize(neg_log_likelihood, params_init, args=(x,),
↪method='Nelder-Mead')
params_negbinom = result.x

print('Negative Binomial Distribution parameters:')
print('r:', params_negbinom[0])
print('p:', params_negbinom[1])

# Perform chi-squared test for negative log-likelihood
nll_observed = -neg_log_likelihood(params_negbinom, x)
nll_expected = -np.sum(stats.nbinom.logpmf(x, *params_negbinom))
chi2_nll = 2 * (nll_observed - nll_expected)
p_nll = 1 - stats.chi2.cdf(chi2_nll, df=len(x) - len(params_negbinom))

print('Chi-squared test for Negative Log-Likelihood:')
print('Chi-square statistic =', chi2_nll)
print('p-value =', p_nll)

```

```

Negative Binomial Distribution parameters:
r: 869.1745263744751
p: 0.9361144660647907
Chi-squared test for Negative Log-Likelihood:
Chi-square statistic = -137899.48781361678
p-value = 1.0

```

```

[4]: # Generate theoretical quantiles for the Negative Binomial distribution

```

```

theoretical_quantiles = stats.nbinom.ppf(np.linspace(0.01, 0.99, len(df)),
    ↪params_negbinom[0], params_negbinom[1])

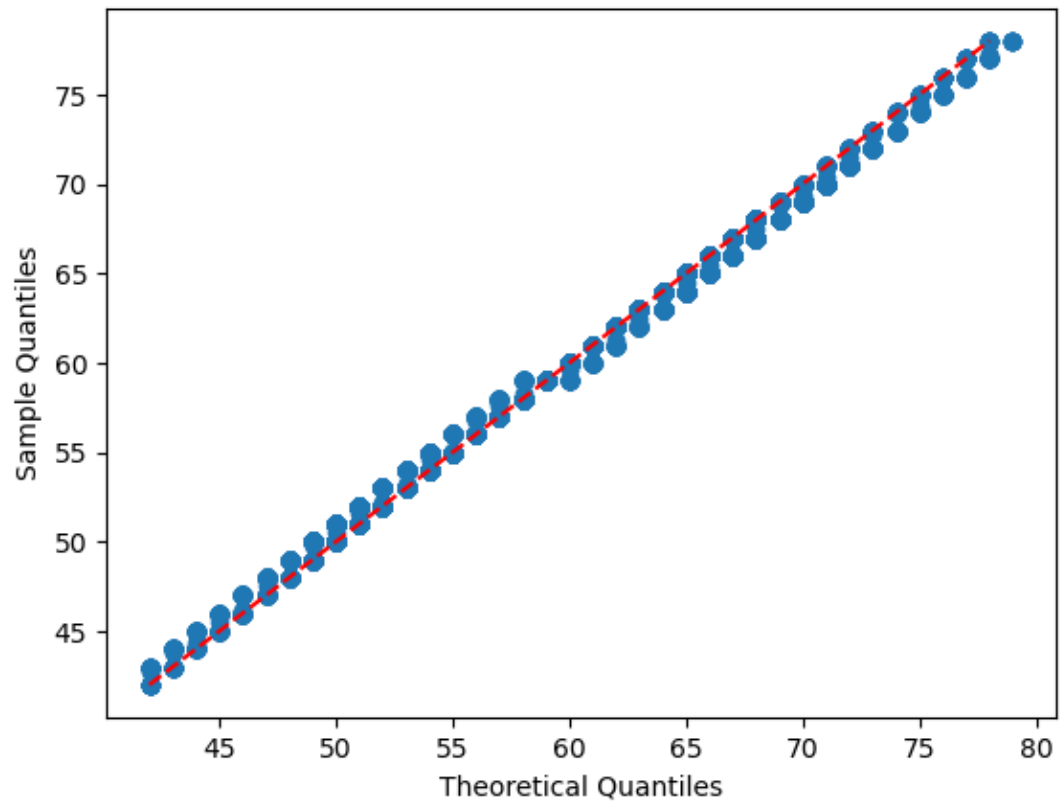
# Generate sample quantiles from the observed data
sample_quantiles = np.percentile(df['number of visitors'], np.linspace(1, 99,
    ↪len(df)))

# Plot the QQ-plot
plt.scatter(theoretical_quantiles, sample_quantiles)
plt.plot([np.min(sample_quantiles), np.max(sample_quantiles)], [np.
    ↪min(sample_quantiles), np.max(sample_quantiles)], color='red',
    ↪linestyle='--')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Sample Quantiles')
plt.title("QQ-plot for Negative Binomial Distribution of 'Number of Visitors'")
plt.show()

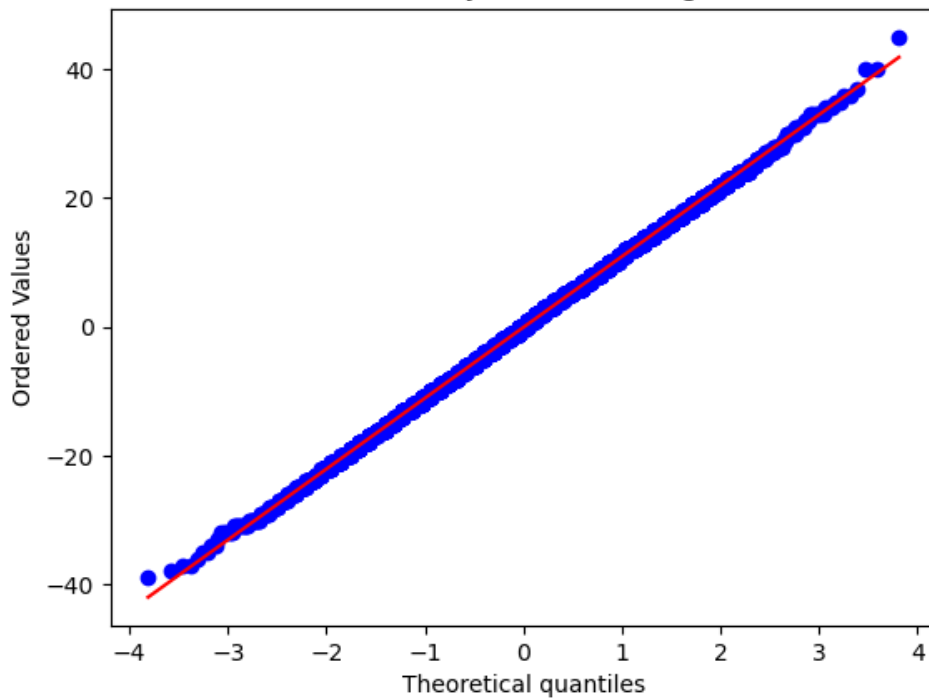
# Residuals for the Negative Binomial distribution
NB_fit = stats.nbinom(params_negbinom[0], params_negbinom[1])
residuals = df['number of visitors'] - NB_fit.rvs(len(df))
fig, ax = plt.subplots()
_ = stats.probplot(residuals, dist=stats.norm, plot=ax)
ax.set_title("Residuals Plot for 'Number of Hourly Visitors' in Negative
    ↪Binomial Distribution")
plt.show()

```

QQ-plot for Negative Binomial Distribution of 'Number of Visitors'



Residuals Plot for 'Number of Hourly Visitors' in Negative Binomial Distribution



```
[5]: wait_time_data = df['estimated wait time (mins)']

# Fit Gamma distribution
params_gamma = stats.gamma.fit(wait_time_data)

# Fit Weibull distribution
params_weibull = stats.weibull_min.fit(wait_time_data, floc=0)

# Fit Lognormal distribution
params_lognormal = stats.lognorm.fit(wait_time_data)

# Create a histogram of the estimated wait time data
plt.hist(wait_time_data, bins=35, density=True, alpha=0.5, label='Data')

# Generate the x-axis values for the PDF plots
x = np.linspace(wait_time_data.min(), wait_time_data.max(), 100)

# Plot the PDFs of the fitted distributions
plt.plot(x, stats.gamma.pdf(x, *params_gamma), label='Gamma')
plt.plot(x, stats.weibull_min.pdf(x, *params_weibull), label='Weibull')
plt.plot(x, stats.lognorm.pdf(x, *params_lognormal), label='Lognormal')

plt.xlabel('Estimated Wait Time (mins)')
```



```

plt.ylabel('Probability Density')
plt.title('Fitted Distributions for Estimated Wait Time')
plt.legend()
plt.show()

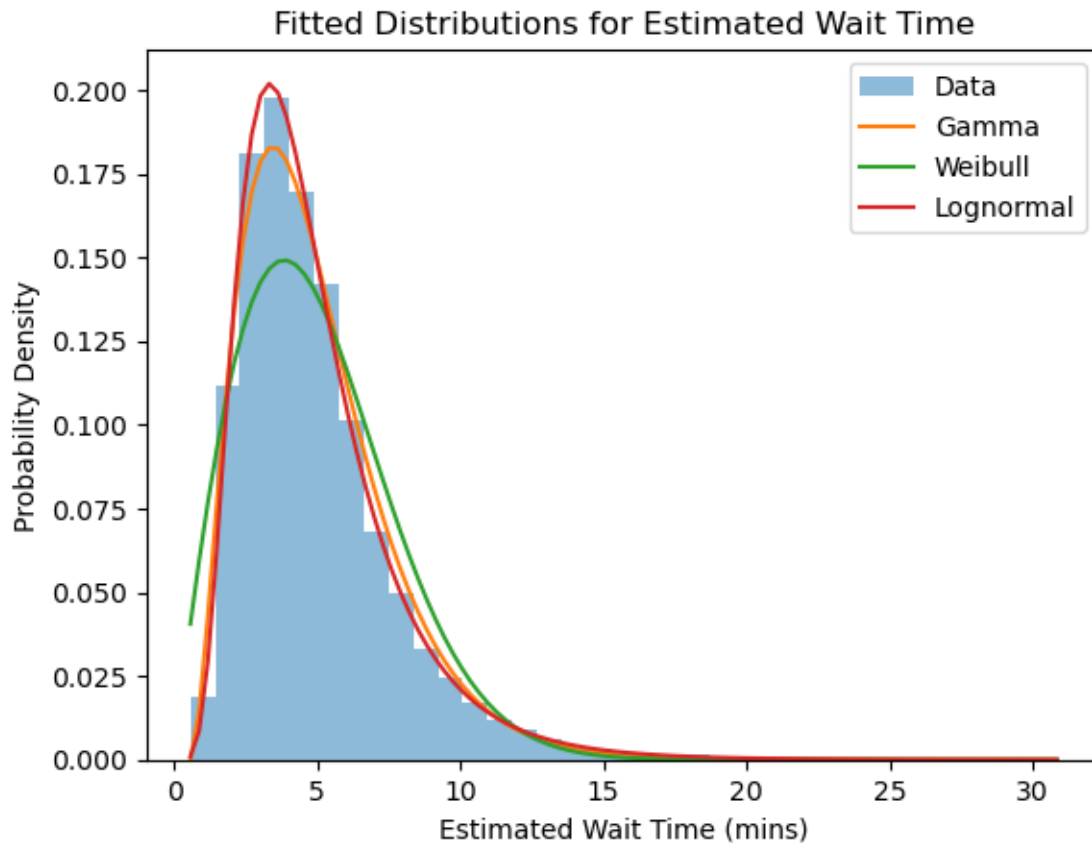
# Perform Kolmogorov-Smirnov test for Gamma distribution
ks_stat_gamma, p_gamma = stats.kstest(wait_time_data, 'gamma',
    ↪args=params_gamma)

# Perform Kolmogorov-Smirnov test for Weibull distribution
ks_stat_weibull, p_weibull = stats.kstest(wait_time_data, 'weibull_min',
    ↪args=params_weibull)

# Perform Kolmogorov-Smirnov test for Lognormal distribution
ks_stat_lognormal, p_lognormal = stats.kstest(wait_time_data, 'lognorm',
    ↪args=params_lognormal)

print('Kolmogorov-Smirnov test results:')
print('Gamma distribution: KS statistic =', ks_stat_gamma, ', p-value =',
    ↪p_gamma)
print('Weibull distribution: KS statistic =', ks_stat_weibull, ', p-value =',
    ↪p_weibull)
print('Lognormal distribution: KS statistic =', ks_stat_lognormal, ', p-value',
    ↪=, p_lognormal)

```



Kolmogorov-Smirnov test results:

Gamma distribution: KS statistic = 0.02287097777524083 , p-value = 5.630547033185526e-05

Weibull distribution: KS statistic = 0.06116701503400005 , p-value = 5.748174729437654e-33

Lognormal distribution: KS statistic = 0.006071646969163624 , p-value = 0.8523915218713285

```
[6]: # Plot histogram and PDF for the estimated wait time data
plt.hist(wait_time_data, bins=35, density=True, alpha=0.5, label='Estimated_
↳Wait Time')
x = np.linspace(wait_time_data.min(), wait_time_data.max(), 100)
pdf_lognormal = stats.lognorm.pdf(x, *params_lognormal)
plt.plot(x, pdf_lognormal, 'r', label='Lognormal Distribution')
plt.xlabel('Estimated Wait Time')
plt.ylabel('Density')
plt.title('Histogram and Lognormal Distribution for Estimated Wait Time')
plt.legend()
plt.show()
```

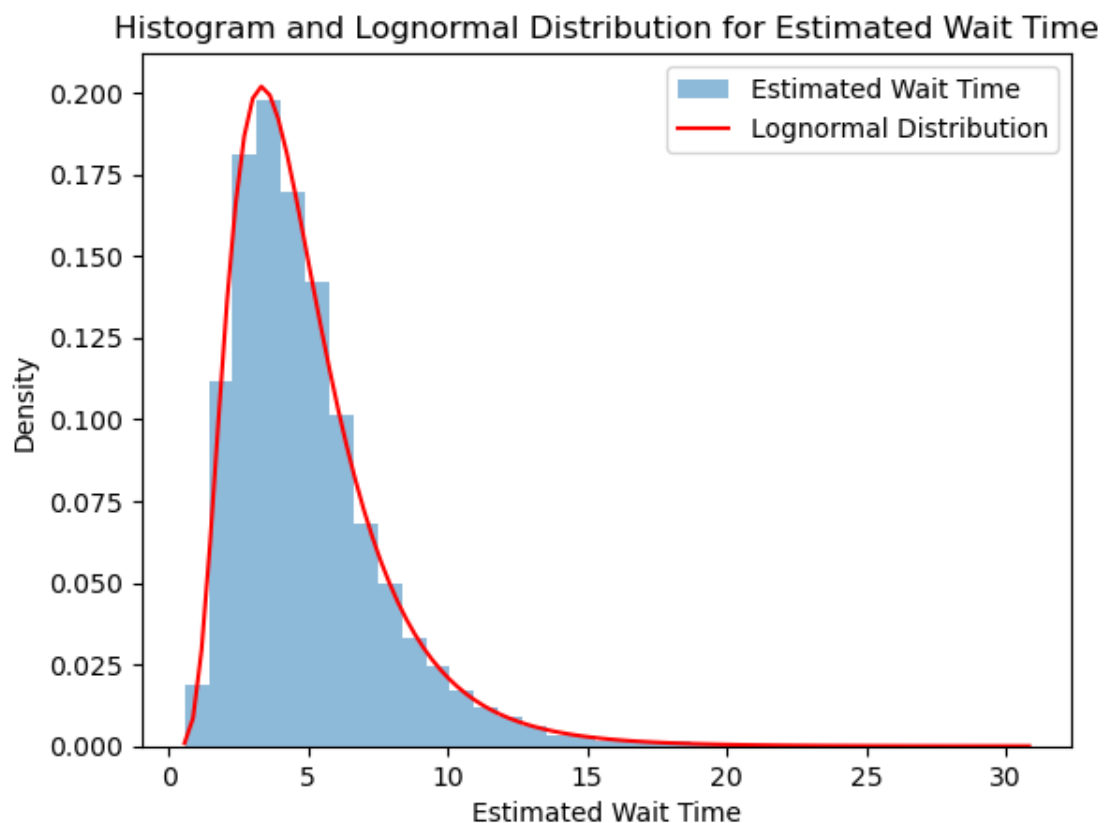
```

# Set the number of bins
num_bins = 35

observed, bin_edges = np.histogram(wait_time_data, bins=num_bins)
expected = stats.lognorm.pdf((bin_edges[:-1] + bin_edges[1:]) / 2,
    ↳*params_lognormal) * len(wait_time_data) * np.diff(bin_edges)
expected = expected * (np.sum(observed) / np.sum(expected))
chi2, p = stats.chisquare(observed, f_exp=expected)

print('Chi-squared goodness-of-fit test for Lognormal distribution:')
print('Chi-square statistic =', chi2)
print('p-value =', p)

```



Chi-squared goodness-of-fit test for Lognormal distribution:  
 Chi-square statistic = 36.32363801465795  
 p-value = 0.36087681875979577

```

[7]: import numpy as np
import scipy.stats as stats

```

```

# Experimenting with different binning strategies
num_bins_strategy1 = 30
num_bins_strategy2 = 33
num_bins_strategy3 = 35

# Calculate observed frequencies for each binning strategy
observed_strategy1, bin_edges_strategy1 = np.histogram(wait_time_data,
    ↪bins=num_bins_strategy1)
observed_strategy2, bin_edges_strategy2 = np.histogram(wait_time_data,
    ↪bins=num_bins_strategy2)
observed_strategy3, bin_edges_strategy3 = np.histogram(wait_time_data,
    ↪bins=num_bins_strategy3)

# Calculate expected frequencies for each binning strategy using the Lognormal
    ↪distribution parameters
# Fit Lognormal distribution
params_lognormal = stats.lognorm.fit(wait_time_data)
params_lognormal = (params_lognormal[0], params_lognormal[1],
    ↪params_lognormal[2])

expected_strategy1 = stats.lognorm.pdf((bin_edges_strategy1[:-1] +
    ↪bin_edges_strategy1[1:]) / 2, *params_lognormal) * len(wait_time_data) * np.
    ↪diff(bin_edges_strategy1)
expected_strategy2 = stats.lognorm.pdf((bin_edges_strategy2[:-1] +
    ↪bin_edges_strategy2[1:]) / 2, *params_lognormal) * len(wait_time_data) * np.
    ↪diff(bin_edges_strategy2)
expected_strategy3 = stats.lognorm.pdf((bin_edges_strategy3[:-1] +
    ↪bin_edges_strategy3[1:]) / 2, *params_lognormal) * len(wait_time_data) * np.
    ↪diff(bin_edges_strategy3)

# Normalize the expected frequencies
expected_strategy1 = expected_strategy1 * np.sum(observed_strategy1) / np.
    ↪sum(expected_strategy1)
expected_strategy2 = expected_strategy2 * np.sum(observed_strategy2) / np.
    ↪sum(expected_strategy2)
expected_strategy3 = expected_strategy3 * np.sum(observed_strategy3) / np.
    ↪sum(expected_strategy3)

# Perform chi-squared goodness-of-fit test for each binning strategy
chi2_strategy1, p_strategy1 = stats.chisquare(observed_strategy1,
    ↪f_exp=expected_strategy1)
chi2_strategy2, p_strategy2 = stats.chisquare(observed_strategy2,
    ↪f_exp=expected_strategy2)
chi2_strategy3, p_strategy3 = stats.chisquare(observed_strategy3,
    ↪f_exp=expected_strategy3)

```

```

# Print the p-values for each binning strategy
print('Binning Strategy 30 bins:')
print('p-value =', p_strategy1)
print('Binning Strategy 33 bins:')
print('p-value =', p_strategy2)
print('Binning Strategy 35 bins:')
print('p-value =', p_strategy3)

```

```

Binning Strategy 30 bins:
p-value = 0.015372757151227972
Binning Strategy 33 bins:
p-value = 0.2892843937010699
Binning Strategy 35 bins:
p-value = 0.36087681875979644

```

```

[8]: import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

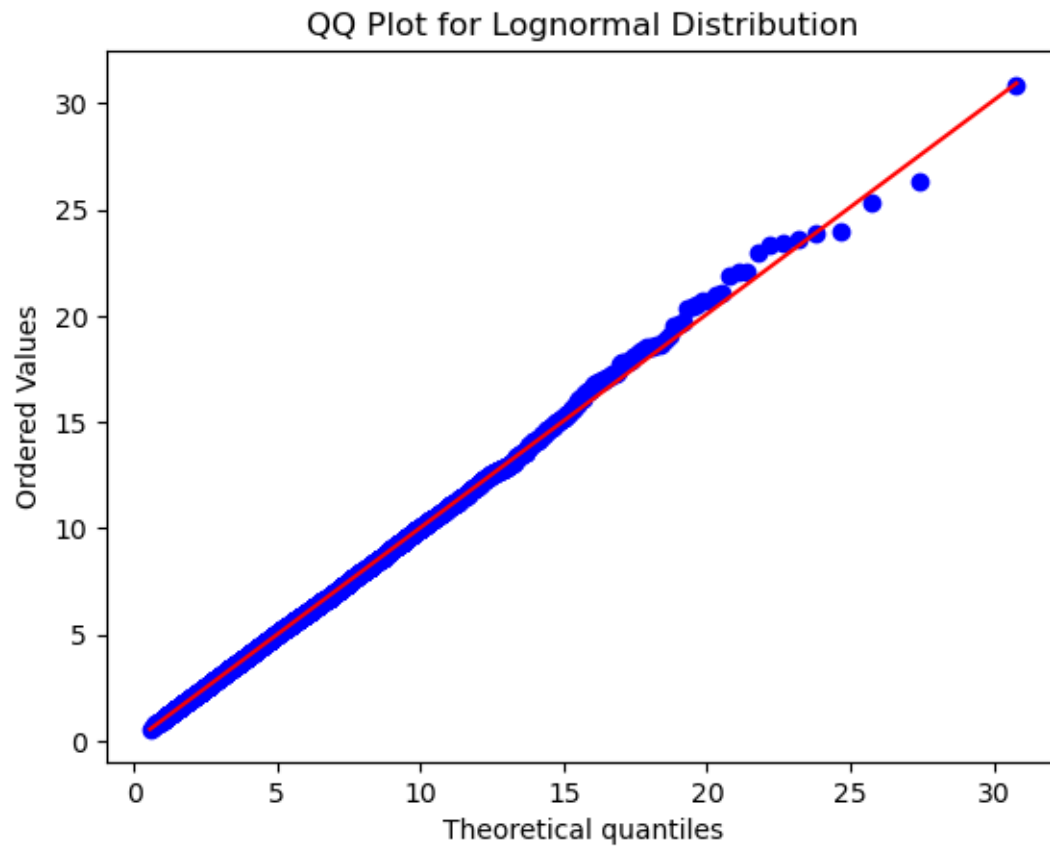
# Fit Lognormal distribution
params_lognormal = stats.lognorm.fit(wait_time_data)
dist_lognormal = stats.lognorm(*params_lognormal)

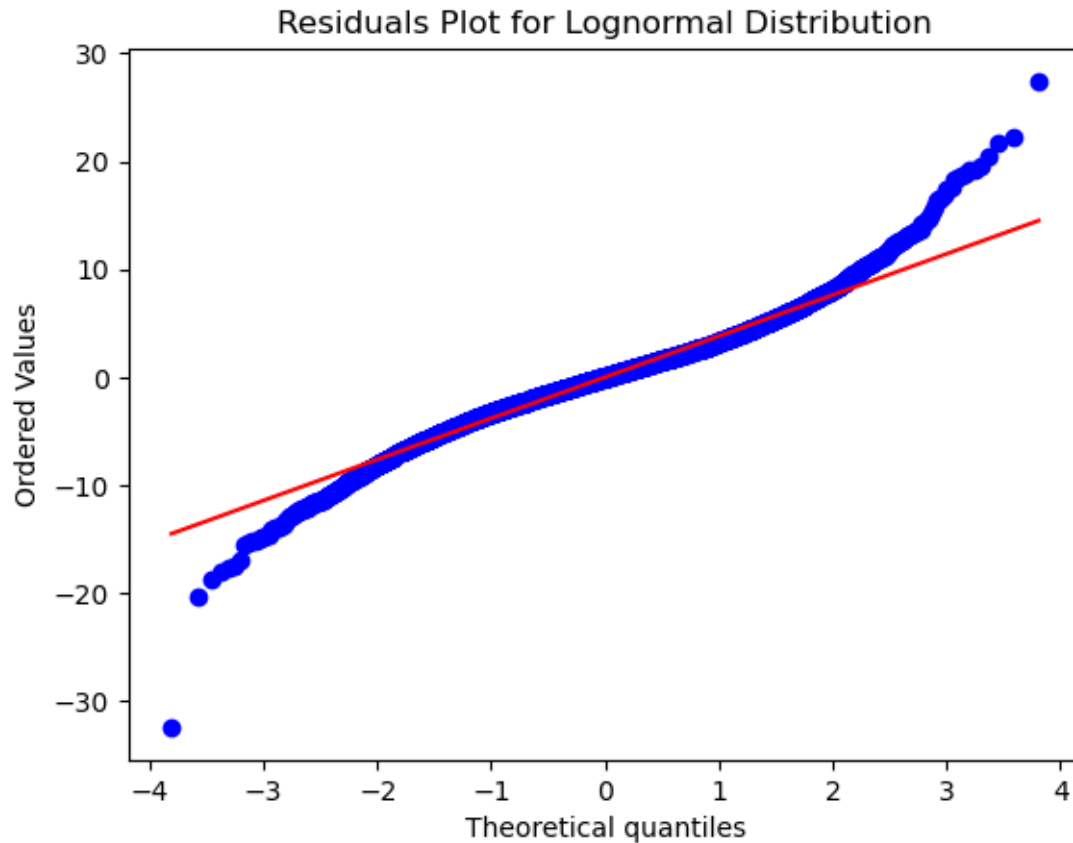
# Generate QQ plot
fig, ax = plt.subplots()
stats.probplot(wait_time_data, dist=dist_lognormal, plot=ax)
ax.set_title("QQ Plot for Lognormal Distribution")
plt.show()

# Calculate residuals
residuals = wait_time_data - dist_lognormal.rvs(len(wait_time_data))

# Plot residuals
fig, ax = plt.subplots()
stats.probplot(residuals, dist=stats.norm, plot=ax)
ax.set_title("Residuals Plot for Lognormal Distribution")
plt.show()

```



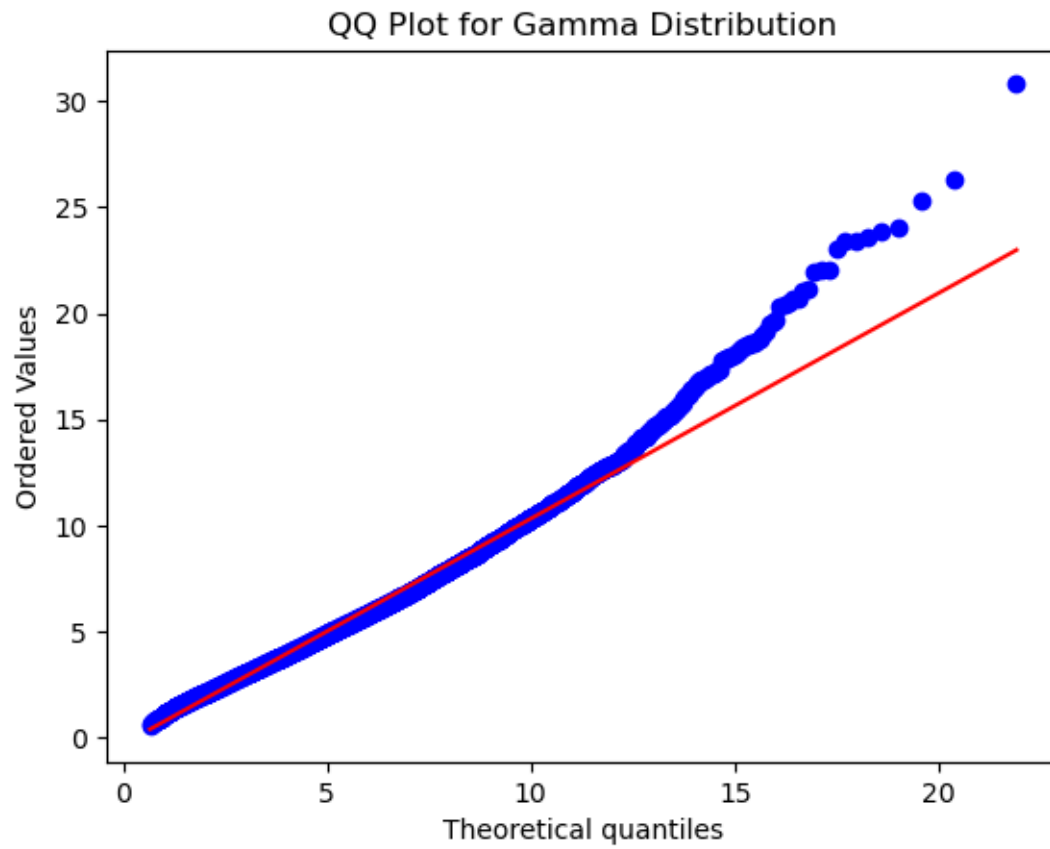


```
[9]: # Quick check for Gamma distribution
params_gamma = stats.gamma.fit(wait_time_data)
dist_gamma = stats.gamma(*params_gamma)

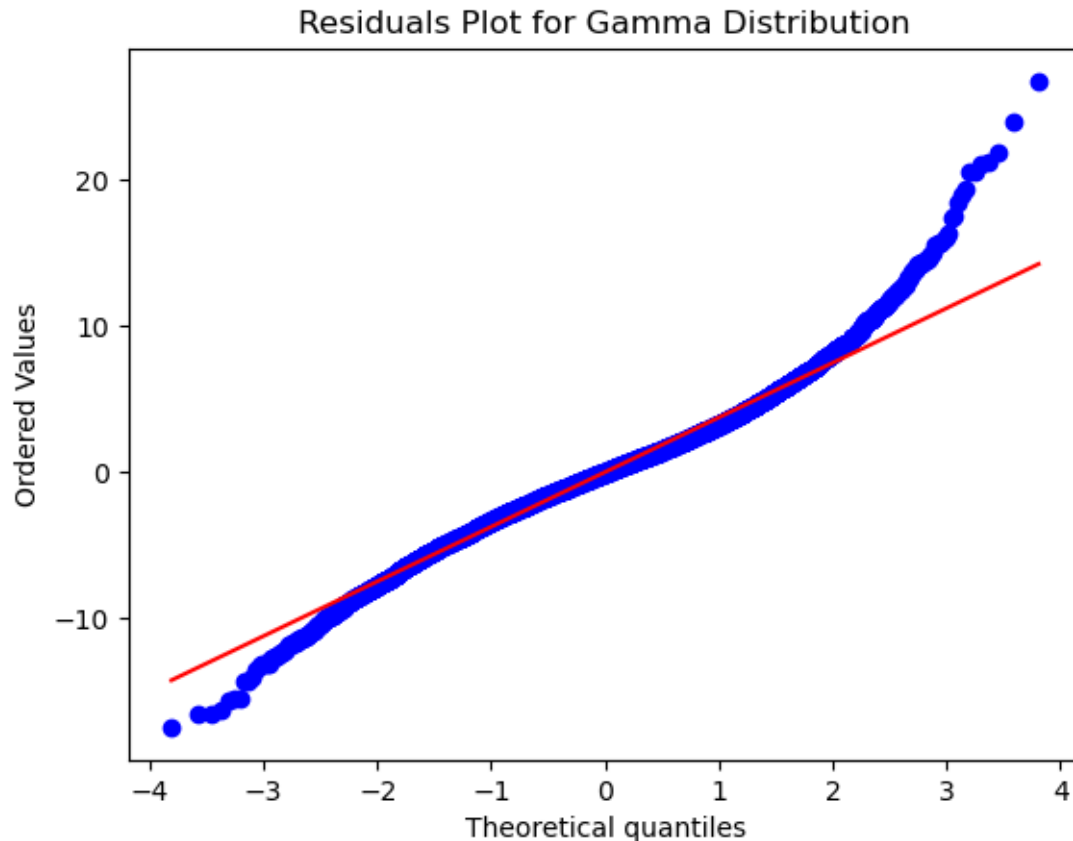
# Generate QQ plot
fig, ax = plt.subplots()
stats.probplot(wait_time_data, dist=dist_gamma, plot=ax)
ax.set_title("QQ Plot for Gamma Distribution")
plt.show()

# Calculate residuals
residuals = wait_time_data - dist_gamma.rvs(len(wait_time_data))

# Plot residuals
fig, ax = plt.subplots()
stats.probplot(residuals, dist=stats.norm, plot=ax)
ax.set_title("Residuals Plot for Gamma Distribution")
plt.show()
```







It seems that the lognormal distribution is a reasonable fit for the majority of the data, as evidenced by the residuals plot being generally aligned with the zero line. However, the heavier tails observed in the residuals plot indicate that the lognormal distribution may not capture the extreme values adequately.

Since the histogram of your data resembles a good fit with the lognormal distribution, it suggests that the lognormal distribution captures the central portion of the data well. However, the deviations in the tails suggest that the lognormal distribution may underestimate the occurrence of extreme values.

In terms of the chi-squared tests, obtaining good p-values for bin values above 30 indicates that the lognormal distribution provides a reasonable fit to the data when using larger bin sizes. This suggests that the lognormal distribution approximates the observed frequencies well in those bins.

Considering all these findings, it seems that the lognormal distribution is a reasonable choice for modeling the majority of the data.

```
[10]: # part 3.2

prob_wait_time_gt_10 = 1 - stats.lognorm.cdf(10, *params_lognormal)
prob_visitors_lt_46 = stats.nbinom.cdf(45, *params_negbinom)
```

```

print("Probability that the estimated wait time is greater than 10 minutes:",
      prob_wait_time_gt_10)
print("Probability that the number of visitors is less than 46 minutes:",
      prob_visitors_lt_46)

```

Probability that the estimated wait time is greater than 10 minutes:  
0.053006985511866  
Probability that the number of visitors is less than 46 minutes:  
0.03667529281022168

```

[11]: import statsmodels.api as sm
num_visitors_data = df['number of visitors']
# Assuming you have data in a DataFrame with 'wait_time' as the dependent
# variable and 'num_visitors' as the independent variable
data = pd.DataFrame({'wait_time': wait_time_data, 'num_visitors':
                     num_visitors_data})

# Add a constant term to the independent variable
data = sm.add_constant(data)

# Fit the linear regression model
model = sm.OLS(data['wait_time'], data[['const', 'num_visitors']])
results = model.fit()

# Print the regression results
print(results.summary())

```

OLS Regression Results						
Dep. Variable:	wait_time	R-squared:	0.479			
Model:	OLS	Adj. R-squared:	0.479			
Method:	Least Squares	F-statistic:	9186.			
Date:	Tue, 23 May 2023	Prob (F-statistic):	0.00			
Time:	09:52:15	Log-Likelihood:	-21059.			
No. Observations:	10000	AIC:	4.212e+04			
Df Residuals:	9998	BIC:	4.214e+04			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	-9.9074	0.156	-63.357	0.000	-10.214	-9.601
num_visitors	0.2507	0.003	95.846	0.000	0.246	0.256
Omnibus:	3510.787	Durbin-Watson:	2.032			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	21979.726			
Skew:	1.545	Prob(JB):	0.00			

## Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

The OLS regression summary provides important information about the regression model. Here is an analysis of the results:

- R-squared: The R-squared value of 0.479 indicates that approximately 47.9% of the variation in the dependent variable (wait\_time) can be explained by the independent variable (num\_visitors). This suggests a moderate level of explanatory power.
- F-statistic: The F-statistic of 9186.0 with a very low p-value indicates that the overall regression model is statistically significant. This suggests that at least one of the independent variables is significantly related to the dependent variable.
- Coefficients: The coefficient of the constant term (-9.9074) indicates the expected value of the dependent variable (wait\_time) when the independent variable (num\_visitors) is zero. The coefficient of the num\_visitors variable (0.2507) indicates that, on average, for each additional unit increase in num\_visitors, the wait\_time is expected to increase by 0.2507 units.
- Standard Errors: The standard errors estimate the variability of the coefficients. Smaller standard errors indicate more precise estimates.
- t-values and p-values: The t-values measure the number of standard errors the coefficients are away from zero. The p-values indicate the statistical significance of the coefficients. In this case, both the constant and num\_visitors coefficients are highly statistically significant ( $p < 0.05$ ).
- Confidence Intervals: The [0.025, 0.975] interval provides a range of values within which we can be 95% confident that the true population coefficients lie.
- Omnibus and Jarque-Bera tests: These tests assess the normality assumption of the residuals. The low p-values indicate that the residuals are not normally distributed.
- Durbin-Watson statistic: The value of 2.032 suggests that there may be some positive autocorrelation present in the residuals.

Overall, the regression model shows a significant relationship between the number of visitors and the wait time. However, further analysis is required to address the issues of non-normality of residuals and potential autocorrelation.

```
[12]: # Diagnostic checks

X = data[['const', 'num_visitors']] # Independent variable(s)
y = data['wait_time'] # Dependent variable

# Diagnostic checks

# Multicollinearity check (Variance Inflation Factor)
```

```

from statsmodels.stats.outliers_influence import variance_inflation_factor

vif = pd.DataFrame()
vif["Variable"] = X.columns
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(vif)

# Normality of residuals
residuals = results.resid

# Histogram of residuals
plt.hist(residuals, bins=30)
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Histogram of Residuals')
plt.show()

# Q-Q plot of residuals
sm.qqplot(residuals, line='45')
plt.title('Q-Q Plot of Residuals')
plt.show()

# Homoscedasticity (Residuals vs. Predicted Values)
plt.scatter(results.predict(), residuals)
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residuals vs. Predicted Values')
plt.axhline(0, color='red', linestyle='--')
plt.show()

# Linearity (Observed vs. Predicted Values)
plt.scatter(results.predict(), y)
plt.xlabel('Predicted Values')
plt.ylabel('Observed Values')
plt.title('Observed vs. Predicted Values')
plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red', linestyle='--')
plt.show()

# Outliers and Influential Observations
from statsmodels.stats.outliers_influence import OLSInfluence

# Leverage values
leverage = OLSInfluence(results).influence
plt.stem(leverage, markerfmt='.')
plt.xlabel('Observation')
plt.ylabel('Leverage')
plt.title('Leverage Plot')

```

```

plt.show()

# Studentized residuals
student_resid = OLSInfluence(results).resid_studentized_internal
plt.scatter(results.predict(), student_resid)
plt.xlabel('Predicted Values')
plt.ylabel('Studentized Residuals')
plt.title('Studentized Residuals vs. Predicted Values')
plt.axhline(0, color='red', linestyle='--')
plt.show()

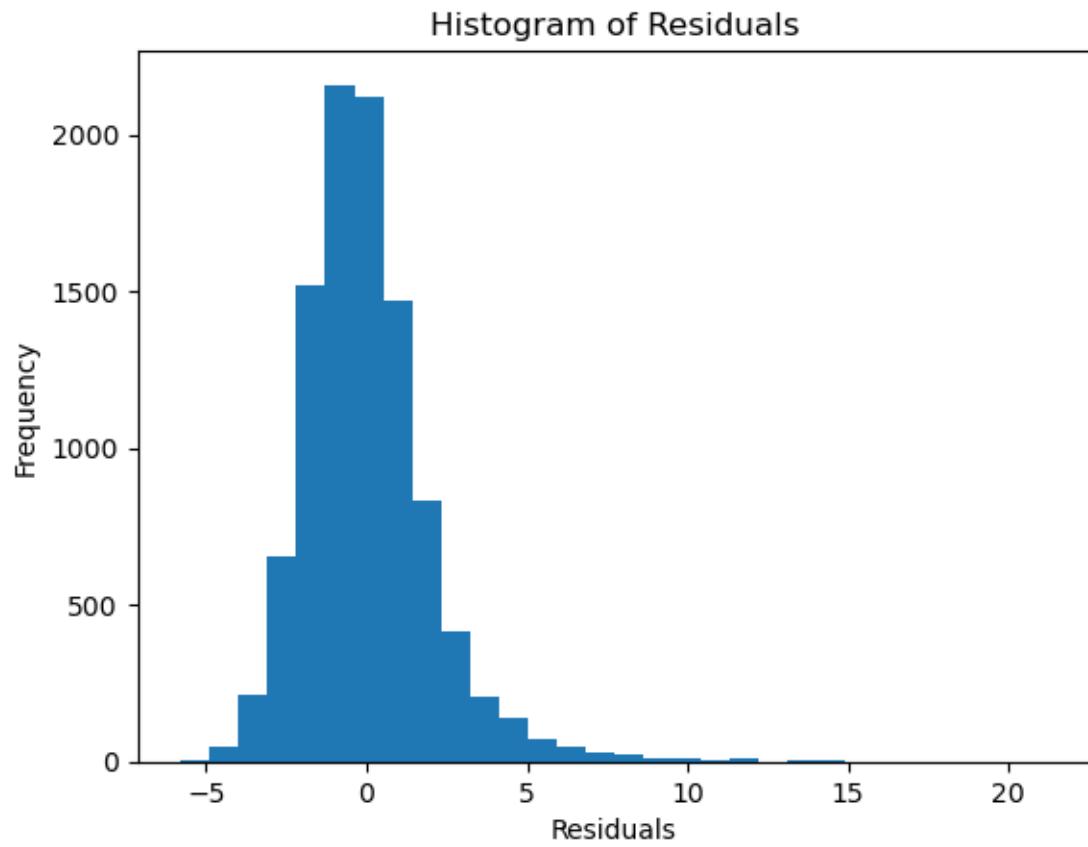
# Cook's distance
cook_distance = OLSInfluence(results).cooks_distance[0]
plt.stem(cook_distance, markerfmt='.')
plt.xlabel('Observation')
plt.ylabel("Cook's Distance")
plt.title("Cook's Distance Plot")
plt.show()

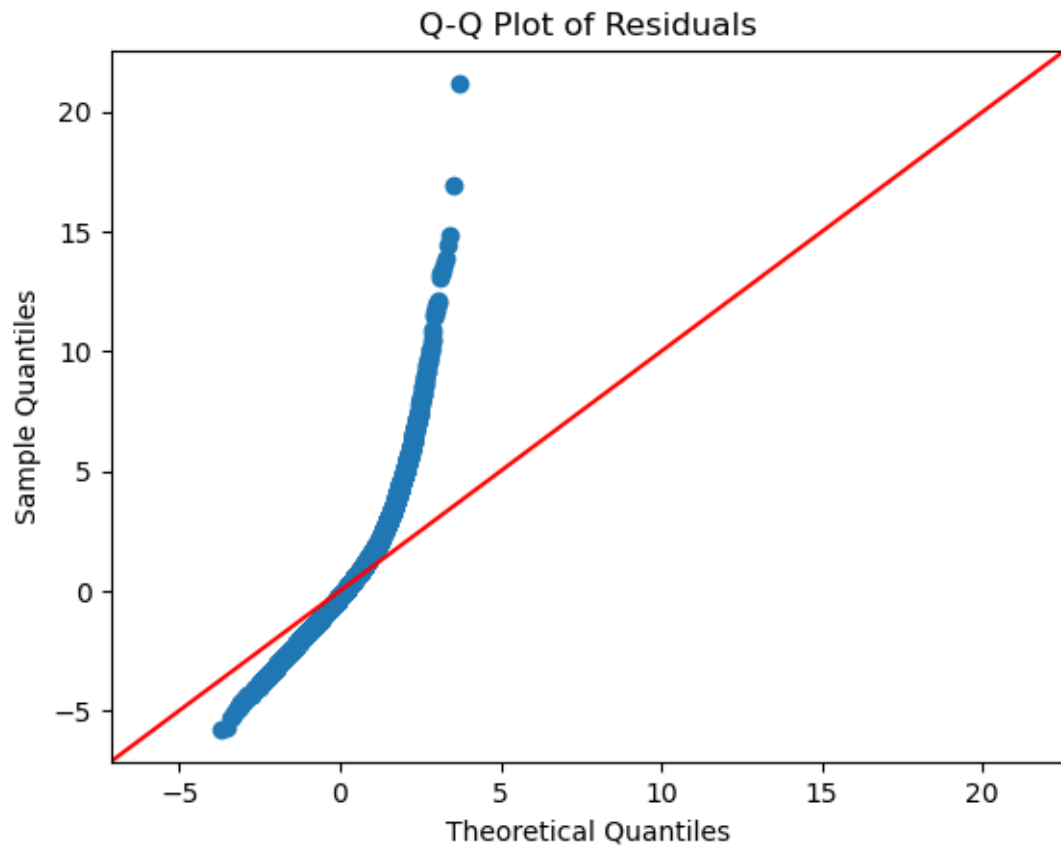
# Autocorrelation (Durbin-Watson test)
from statsmodels.stats.stattools import durbin_watson

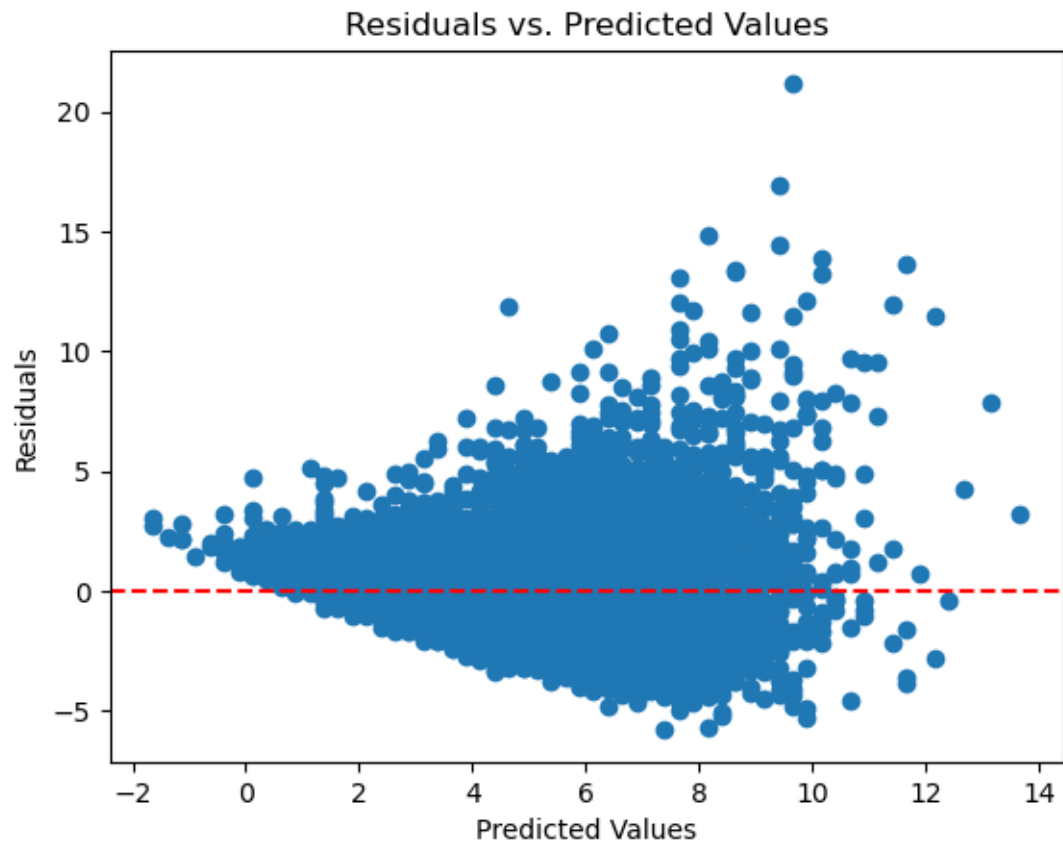
dw_statistic = durbin_watson(residuals)
print("Durbin-Watson statistic:", dw_statistic)

```

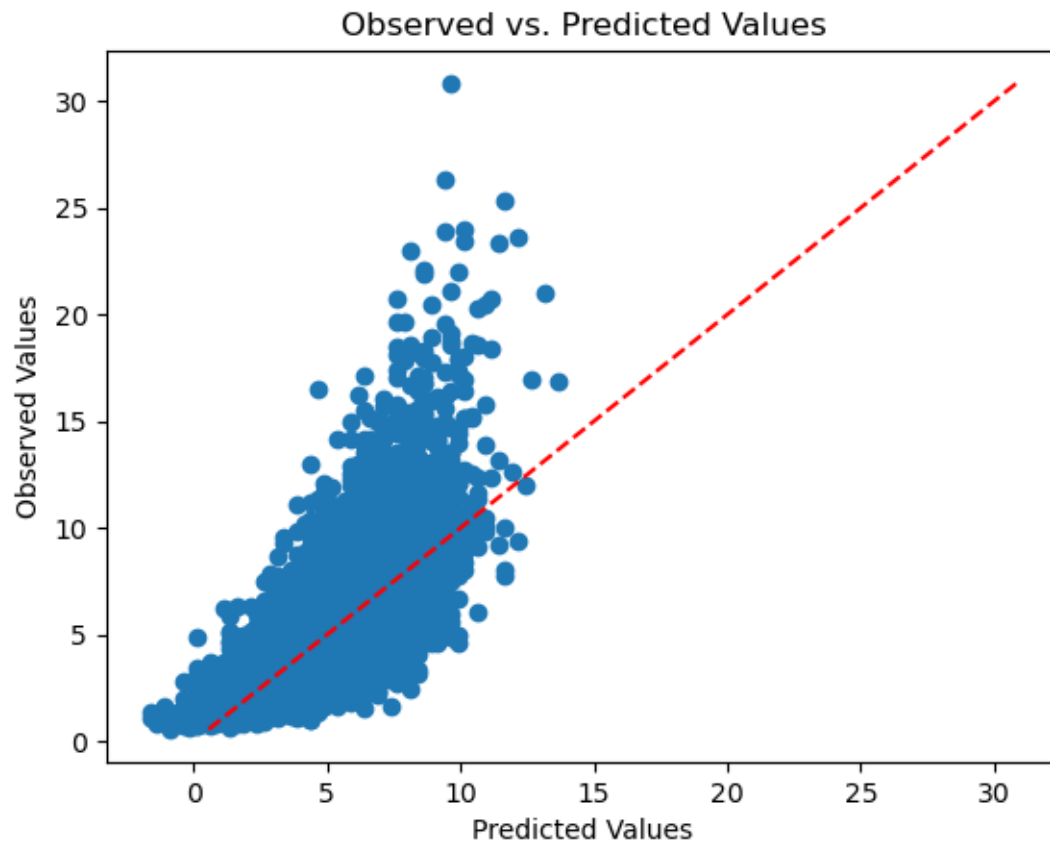
	Variable	VIF
0	const	61.878104
1	num_visitors	1.000000

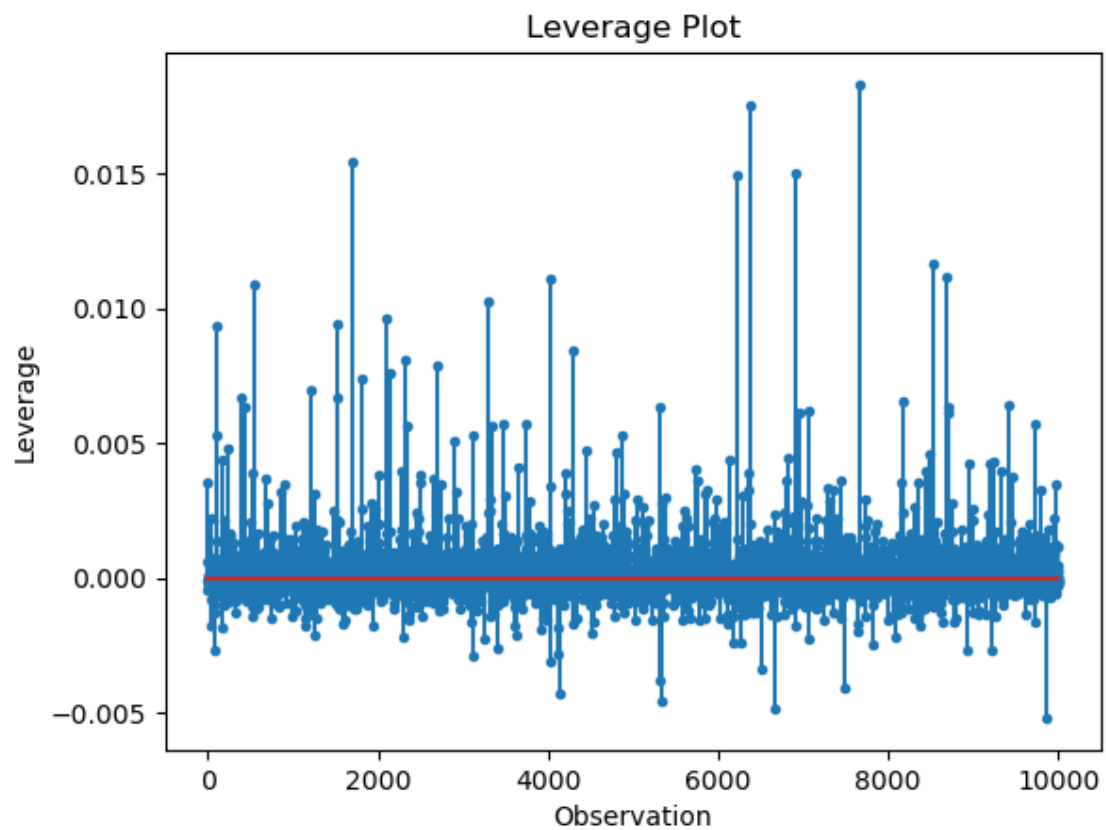


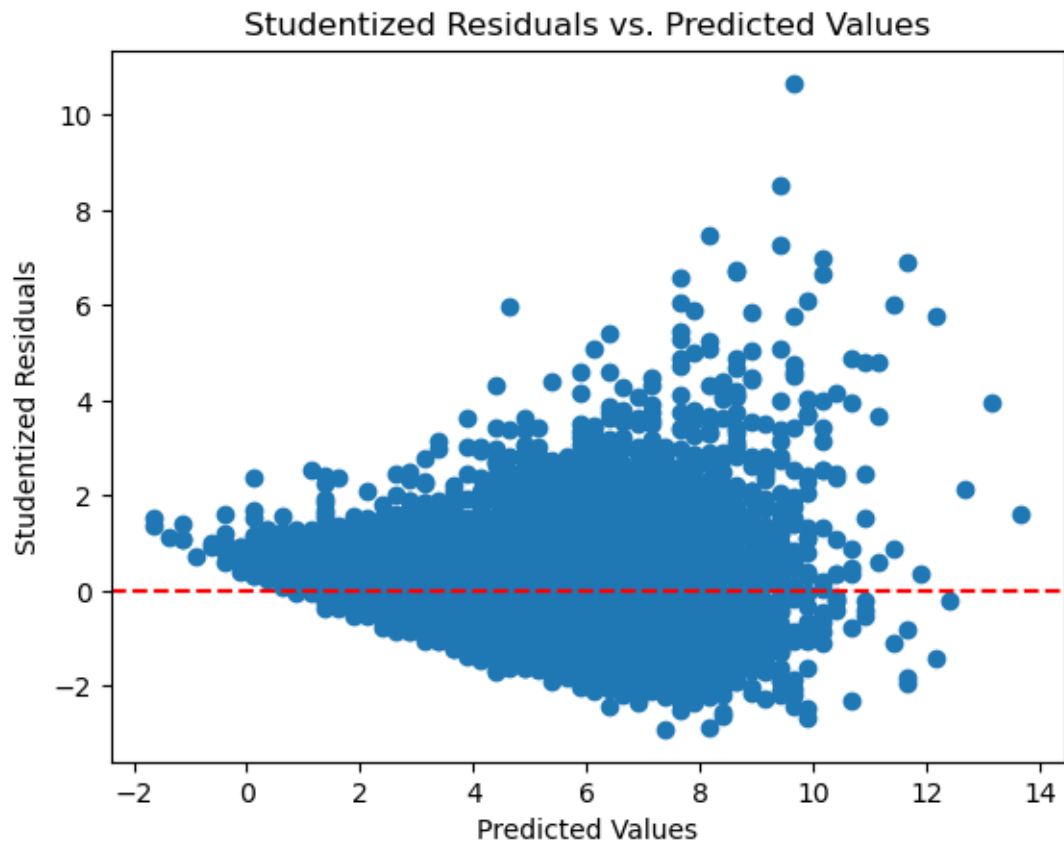


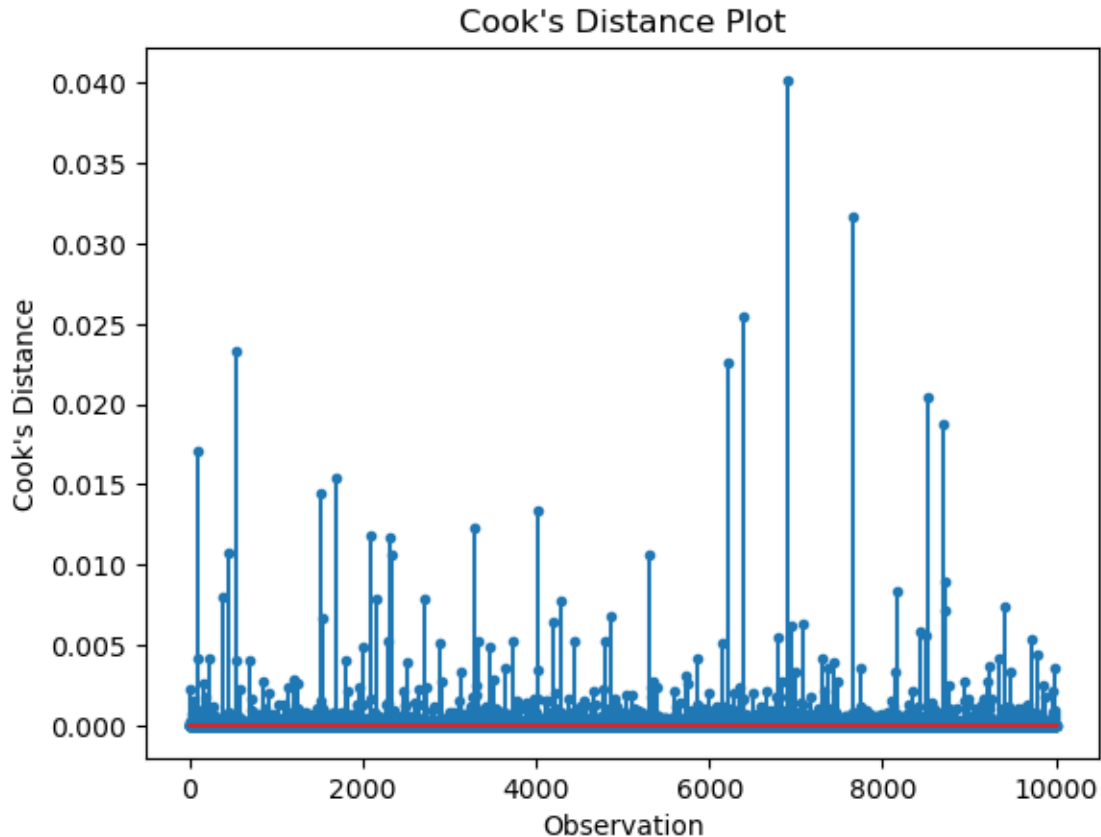












Durbin-Watson statistic: 2.032198935288371

the Durbin-Watson statistic is 2.032198935288371, which suggests that there is no significant autocorrelation present in the residuals.

Regarding the residual analysis, it is important to assess the assumptions of normality, linearity, and homoscedasticity.

- Residuals Histogram: The fact that the residuals histogram appears to be approximately normally distributed is a positive indication. However, the deviation from a perfect normal distribution suggests that there might be some departure from normality.
- QQ Plot: The absence of dots on the red line in the QQ plot indicates a departure from the assumption of normality. The deviations from the line at the extremes suggest that the tails of the residuals may not follow a normal distribution.
- Residuals vs. Predicted Values Plot: The clustering of the majority of the residuals around a central region (a wide bunch of blue dots) suggests a non-linear relationship between the predictors and the response variable. The pattern of the residuals becoming more spread out as the predicted values increase indicates possible heteroscedasticity.

Based on these observations, there may be some violations of the assumptions of normality, linearity, and homoscedasticity

```
[13]: # part 3.4

prob_wait_time_gt_7_5 = 1 - stats.nbinom.cdf(7, params_negbinom[0],
↳params_negbinom[1])
print("probability that when there are 50 or more visitors per hour, they can
↳expect to wait more than 7.5 minutes to log on:", prob_wait_time_gt_7_5)
```

probability that when there are 50 or more visitors per hour, they can expect to wait more than 7.5 minutes to log on: 1.0

## 1 3.5

To determine which assets are real and which are fake/simulated, we can use a hypothesis testing approach. We will test whether the sample returns for each asset come from a normal distribution or not. If the p-value of the test is less than the chosen significance level (e.g. 0.05), we reject the null hypothesis that the sample is from a normal distribution and conclude that the returns are likely to be fake or simulated.

```
[14]: import pandas as pd
import numpy as np
import scipy.stats as stats

# read the tick data sheet into a pandas dataframe
df_tick = pd.read_excel('data.xlsx', sheet_name='tick data')

# set the significance level
alpha = 0.05

# iterate over columns (Assets A-J)
for col in df_tick.columns:
    # perform the Shapiro-Wilk test for normality
    stat, p = stats.shapiro(df_tick[col].dropna())
    # print the test results
    print(f'{col} - Test statistic: {stat:.4f}, p-value: {p:.4f}')
    # check if p-value is less than the chosen significance level
    if p < alpha:
        print(f'{col} - Reject the null hypothesis of normality at {alpha}
↳significance level. Returns are likely to be fake or simulated.")
    else:
        print(f'{col} - Fail to reject the null hypothesis of normality at
↳{alpha} significance level. Returns are likely to be real.")
```

Asset A - Test statistic: 0.9999, p-value: 0.7220

Asset A - Fail to reject the null hypothesis of normality at 0.05 significance level. Returns are likely to be real.

Asset B - Test statistic: 0.9485, p-value: 0.0000

Asset B - Reject the null hypothesis of normality at 0.05 significance level.

Returns are likely to be fake or simulated.  
 Asset C - Test statistic: 0.9475, p-value: 0.0000  
 Asset C - Reject the null hypothesis of normality at 0.05 significance level.  
 Returns are likely to be fake or simulated.  
 Asset D - Test statistic: 0.9478, p-value: 0.0000  
 Asset D - Reject the null hypothesis of normality at 0.05 significance level.  
 Returns are likely to be fake or simulated.  
 Asset E - Test statistic: 0.9999, p-value: 0.7669  
 Asset E - Fail to reject the null hypothesis of normality at 0.05 significance level. Returns are likely to be real.  
 Asset F - Test statistic: 0.9127, p-value: 0.0000  
 Asset F - Reject the null hypothesis of normality at 0.05 significance level.  
 Returns are likely to be fake or simulated.  
 Asset G - Test statistic: 0.9557, p-value: 0.0000  
 Asset G - Reject the null hypothesis of normality at 0.05 significance level.  
 Returns are likely to be fake or simulated.  
 Asset H - Test statistic: 0.9999, p-value: 0.4204  
 Asset H - Fail to reject the null hypothesis of normality at 0.05 significance level. Returns are likely to be real.  
 Asset I - Test statistic: 0.9999, p-value: 0.7101  
 Asset I - Fail to reject the null hypothesis of normality at 0.05 significance level. Returns are likely to be real.  
 Asset J - Test statistic: 0.9999, p-value: 0.1587  
 Asset J - Fail to reject the null hypothesis of normality at 0.05 significance level. Returns are likely to be real.

```
/Applications/anaconda3/lib/python3.9/site-
packages/scipy/stats/_morestats.py:1800: UserWarning: p-value may not be
accurate for N > 5000.
```

```
warnings.warn("p-value may not be accurate for N > 5000.")
```

The output suggests that Assets B, C, D, F, and G are likely to have fake or simulated returns and the remaining assets are likely to have real returns.

To make a final determination, I perform additional tests and analyses:

Time Series plot of asset returns: visually inspect the plots for any patterns or discrepancies that could indicate that the returns are fake or simulated.

Autocorrelation analysis: look for any temporal dependence or serial correlation in the data. If the data has been simulated, we may expect to see little to no autocorrelation.

Cross-correlation analysis: compare the correlations between the real and fake assets. If the correlations between the fake assets are higher than expected, it may indicate that they have been generated from a common source.

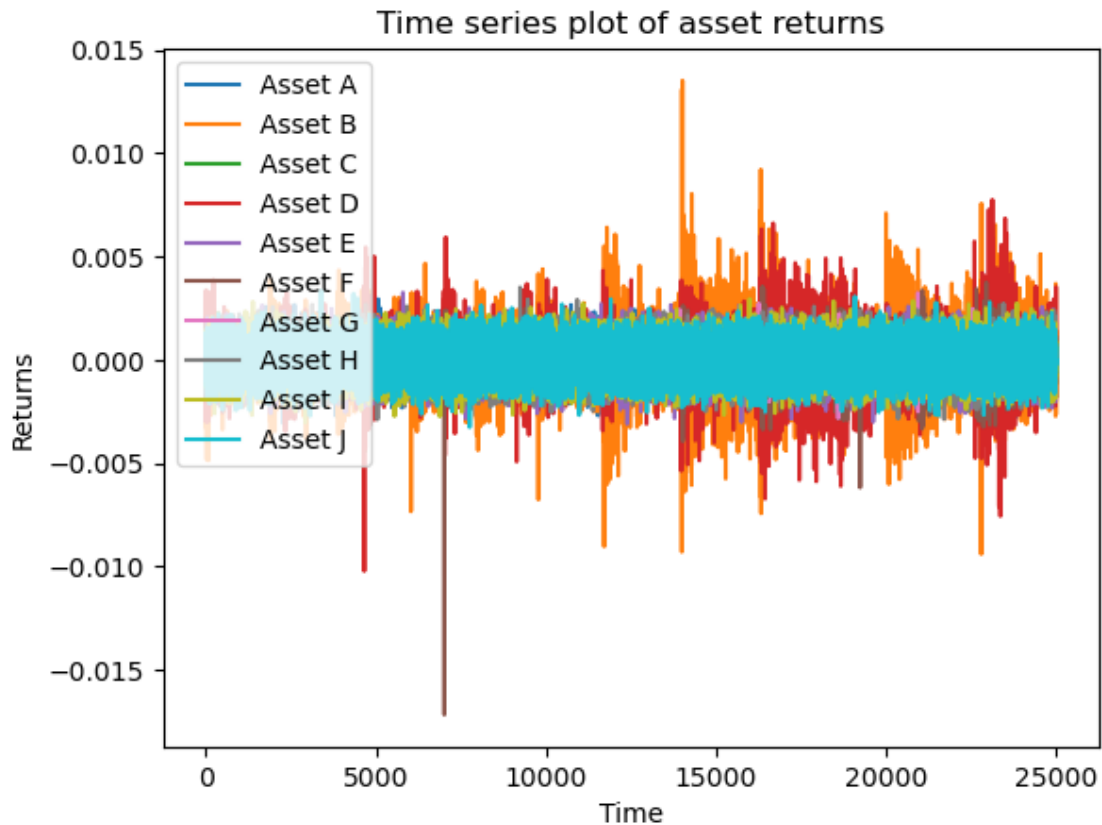
```
[15]: # read the tick data sheet into a pandas dataframe
df = pd.read_excel('data.xlsx', sheet_name='tick data')

# plot the returns data
```

```

df.plot()
plt.xlabel('Time')
plt.ylabel('Returns')
plt.legend(loc='upper left')
plt.title('Time series plot of asset returns')
plt.show()

```



```

[16]: import statsmodels.graphics.tsaplots as tsaplots

# plot the autocorrelation and partial autocorrelation functions for each
# asset, which we can use to detect any temporal dependence or serial
# correlation in the data.
# iterate over columns (Assets A-J)
for col in df.columns:
    # plot autocorrelation and partial autocorrelation
    fig, ax = plt.subplots(2,1, figsize=(8,6))
    tsaplots.plot_acf(df[col].dropna(), ax=ax[0])
    tsaplots.plot_pacf(df[col].dropna(), ax=ax[1])
    fig.suptitle(f'Autocorrelation of {col} returns')
    plt.show()

```

```

#cross-correlation analysis on the returns data and compare the correlations
↳ between the real and fake assets:
#compute the correlation matrix for all assets
corr_matrix = df.corr()

#plot the correlation matrix as a heatmap
fig, ax = plt.subplots(figsize=(8,8))
im = ax.imshow(corr_matrix, cmap='coolwarm', vmin=-1, vmax=1)
ax.set_xticks(range(len(df.columns)))
ax.set_yticks(range(len(df.columns)))
ax.set_xticklabels(df.columns, rotation=90)
ax.set_yticklabels(df.columns)
ax.set_title('Correlation Matrix')
fig.colorbar(im, ax=ax, label='Correlation')
plt.show()

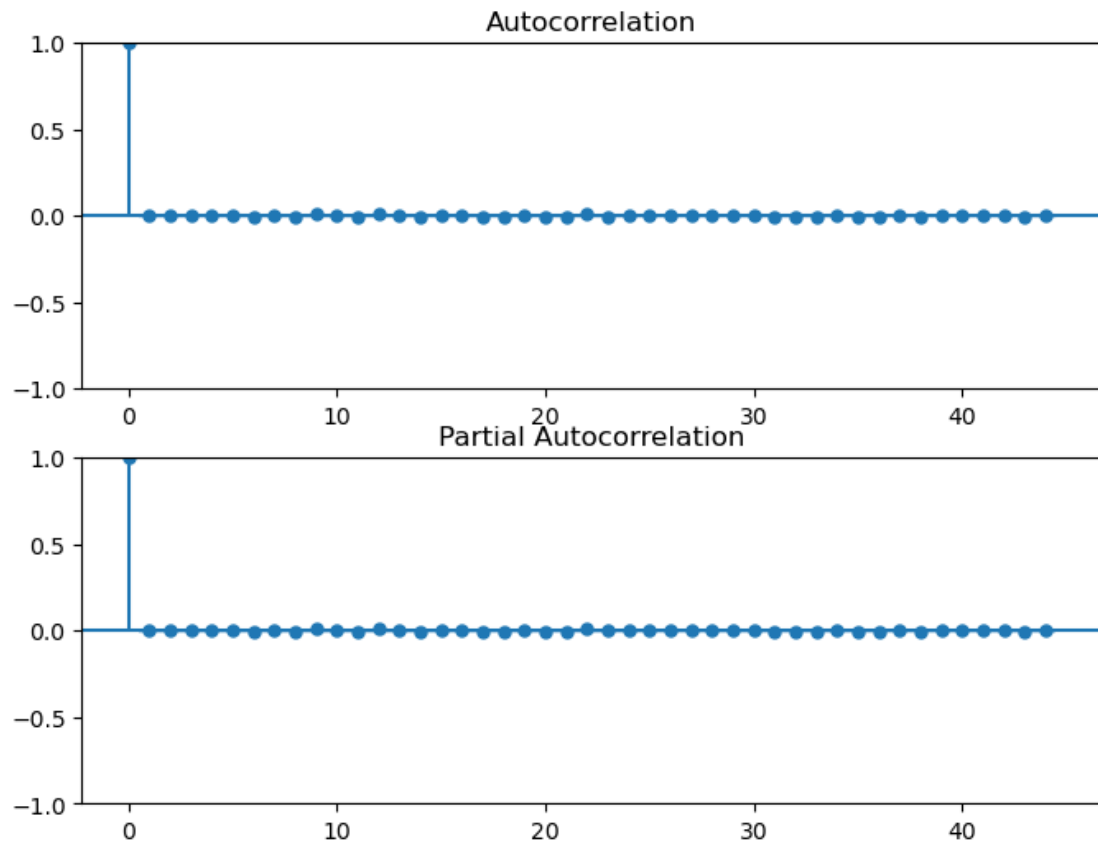
```

/Applications/anaconda3/lib/python3.9/site-packages/statsmodels/graphics/tsaplots.py:348: FutureWarning: The default method 'yw' can produce PACF values outside of the [-1,1] interval. After 0.13, the default will change to unadjusted Yule-Walker ('ywm'). You can use this method now by setting method='ywm'.

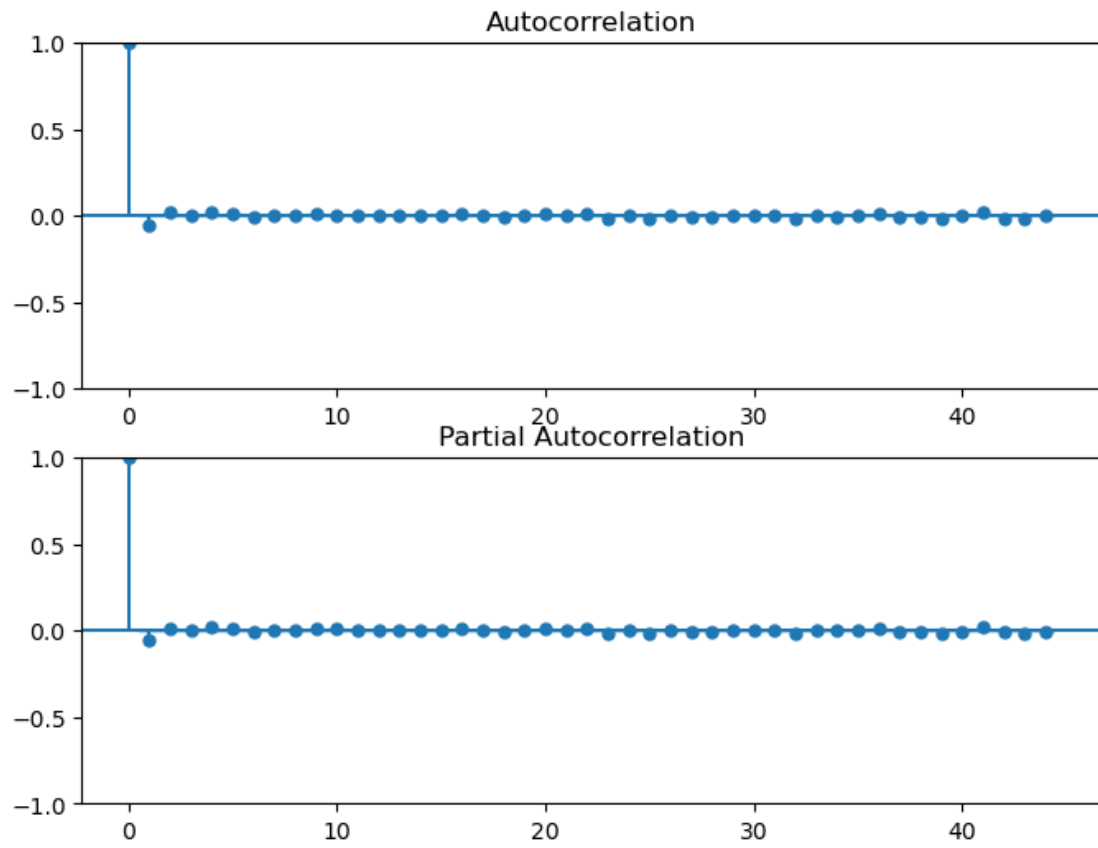
```
warnings.warn(
```



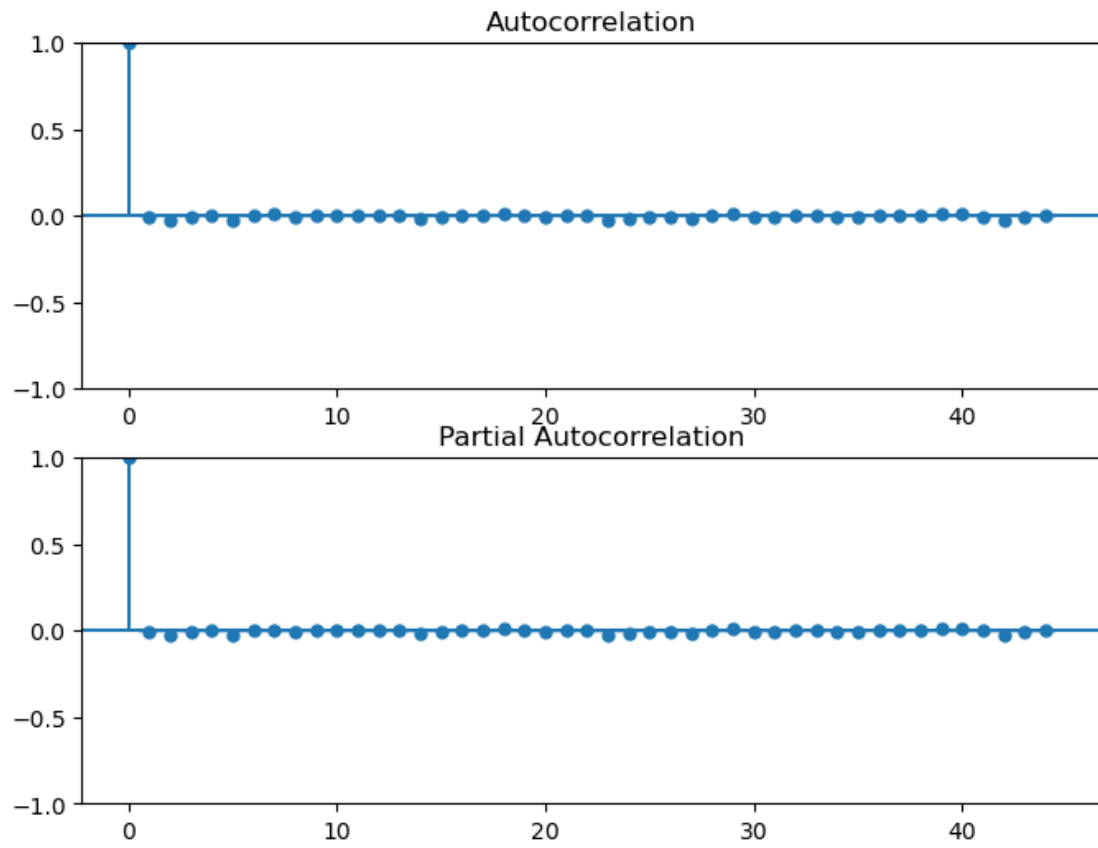
### Autocorrelation of Asset A returns



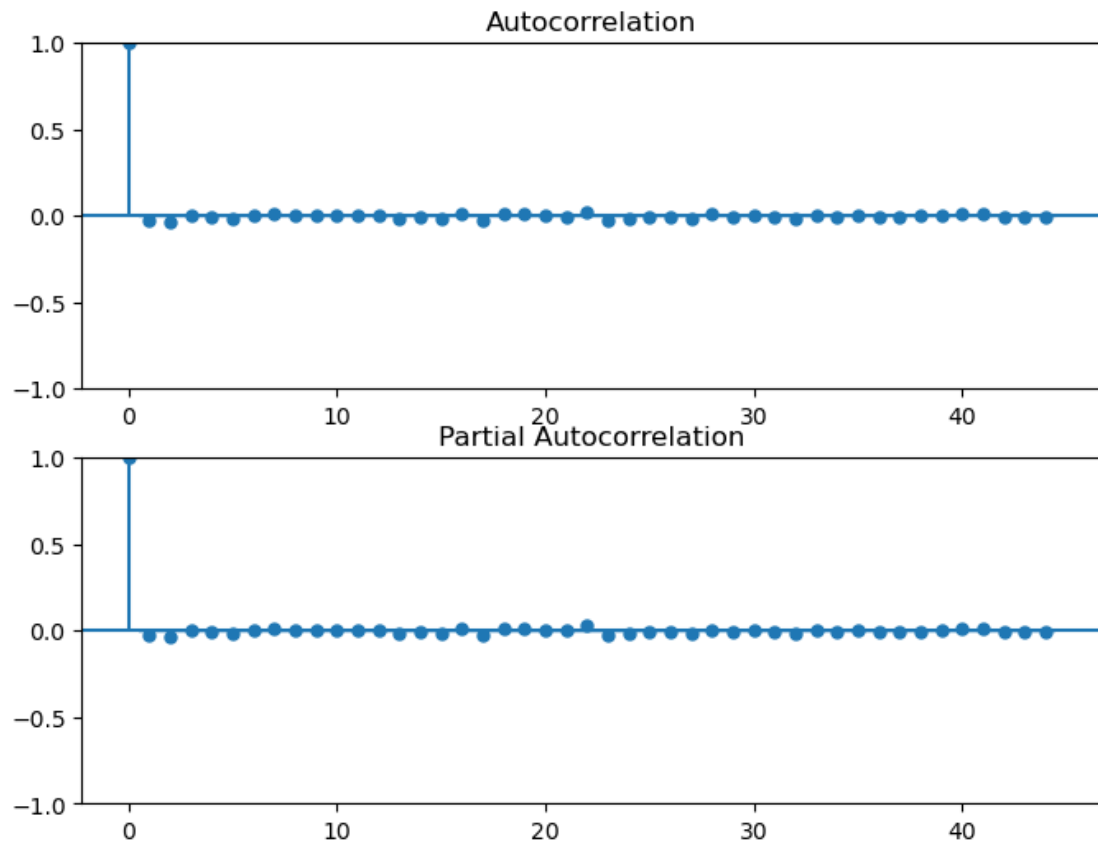
### Autocorrelation of Asset B returns



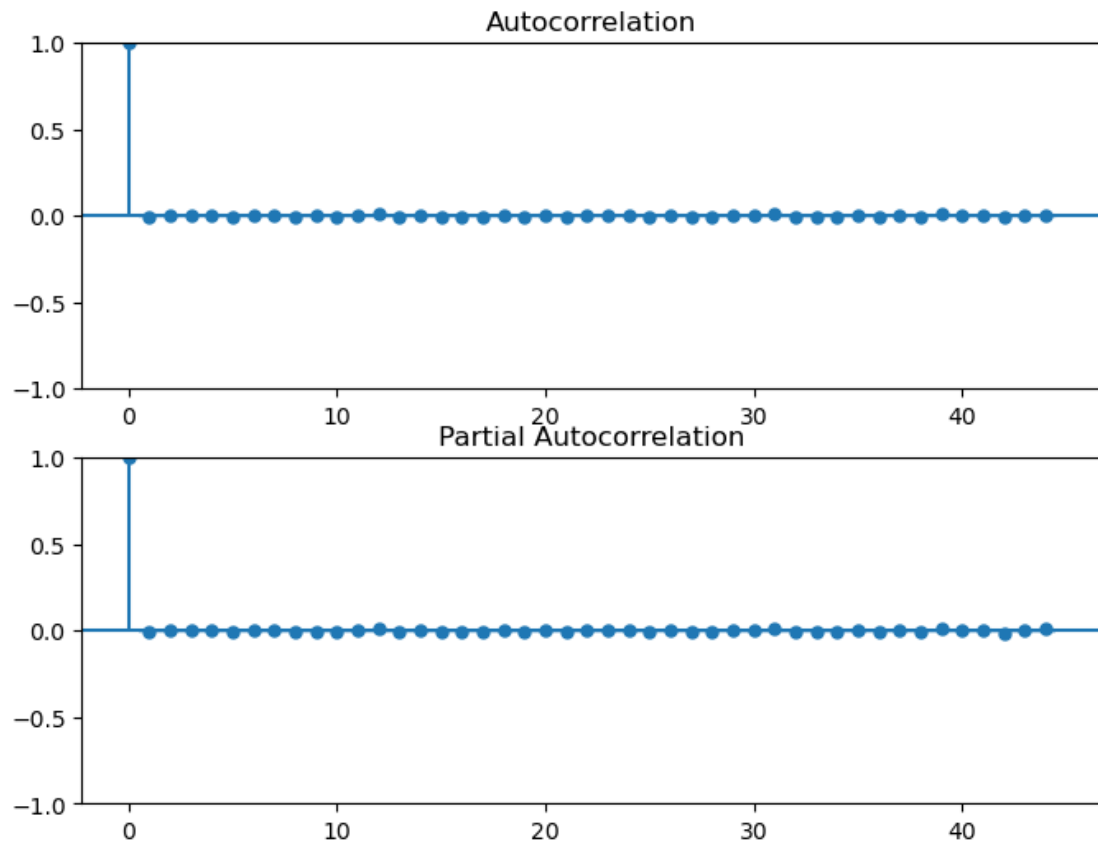
### Autocorrelation of Asset C returns



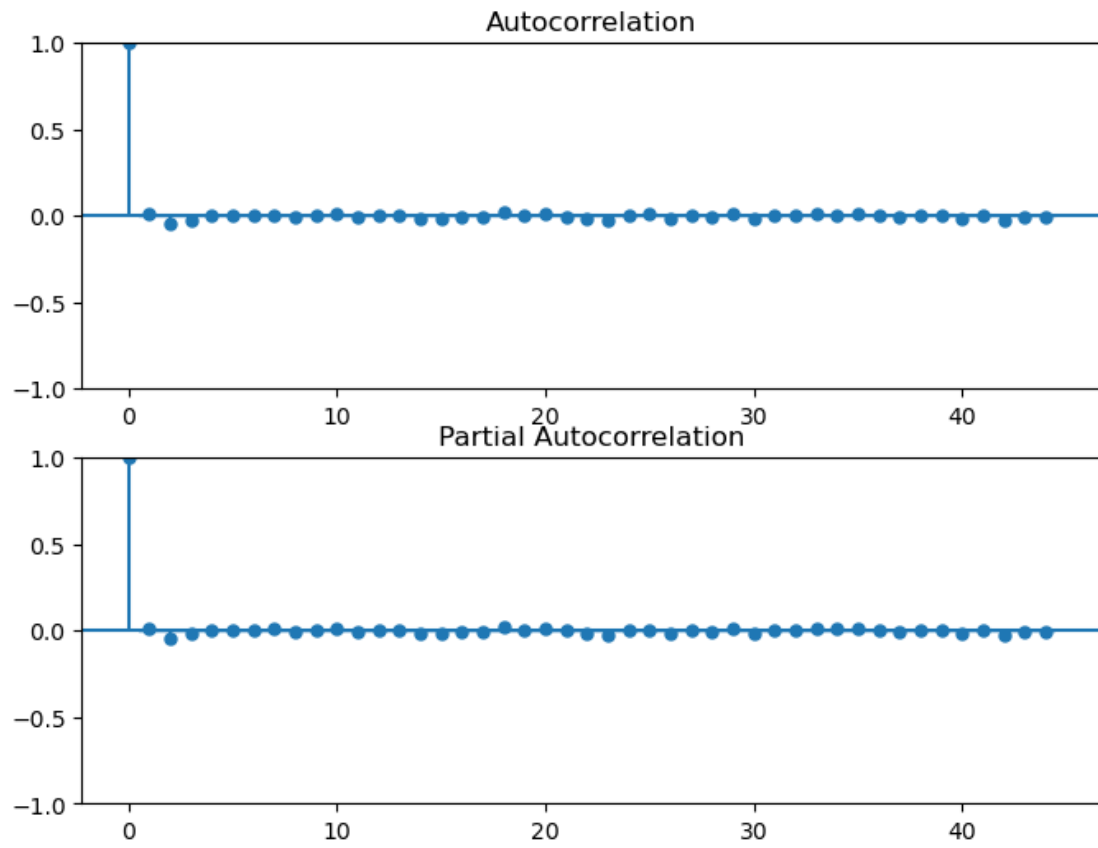
### Autocorrelation of Asset D returns



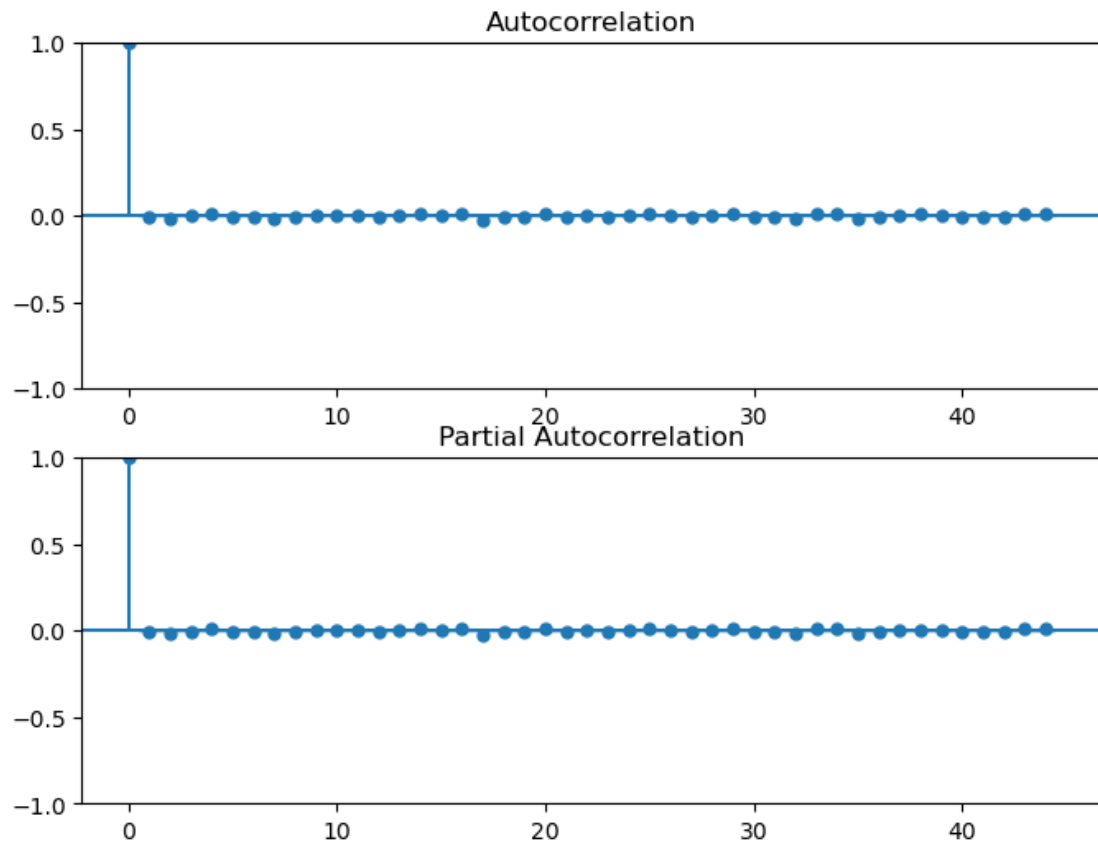
### Autocorrelation of Asset E returns



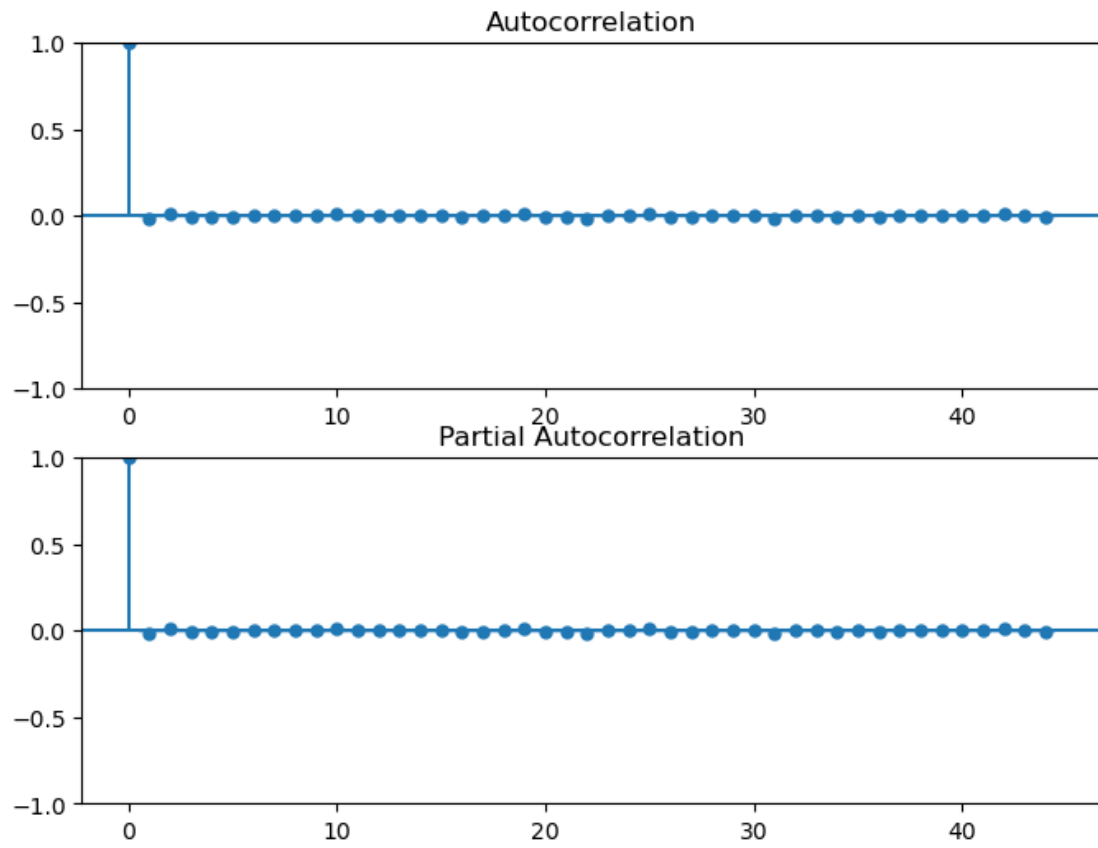
### Autocorrelation of Asset F returns



### Autocorrelation of Asset G returns

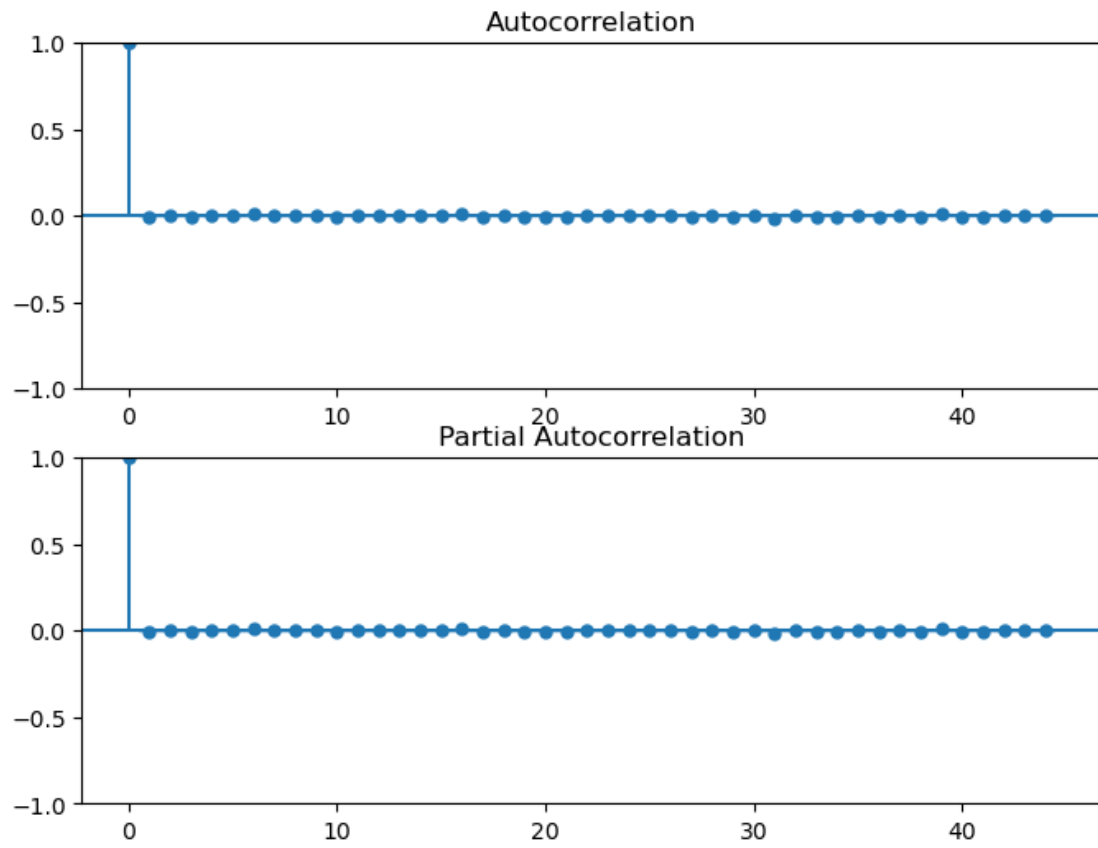


# Autocorrelation of Asset H returns

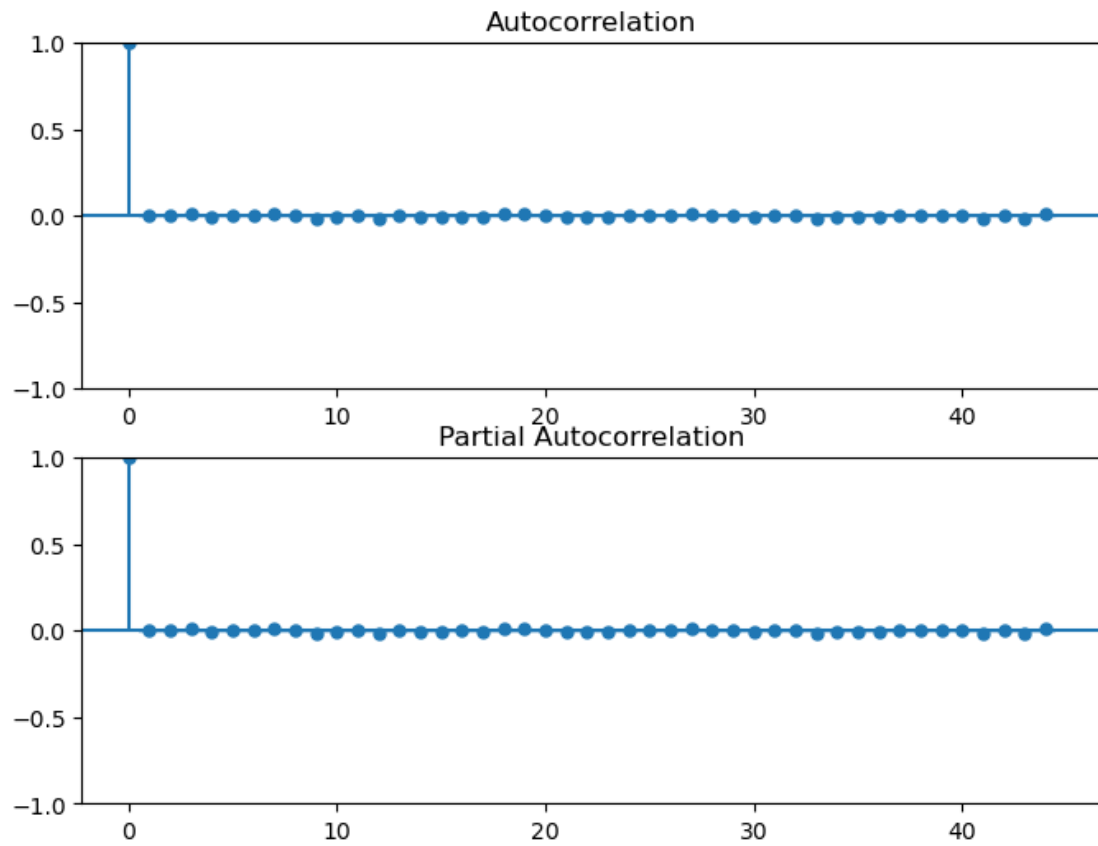


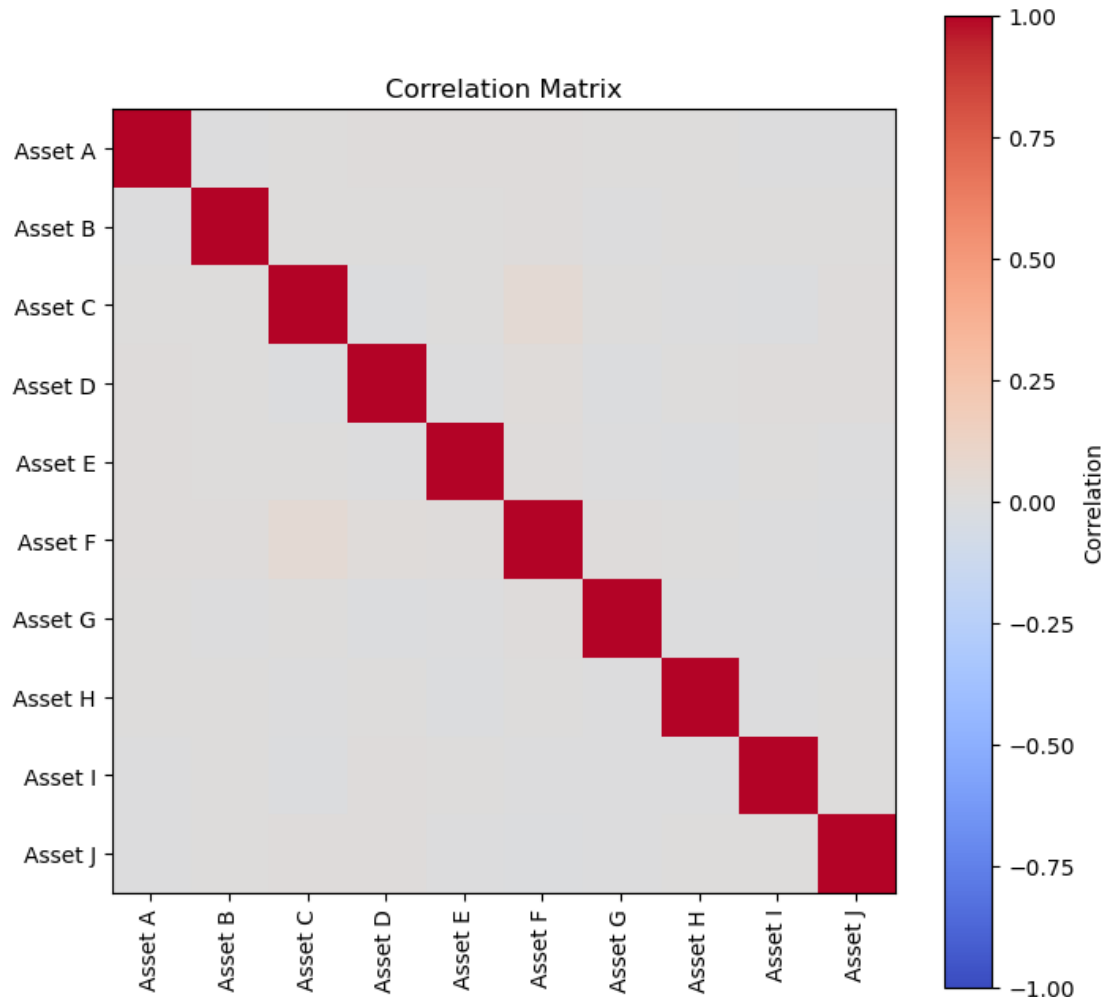


### Autocorrelation of Asset I returns



### Autocorrelation of Asset J returns





Interpretations of the results:

- Time series plot: Asset J seems to be more volatile than the other assets, as it occupies a larger range on the y-axis. The other assets largely cluster within a smaller range, with only a few having an occasional peak going beyond the range. This may suggest that Asset J is riskier or more unpredictable than the other assets.
- Autocorrelation plots: all the dots on the autocorrelation plots are on or almost on the line, which suggests that the returns for each asset is random and does not have any significant temporal dependence or serial correlation. Specifically, the autocorrelation is close to 0 for all lags, meaning that the returns on any day are generally uncorrelated to returns on previous days. This is a good sign that the returns for each of the assets are genuine.
- Correlation plot: there seems to be no correlation between the assets based on the heat map or correlation plot, this may suggest that the assets have no significant relationship between them. This may indicate that they are independent assets without any influence from each other.

Overall, the results of the analysis suggest that the returns for each of the assets are likely to be genuine and not fake or simulated since the autocorrelation plots show no temporal dependence or serial correlation, and the correlation plot shows no significant correlation between the assets.