I will be performing multiple regression analysis on a dataset using Python. For this, I will choose a finance dataset for regression analysis. One commonly used dataset in finance is the "Stock Market Dataset." It typically includes information about stock prices, trading volumes, and various financial indicators for different companies over a period of time.

In [3]:
```
pip install yfinance
```

```
Requirement already satisfied: yfinance in c:\users\sande\anaconda3\lib\si
te-packages (0.2.38)
Requirement already satisfied: pandas>=1.3.0 in c:\users\sande\anaconda3\l
ib\site-packages (from yfinance) (1.5.3)
Requirement already satisfied: numpy>=1.16.5 in c:\users\sande\anaconda3\l
ib\site-packages (from yfinance) (1.24.3)
Requirement already satisfied: requests>=2.31 in c:\users\sande\anaconda3
\lib\site-packages (from yfinance) (2.31.0)
Requirement already satisfied: multitasking>=0.0.7 in c:\users\sande\anaco
nda3\lib\site-packages (from yfinance) (0.0.11)
Requirement already satisfied: lxml>=4.9.1 in c:\users\sande\anaconda3\lib
\site-packages (from yfinance) (4.9.2)
Requirement already satisfied: appdirs>=1.4.4 in c:\users\sande\anaconda3
\lib\site-packages (from yfinance) (1.4.4)
Requirement already satisfied: pytz>=2022.5 in c:\users\sande\anaconda3\li
b\site-packages (from yfinance) (2022.7)
Requirement already satisfied: frozendict>=2.3.4 in c:\users\sande\anacond
a3\lib\site-packages (from yfinance) (2.4.4)
Requirement already satisfied: peewee>=3.16.2 in c:\users\sande\anaconda3
\lib\site-packages (from yfinance) (3.17.5)
Requirement already satisfied: beautifulsoup4>=4.11.1 in c:\users\sande\an
aconda3\lib\site-packages (from yfinance) (4.12.2)
Requirement already satisfied: html5lib>=1.1 in c:\users\sande\anaconda3\l
ib\site-packages (from yfinance) (1.1)
Requirement already satisfied: soupsieve>1.2 in c:\users\sande\anaconda3\l
ib\site-packages (from beautifulsoup4>=4.11.1->yfinance) (2.4)
Requirement already satisfied: six>=1.9 in c:\users\sande\appdata\roaming
\python\python311\site-packages (from html5lib>=1.1->yfinance) (1.16.0)
Requirement already satisfied: webencodings in c:\users\sande\anaconda3\li
b\site-packages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\sande\ap
pdata\roaming\python\python311\site-packages (from pandas>=1.3.0->yfinanc
e) (2.8.2)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\sande
\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\sande\anaconda3\li
b\site-packages (from requests>=2.31->yfinance) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\sande\anacon
da3\lib\site-packages (from requests>=2.31->yfinance) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\sande\anacon
da3\lib\site-packages (from requests>=2.31->yfinance) (2023.7.22)
Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
# Importing necessary libraries
import pandas as pd
import numpy as np
import yfinance as yf
import statsmodels.api as sm

# List of companies we want to analyze
companies = ['AAPL', 'MSFT', 'GOOG', 'AMZN']

# Fetching historical stock price data from Yahoo Finance
stock_data = yf.download(companies, start='2020-01-01', end='2021-12-31')

# Dropping missing values if any
stock_data.dropna(inplace=True)

# Displaying the first few rows of the dataset
print(stock_data.head())
```

```
[*********************100%%**********************]  4 of 4 completed
```

```
Price       Adj Close                                                Close
\
Ticker           AAPL        AMZN        GOOG        MSFT        AAPL        A
MZN
Date
2020-01-02  72.960472  94.900497  68.368500  154.215652  75.087502  94.900
497
2020-01-03  72.251144  93.748497  68.032997  152.295425  74.357498  93.748
497
2020-01-06  72.826851  95.143997  69.710503  152.689087  74.949997  95.143
997
2020-01-07  72.484352  95.343002  69.667000  151.296875  74.597504  95.343
002
2020-01-08  73.650337  94.598503  70.216003  153.706818  75.797501  94.598
503

Price                                High                ...        Low  \
Ticker           GOOG        MSFT       AAPL        AMZN  ...       GOOG
Date                                                      ...
2020-01-02  68.368500  160.619995  75.150002  94.900497  ...  67.077499
2020-01-03  68.032997  158.619995  75.144997  94.309998  ...  67.277199
2020-01-06  69.710503  159.029999  74.989998  95.184502  ...  67.500000
2020-01-07  69.667000  157.580002  75.224998  95.694504  ...  69.518997
2020-01-08  70.216003  160.089996  76.110001  95.550003  ...  69.542000

Price                          Open                                      \
Ticker            MSFT         AAPL        AMZN       GOOG        MSFT
Date
2020-01-02  158.330002  74.059998  93.750000  67.077499  158.779999
2020-01-03  158.059998  74.287498  93.224998  67.392998  158.320007
2020-01-06  156.509995  73.447502  93.000000  67.500000  157.080002
2020-01-07  157.320007  74.959999  95.224998  69.897003  159.320007
2020-01-08  157.949997  74.290001  94.902000  69.603996  158.929993

Price          Volume
Ticker           AAPL      AMZN      GOOG      MSFT
Date
2020-01-02  135480400  80580000  28132000  22622100
2020-01-03  146322800  75288000  23728000  21116200
2020-01-06  118387200  81236000  34646000  20813700
2020-01-07  108872000  80898000  30054000  21634100
2020-01-08  132079200  70160000  30560000  27746500

[5 rows x 24 columns]
```

Will proceed with Data Preparation step. Here, I will

1. Check the data types: Ensure that the data types are appropriate for each column.
2. Handle missing values: Check for and handle any missing values in the dataset.
3. Ensure data consistency: Check if the data is consistent and formatted correctly for analysis.

This will display the data types of each column, the number of missing values in each column, and basic descriptive statistics for the dataset.

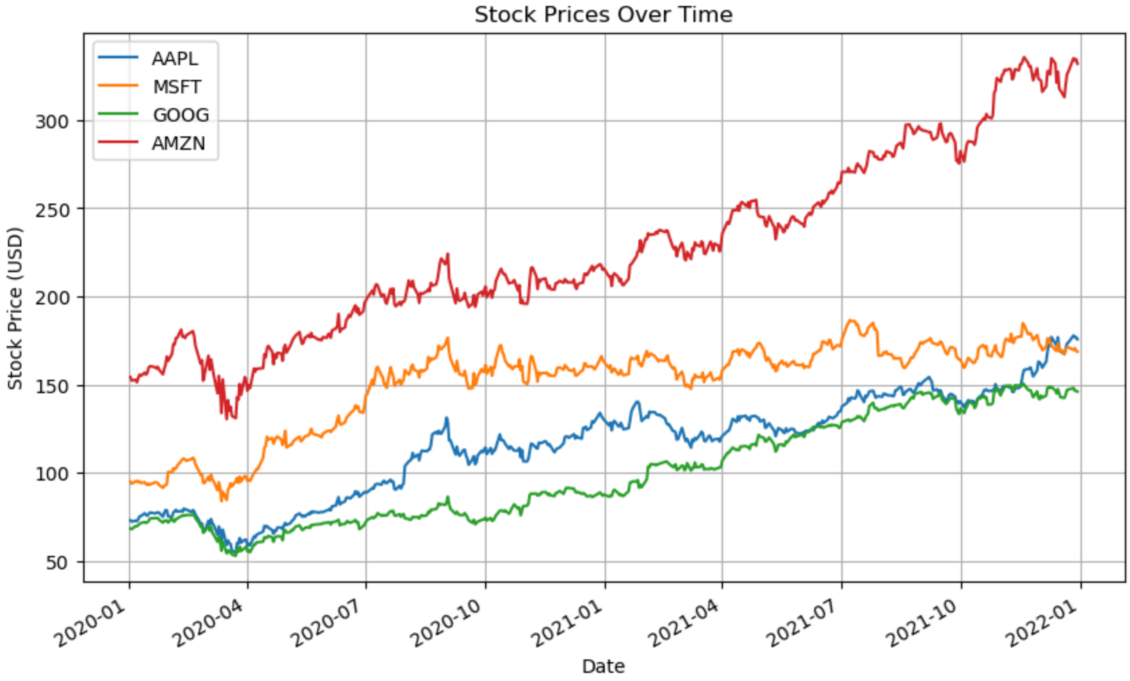
In [5]:
```
# Check data types
print(stock_data.dtypes)

# Check for missing values
print(stock_data.isnull().sum())

# Ensure data consistency
print(stock_data.describe())
```

```
Price      Ticker
Adj Close  AAPL      float64
           AMZN      float64
           GOOG      float64
           MSFT      float64
Close      AAPL      float64
           AMZN      float64
           GOOG      float64
           MSFT      float64
High       AAPL      float64
           AMZN      float64
           GOOG      float64
           MSFT      float64
Low        AAPL      float64
           AMZN      float64
           GOOG      float64
           MSFT      float64
Open       AAPL      float64
           AMZN      float64
           GOOG      float64
           MSFT      float64
Volume     AAPL        int64
           AMZN        int64
           GOOG        int64
           MSFT        int64
dtype: object
Price      Ticker
Adj Close  AAPL      0
           AMZN      0
           GOOG      0
           MSFT      0
Close      AAPL      0
           AMZN      0
           GOOG      0
           MSFT      0
High       AAPL      0
           AMZN      0
           GOOG      0
           MSFT      0
Low        AAPL      0
           AMZN      0
           GOOG      0
           MSFT      0
Open       AAPL      0
           AMZN      0
           GOOG      0
           MSFT      0
Volume     AAPL      0
           AMZN      0
           GOOG      0
           MSFT      0
dtype: int64
Price    Adj Close                                                  Close  \
Ticker        AAPL        AMZN        GOOG        MSFT        AAPL
count   504.000000  504.000000  504.000000  504.000000  504.000000
mean    115.741158  150.553222   99.660341  227.412939  118.005079
std      29.183268   26.083053   29.457459   51.198459   29.349252
min      54.632900   83.830498   52.831001  130.375595   56.092499
25%      89.386547  144.369999   73.789251  194.782990   91.526875
50%     121.373463  159.871246   89.078751  215.896286  123.645000
75%     138.798916  167.748619  127.641247  266.068382  140.960003
```

```
max      177.824463  186.570496  150.709000  335.709808  180.330002

Price                                                  High              ...  \
Ticker         AMZN        GOOG        MSFT        AAPL        AMZN  ...
count   504.000000  504.000000  504.000000  504.000000  504.000000  ...
mean    150.553222   99.660341  234.199127  119.340968  152.328902  ...
std      26.083053   29.457459   51.484969   29.495388   26.301452  ...
min      83.830498   52.831001  135.419998   57.125000   87.972504  ...
25%     144.369999   73.789251  201.757504   92.881876  147.021000  ...
50%     159.871246   89.078751  222.669998  125.080002  161.657249  ...
75%     167.748619  127.641247  272.952507  142.164997  169.445499  ...
max     186.570496  150.709000  343.109985  182.130005  188.654007  ...

Price          Low                    Open                            \
Ticker        GOOG        MSFT        AAPL        AMZN        GOOG
count   504.000000  504.000000  504.000000  504.000000  504.000000
mean     98.592506  231.619563  117.900437  150.599012   99.552615
std      29.385660   51.492485   29.362344   26.248373   29.482984
min      50.676800  132.520004   57.020000   82.075500   52.825500
25%      72.982878  197.682503   91.272499  143.743748   73.570501
50%      88.013500  219.539993  123.705002  160.102753   88.948502
75%     126.816748  270.325005  139.657505  167.668873  127.013374
max     149.887497  342.200012  181.119995  187.199997  151.000000

Price                     Volume
Ticker        MSFT          AAPL          AMZN          GOOG          MSFT
count   504.000000  5.040000e+02  5.040000e+02  5.040000e+02  5.040000e+02
mean    234.000119  1.242302e+08  8.329235e+07  3.159975e+07  3.187531e+07
std      51.438170  6.316090e+07  3.590935e+07  1.435656e+07  1.447710e+07
min     137.009995  4.100000e+07  2.903800e+07  6.936000e+06  1.055060e+07
25%     200.327499  8.119942e+07  5.855200e+07  2.193050e+07  2.272362e+07
50%     222.705002  1.088506e+08  7.419100e+07  2.815800e+07  2.777180e+07
75%     271.222504  1.478633e+08  1.001430e+08  3.642550e+07  3.607522e+07
max     344.619995  4.265100e+08  3.113460e+08  8.658200e+07  9.701270e+07

[8 rows x 24 columns]
```

Perform Explonatory Data Analysis(EDA): Here, we'll explore the data to understand its characteristics and relationships between variables. We'll visualize the data to gain insights and identify any patterns or trends.

In [6]:
```
import matplotlib.pyplot as plt

# Plot stock prices over time
stock_data['Adj Close'].plot(figsize=(10, 6))
plt.title('Stock Prices Over Time')
plt.xlabel('Date')
plt.ylabel('Stock Price (USD)')
plt.legend(companies)
plt.grid(True)
plt.show()
```



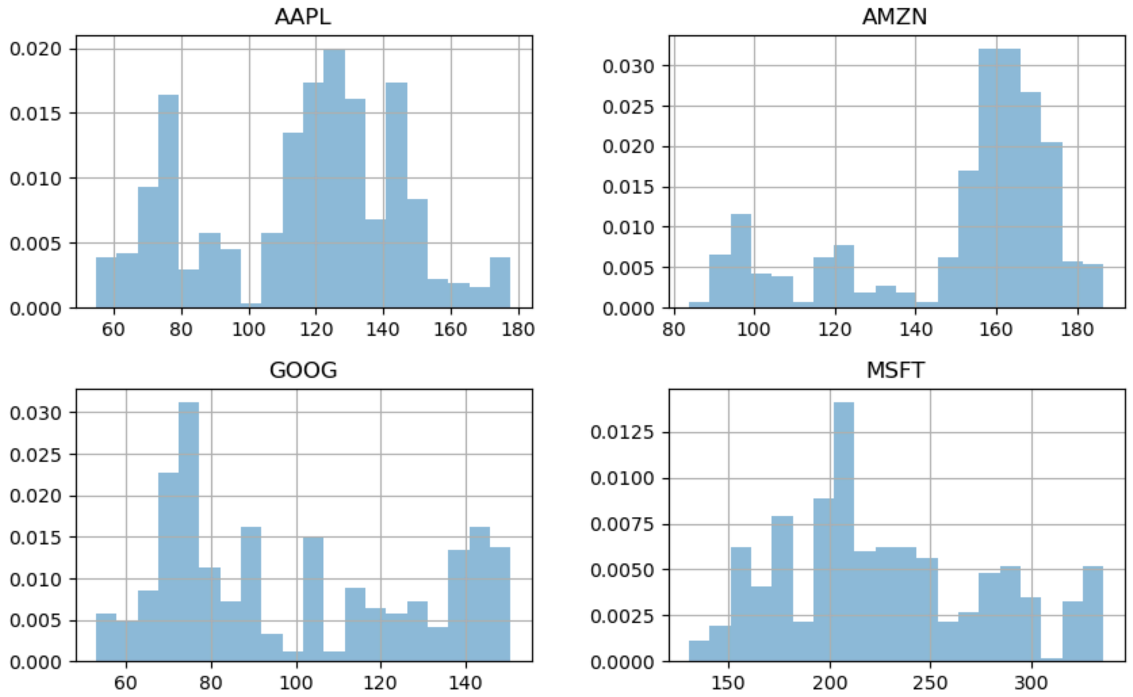
EDA Technique: Correlation Analysis: We'll compute the correlation matrix to identify relationships between different stocks.

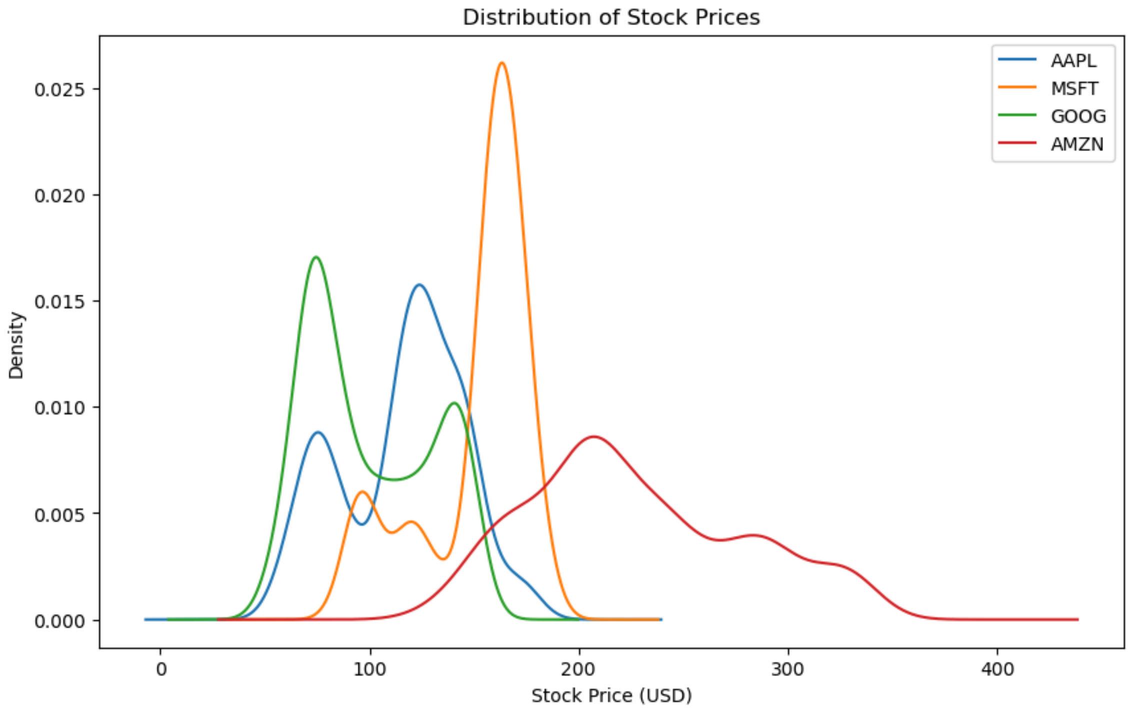
In [7]:
```
# Compute correlation matrix
correlation_matrix = stock_data['Adj Close'].corr()
print("Correlation Matrix:")
print(correlation_matrix)
```

```
Correlation Matrix:
Ticker       AAPL      AMZN      GOOG      MSFT
Ticker
AAPL     1.000000  0.897824  0.895248  0.936209
AMZN     0.897824  1.000000  0.734451  0.805830
GOOG     0.895248  0.734451  1.000000  0.967988
MSFT     0.936209  0.805830  0.967988  1.000000
```

Histograms and Density Plots: We'll visualize the distributions of stock prices for each company.

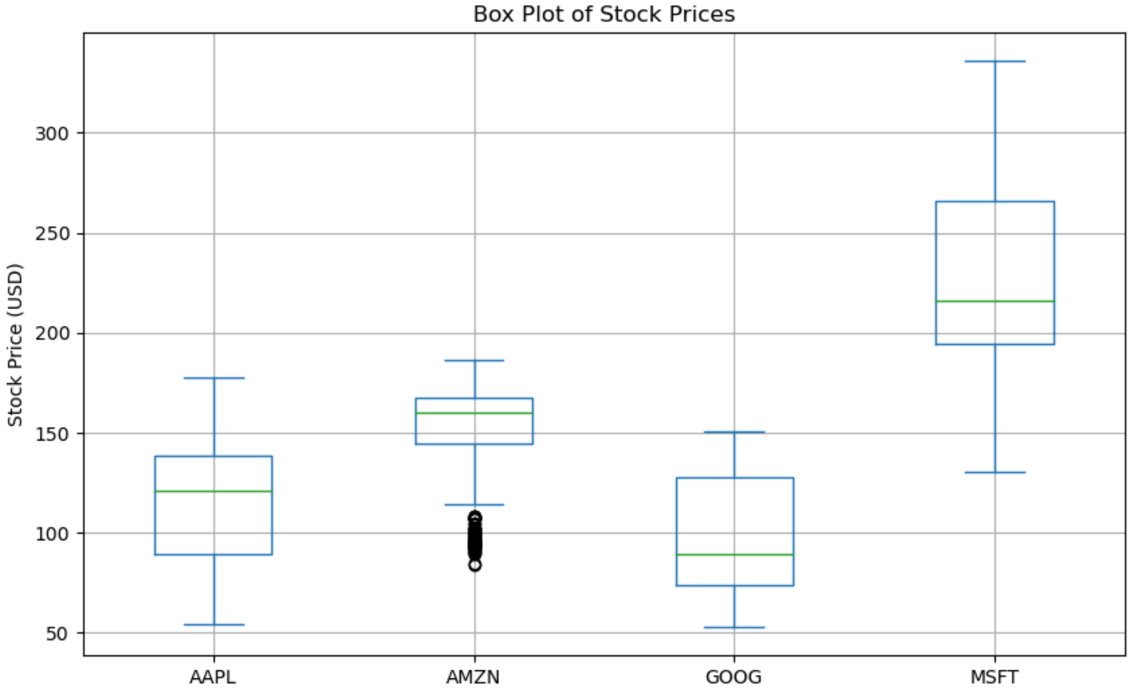
In [8]:
```
# Plot histograms and density plots
stock_data['Adj Close'].hist(bins=20, figsize=(10, 6), density=True, alpha=
stock_data['Adj Close'].plot(kind='kde', figsize=(10, 6))
plt.title('Distribution of Stock Prices')
plt.xlabel('Stock Price (USD)')
plt.ylabel('Density')
plt.legend(companies)
plt.show()
```





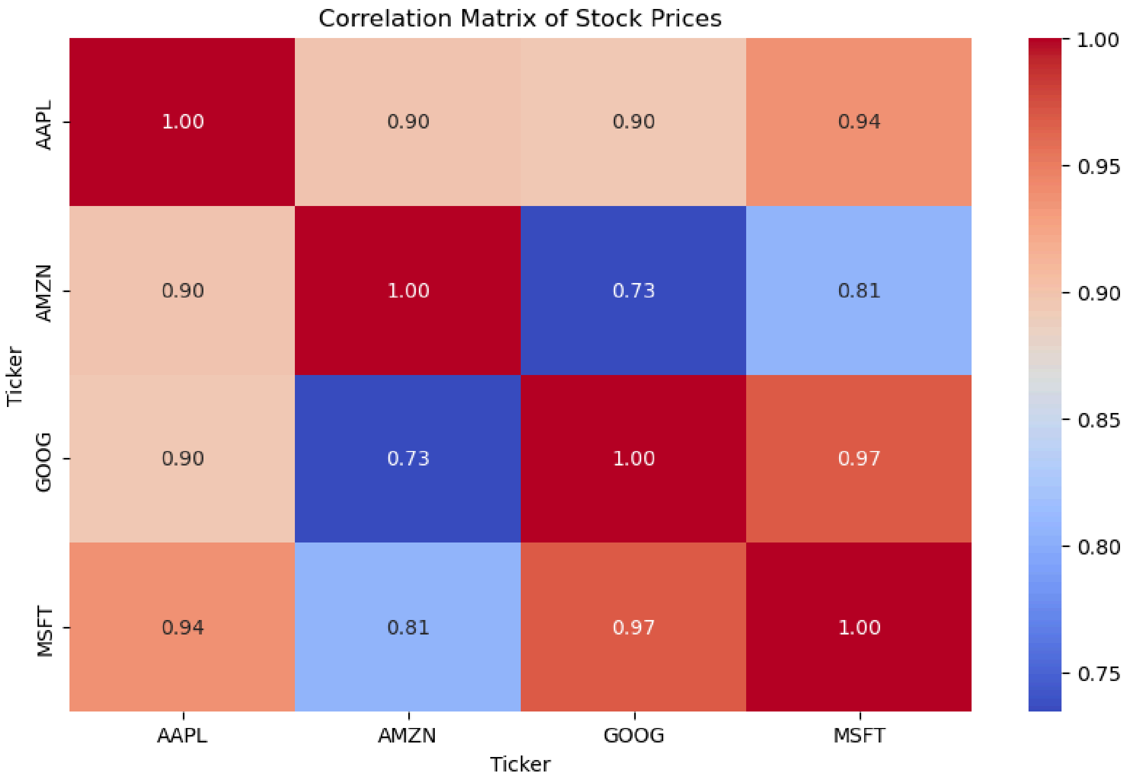
Box Plots: We'll use box plots to identify outliers and compare distributions between different stocks.

In [9]:
```
# Plot box plots
stock_data['Adj Close'].plot(kind='box', figsize=(10, 6))
plt.title('Box Plot of Stock Prices')
plt.ylabel('Stock Price (USD)')
plt.grid(True)
plt.show()
```



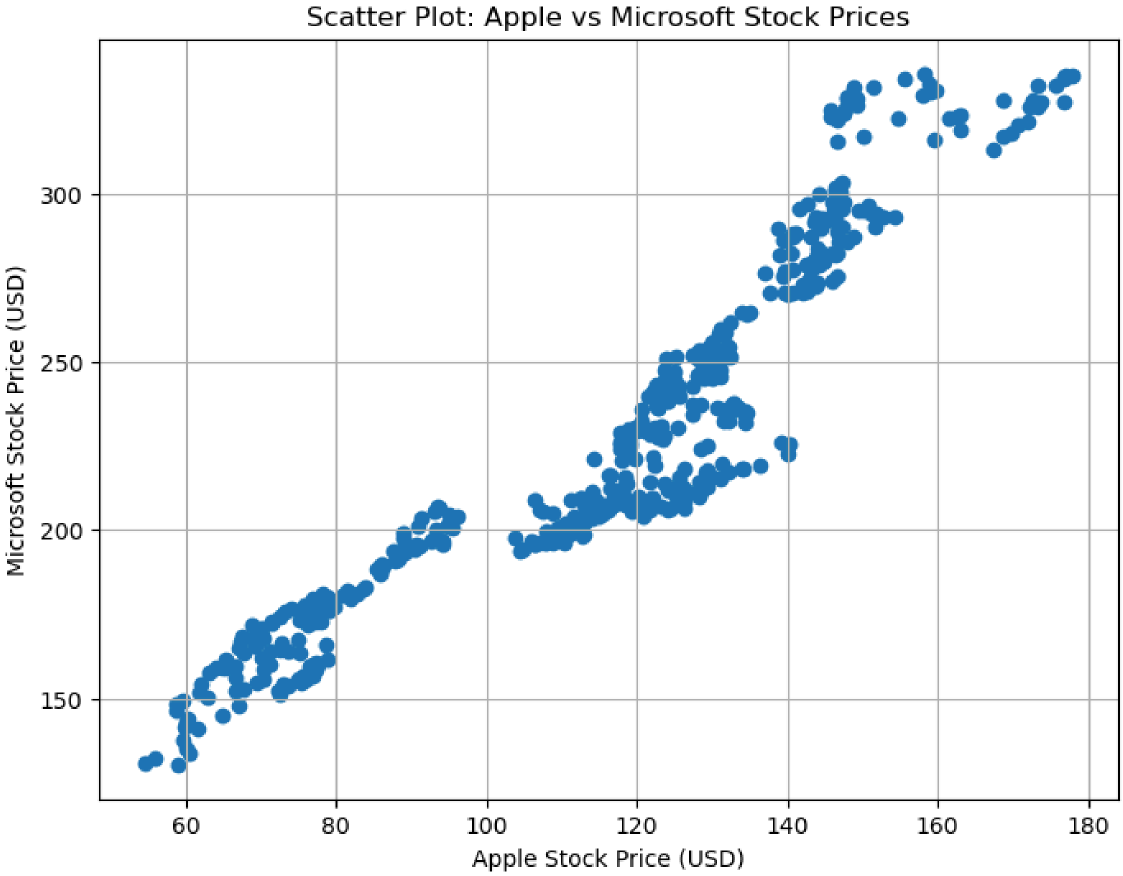
We'll explore the correlation between the stock prices of the selected companies in more detail. For this, we Visualize the correlation matrix using a heatmap. This heatmap will show the correlation coefficients between different stocks. Positive values indicate a positive correlation (movement in the same direction), while negative values indicate a negative correlation (movement in opposite directions). Values closer to 1 or -1 indicate stronger correlations.

In [10]:
```
import seaborn as sns

# Visualize correlation matrix using heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Stock Prices')
plt.show()
```



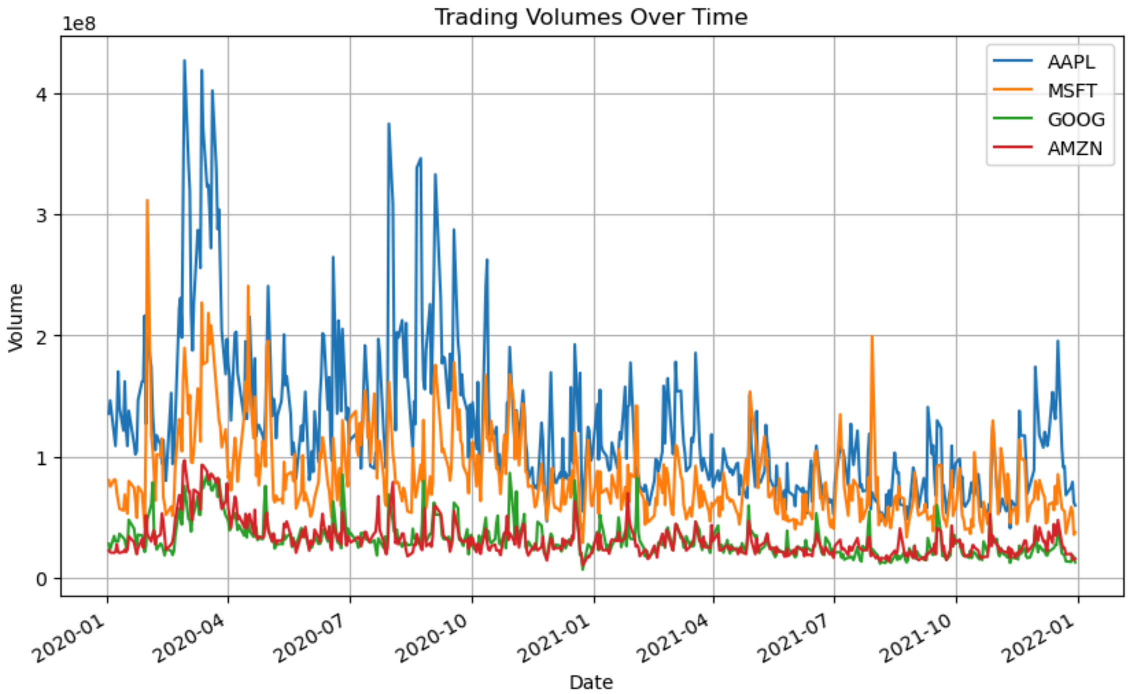
Identify highly correlated pairs of stocks and analyze their relationships using scatter plots

In [11]:
```
# Scatter plot between Apple and Microsoft stock prices
plt.figure(figsize=(8, 6))
plt.scatter(stock_data['Adj Close']['AAPL'], stock_data['Adj Close']['MSFT'
plt.title('Scatter Plot: Apple vs Microsoft Stock Prices')
plt.xlabel('Apple Stock Price (USD)')
plt.ylabel('Microsoft Stock Price (USD)')
plt.grid(True)
plt.show()
```



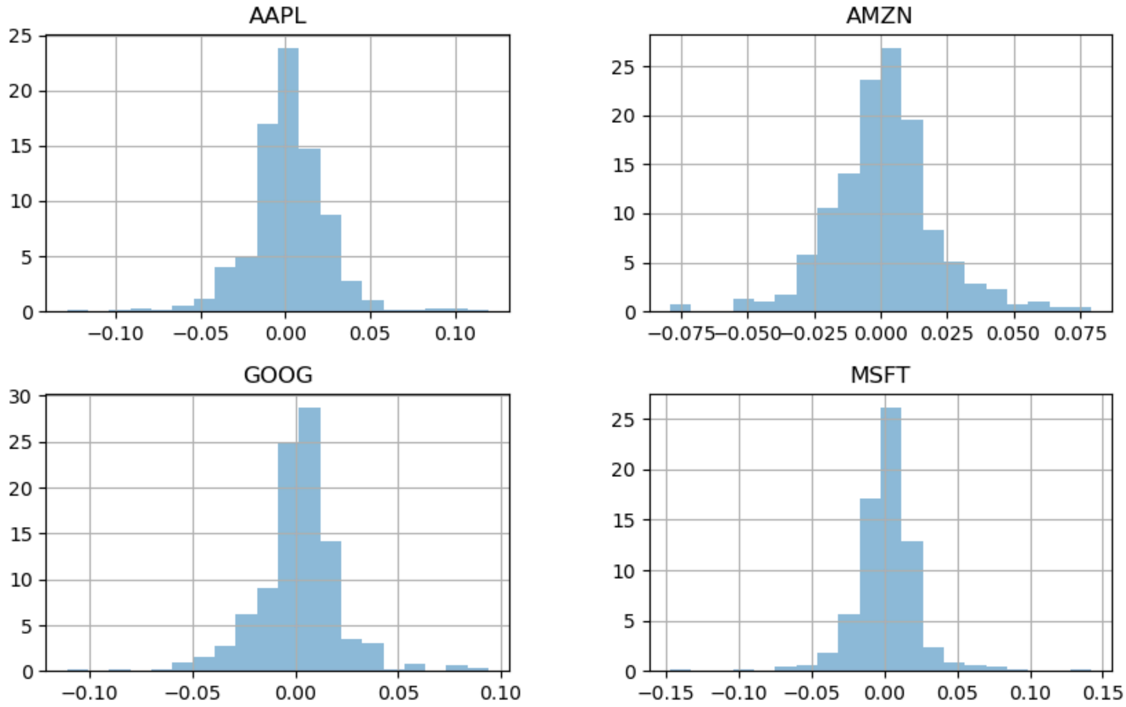
Analyze trading volumes: Visualize the trading volumes for each company to understand the level of trading activity.

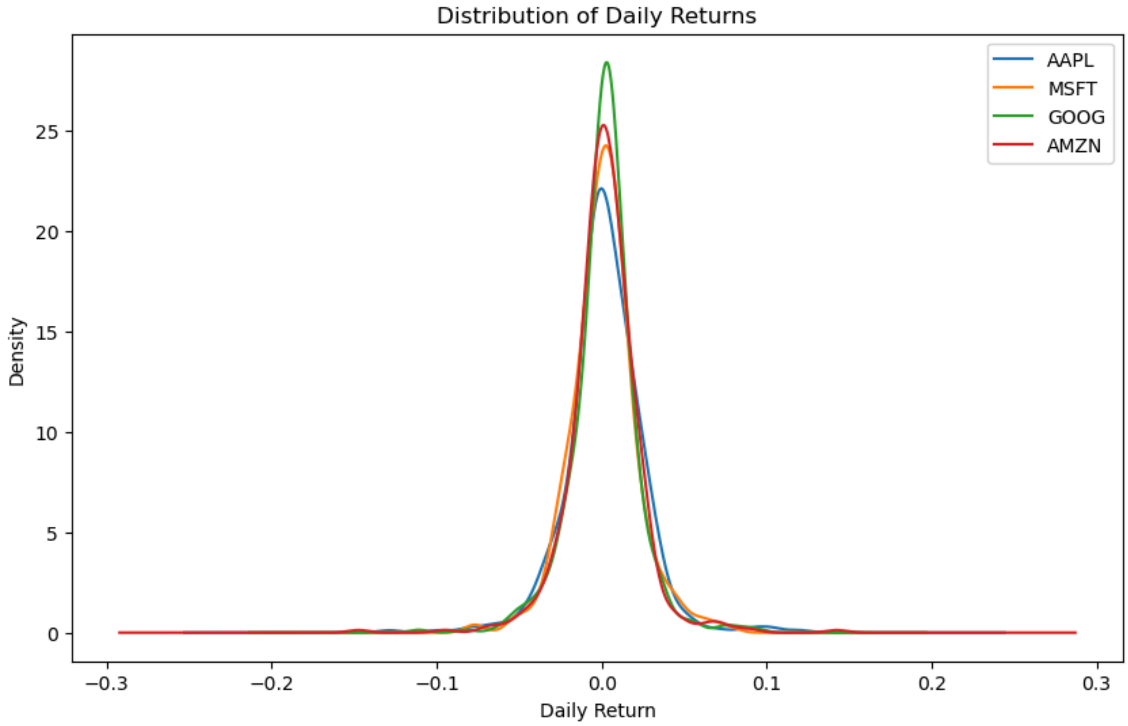
In [12]:
```
# Plot trading volumes
stock_data['Volume'].plot(figsize=(10, 6))
plt.title('Trading Volumes Over Time')
plt.xlabel('Date')
plt.ylabel('Volume')
plt.legend(companies)
plt.grid(True)
plt.show()
```

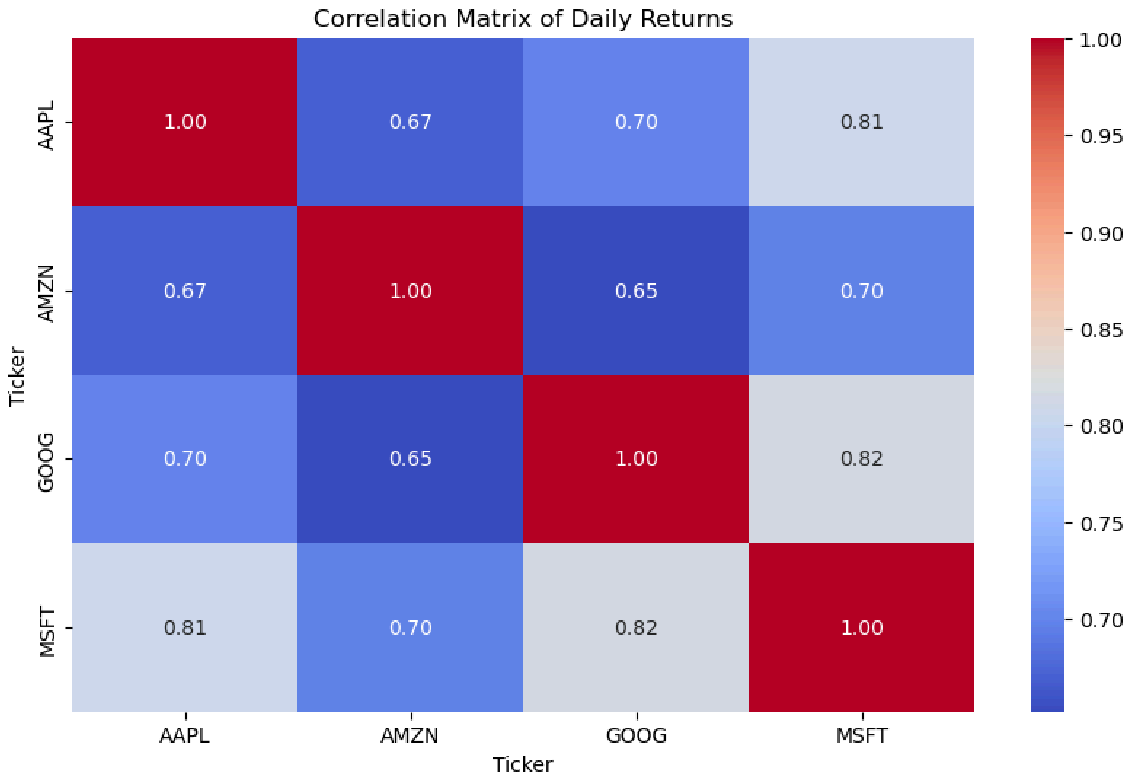


Calculate and analyze daily returns: Compute the daily returns for each stock and analyze their distributions and correlations.

In [13]:
```
# Calculate daily returns
daily_returns = stock_data['Adj Close'].pct_change()

# Plot daily returns distributions
daily_returns.hist(bins=20, figsize=(10, 6), density=True, alpha=0.5)
daily_returns.plot(kind='kde', figsize=(10, 6))
plt.title('Distribution of Daily Returns')
plt.xlabel('Daily Return')
plt.ylabel('Density')
plt.legend(companies)
plt.show()

# Compute correlation matrix of daily returns
correlation_matrix_returns = daily_returns.corr()

# Visualize correlation matrix of daily returns using heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix_returns, annot=True, cmap='coolwarm', fmt=".
plt.title('Correlation Matrix of Daily Returns')
plt.show()
```




Distribution of Daily Returns
AAPL
MSFT
GOOG
AMZN
Density
Daily Return
0
5
10
15
20
25
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3


Correlation Matrix of Daily Returns
Ticker
AAPL
AMZN
GOOG
MSFT
1.00
0.67
0.70
0.81
0.67
1.00
0.65
0.70
0.70
0.65
1.00
0.82
0.81
0.70
0.82
1.00
0.70
0.75
0.80
0.85
0.90
0.95
1.00

In [14]:
```python
# Importing necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Selecting independent variables (trading volumes)
independent_vars = ['Volume']

# Extracting independent and dependent variables
X = stock_data[independent_vars]
y = stock_data['Adj Close']

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra

# Fitting the regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Making predictions
y_pred = model.predict(X_test)

# Evaluating the model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 1034.5376626372279
```

In [15]:
```
# Fetching historical data for S&P 500 and NASDAQ indices
index_data = yf.download(['^GSPC', '^IXIC'], start='2020-01-01', end='2021-

# Print index data to verify availability
print(index_data)

# Extracting adjusted close prices of indices
index_data_close = index_data['Adj Close']

# Merging index data with stock data with consistent suffixes
stock_data = pd.merge(stock_data, index_data_close, left_index=True, right_

# Print stock data after merging
print(stock_data)

# Selecting additional independent variables with consistent suffixes
additional_independent_vars = ['^GSPC_Index', '^IXIC_Index']  # S&P 500 and

# Extracting additional independent variables
X_additional = stock_data[additional_independent_vars]

# Concatenating additional independent variables with existing ones
X = pd.concat([X, X_additional], axis=1)
```

```
[*********************100%%**********************]  2 of 2 completed
C:\Users\sande\AppData\Local\Temp\ipykernel_18032\634464011.py:11: Futu
reWarning: merging between different levels is deprecated and will be r
emoved in a future version. (2 levels on the left, 1 on the right)
  stock_data = pd.merge(stock_data, index_data_close, left_index=True,
right_index=True, suffixes=('', '_Index'))

Price         Adj Close                         Close
High  \
Ticker            ^GSPC          ^IXIC          ^GSPC          ^IXIC
^GSPC
Date
2020-01-02  3257.850098   9092.190430  3257.850098   9092.190430  3258.
139893
2020-01-03  3234.850098   9020.769531  3234.850098   9020.769531  3246.
149902
2020-01-06  3246.280029   9071.469727  3246.280029   9071.469727  3246.
840088
2020-01-07  3237.179932   9068.580078  3237.179932   9068.580078  3244.
909912
```

In [ ]:

In [16]:
```
# Create a list of tuples containing variable names and coefficients
coefficients_list = [(variable, coefficient) for variable, coefficient in z

# Create DataFrame from the list of tuples
coefficients_df = pd.DataFrame(coefficients_list, columns=['Variable', 'Coe

# Display the DataFrame
print("Model Coefficients:")
print(coefficients_df)

# Display the intercept
print("Intercept:", model.intercept_)
```

```
Model Coefficients:
         Variable   Coefficient
0  (Volume, AAPL) -9.357792e-08
1  (Volume, AMZN) -5.126573e-08
2  (Volume, GOOG) -3.213529e-07
3  (Volume, MSFT) -4.221551e-07
Intercept: [155.18431321 183.43634021 139.21791483 293.95811243]
```

In [17]:
```
# Calculate R-squared
r_squared = model.score(X_test, y_test)
print("R-squared:", r_squared)

# Calculate Adjusted R-squared
n = len(X_test)
p = len(X.columns)
adjusted_r_squared = 1 - (1 - r_squared) * ((n - 1) / (n - p - 1))
print("Adjusted R-squared:", adjusted_r_squared)
```

```
R-squared: 0.3214337752524574
Adjusted R-squared: 0.29316018255464305
```

In [ ]:

In [ ]: