## I .Introduction

Stock price prediction has long been a focal point in the world of finance, captivating investors, traders, and financial analysts alike. The ability to foresee future stock prices with a high degree of accuracy is an invaluable asset in making informed investment decisions. Traditionally, forecasting stock prices has relied on fundamental analysis, technical indicators, and statistical methods. However, as the financial markets evolve and the volume of data generated grows exponentially, traditional methods often fall short in capturing the complex patterns and subtleties present in stock price movements.

In this era of rapid technological advancements, we stand on the precipice of a new frontier in stock price prediction. Leveraging the power of advanced deep learning techniques, including Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM) networks, and attention mechanisms, we embark on a journey to revolutionize the way we forecast stock prices.

This endeavor seeks not only to predict stock prices with higher accuracy but also to uncover previously undetectable trends, correlations, and patterns that have the potential to redefine investment strategies. The dataset provided to us from Kaggle, containing Microsoft's lifetime stock prices, serves as the foundation for our exploration into the application of cutting-edge deep learning techniques in stock price prediction.

This dataset is a valuable resource with a rich history of stock price movements, enabling us to develop, test, and validate innovative models that can be deployed in real-world trading scenarios. In this document, we will detail the complete process of transitioning from a traditional approach to stock price prediction to an innovative, deep learning-based methodology.

We will walk through each step of our journey, from data preprocessing and feature engineering to model development and evaluation. We will also explore the nuances of combining CNN, LSTM, and attention mechanisms in our model architecture, aiming for superior predictive performance. Our pursuit is to unlock the potential of deep learning in forecasting stock prices, thereby providing investors and financial professionals with a robust and sophisticated tool for making more informed decisions in the ever-dynamic financial markets. Through this innovative approach, we aim to enhance our understanding of the intricacies of stock price movements and contribute to the evolution of financial analytics.

**Importing Libraries:**

- from mpl_toolkits.mplot3d import Axes3D: Imports the Axes3D module from the mpl_toolkits.mplot3d package. This module is used for creating 3D plots.
- from sklearn.preprocessing import StandardScaler: Imports the StandardScaler class from scikit-learn, a library for machine learning and data preprocessing.
- import matplotlib.pyplot as plt: Imports the Matplotlib library, commonly used for creating plots and charts.
- import numpy as np: Imports the NumPy library, which provides support for numerical operations and data manipulation.
- import os: Imports the os module, which provides functions for interacting with the operating system.
- import pandas as pd: Imports the Pandas library, used for data manipulation and analysis.

code:

```
 from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
```

## Providing file paths

os.walk('MSFT.csv'): This code initiates a directory traversal using the os.walk() function. It starts at the directory specified as 'MSFT.csv'. The os.walk() function returns an iterable that generates a sequence of directory names, lists of subdirectories, and filenames. Specifically, it returns a tuple for each directory it encounters, containing three values:

- The current directory path (string): dirname
- A list of subdirectory names (strings): _
- A list of filenames (strings): filenames

Looping Through Directory Structure:

- for dirname, _, filenames in os.walk('MSFT.csv'):: This loop iterates through the directory structure starting from 'MSFT.csv'. It captures the current directory path in dirname, ignores the list of subdirectories (denoted by _), and captures the list of filenames in filenames.

Printing File Paths:

- print(os.path.join(dirname, filename)): This line of code prints the complete file paths by joining the dirname and filename using the os.path.join() function. It prints the full paths of all the files found within the directory and its subdirectories.

Code:

```
for dirname, _, filenames in os.walk('MSFT.csv'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

**Reading dataset:**

```
nRowsRead = 1000
df1 = pd.read_csv('MSFT.csv', delimiter=',', nrows = nRowsRead)
df1.dataframeName = 'MSFT.csv'
nRow, nCol = df1.shape
print(f'There are {nRow} rows and {nCol} columns')
```

Output;

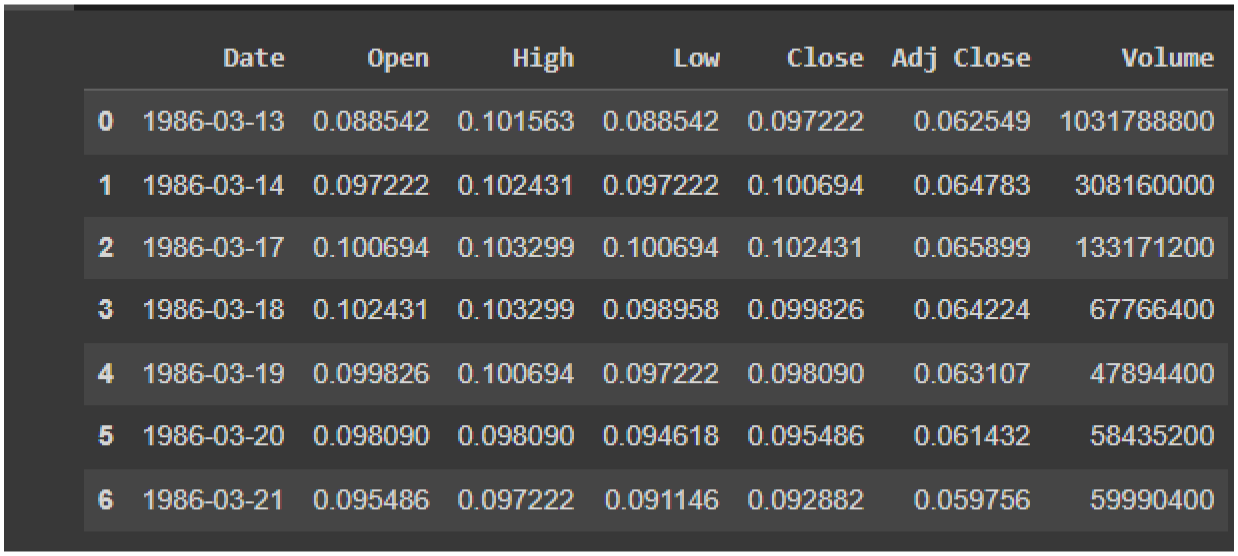
```
There are 1000 rows and 7 columns
```

**Displaying the specified header content of csv:**

#df1.head(7) is used to display the first 7 rows of the Pandas DataFrame df1

```
df1.head(7)
```

Output:



|   | Date | Open | High | Low | Close | Adj Close | Volume |
|---|---|---|---|---|---|---|---|
| 0 | 1986-03-13 | 0.088542 | 0.101563 | 0.088542 | 0.097222 | 0.062549 | 1031788800 |
| 1 | 1986-03-14 | 0.097222 | 0.102431 | 0.097222 | 0.100694 | 0.064783 | 308160000 |
| 2 | 1986-03-17 | 0.100694 | 0.103299 | 0.100694 | 0.102431 | 0.065899 | 133171200 |
| 3 | 1986-03-18 | 0.102431 | 0.103299 | 0.098958 | 0.099826 | 0.064224 | 67766400 |
| 4 | 1986-03-19 | 0.099826 | 0.100694 | 0.097222 | 0.098090 | 0.063107 | 47894400 |
| 5 | 1986-03-20 | 0.098090 | 0.098090 | 0.094618 | 0.095486 | 0.061432 | 58435200 |
| 6 | 1986-03-21 | 0.095486 | 0.097222 | 0.091146 | 0.092882 | 0.059756 | 59990400 |

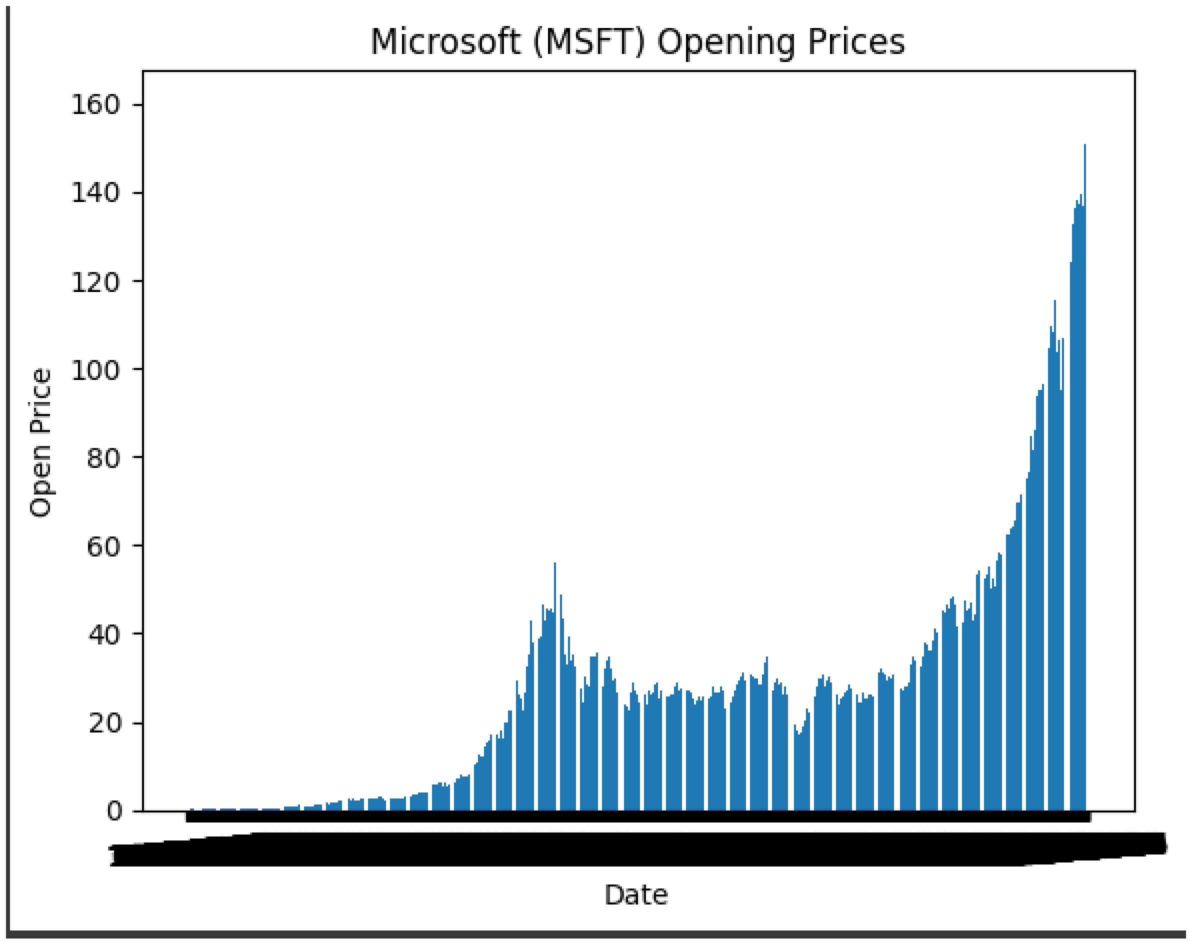
## DATA VISUALIZATION;

Pandas and Matplotlib libraries to create a bar chart that displays the opening prices of Microsoft (MSFT) stock over time.

CODE:

```
import pandas as pd
import matplotlib.pyplot as plt
file_path = 'MSFT.csv'
df = pd.read_csv(file_path)
x = df['Date']
y = df['Open']
plt.bar(x, y)
plt.xlabel('Date')
plt.ylabel('Open Price')
plt.title('Microsoft (MSFT) Opening Prices')
plt.xticks(rotation=5)
plt.show()
```

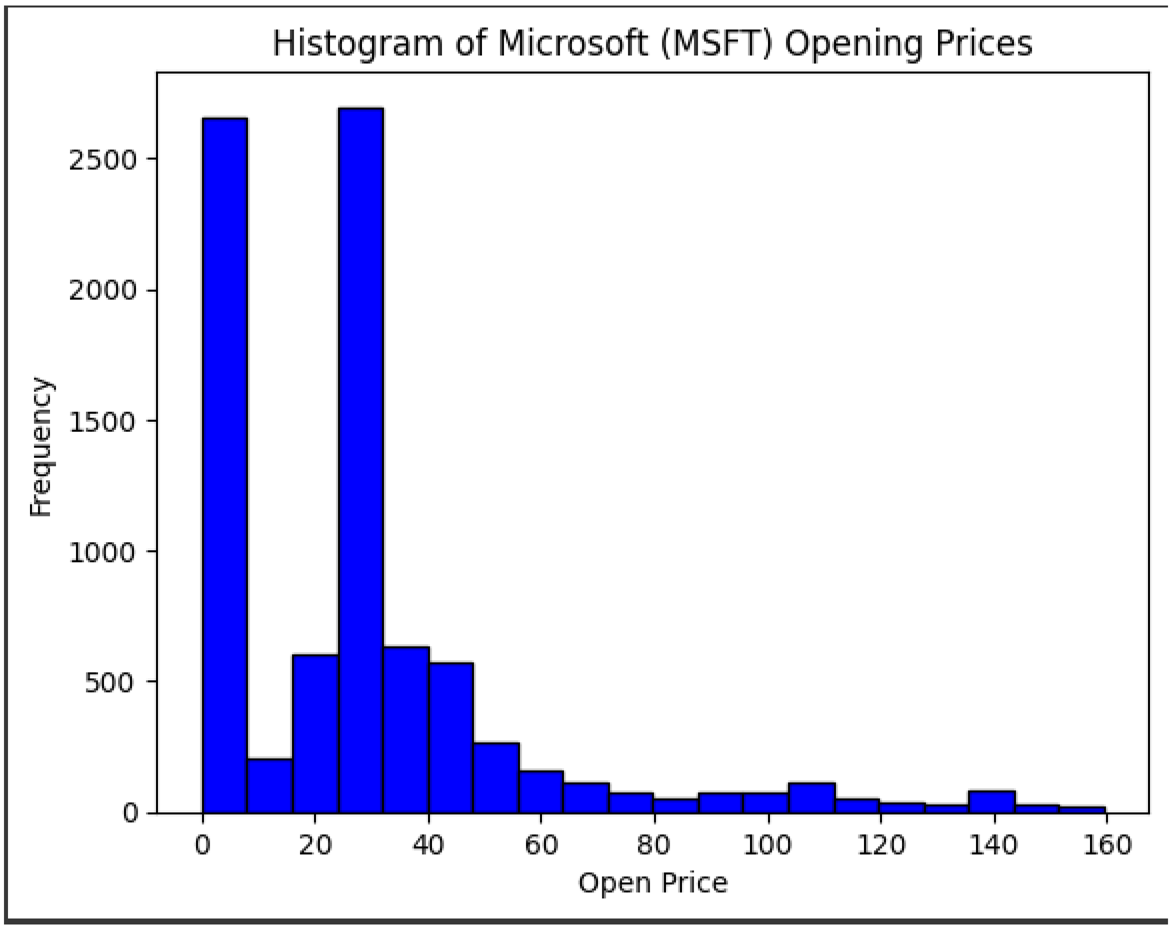
OUTPUT:



**code;**

```
import pandas as pd
import matplotlib.pyplot as plt
file_path = 'MSFT.csv'
df = pd.read_csv(file_path)
data = df['Open']
plt.hist(data, bins=20, color='blue', edgecolor='black')
plt.xlabel('Open Price')
plt.ylabel('Frequency')
plt.title('Histogram of Microsoft (MSFT) Opening Prices')
plt.show()
```

Output:


Histogram of Microsoft (MSFT) Opening Prices
Frequency
0
500
1000
1500
2000
2500
0
20
40
60
80
100
120
140
160
Open Price

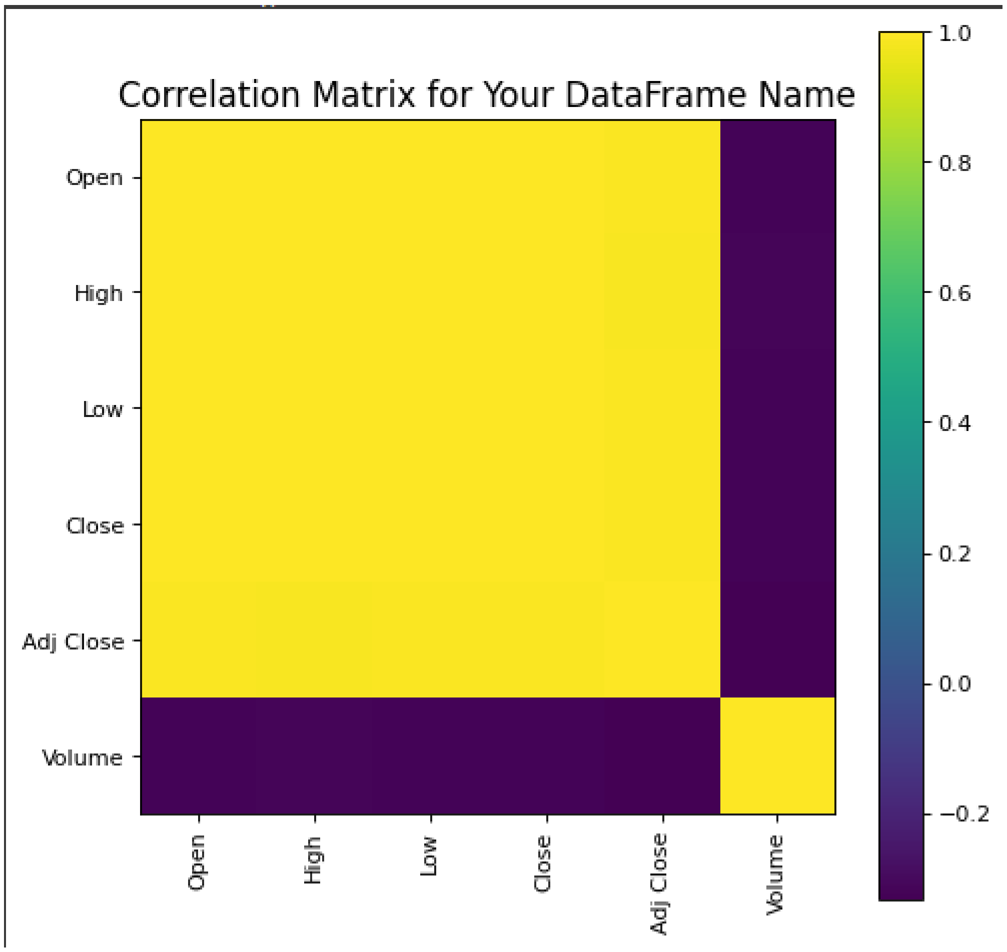
**Plot correlation matrix;**

The code defines a Python function called plotCorrelationMatrix and uses it to generate and display a correlation matrix plot for a given DataFram

***Code:***

```
import pandas as pd
import matplotlib.pyplot as plt
def plotCorrelationMatrix(df, graphWidth):
    filename = df.dataframeName
    df = df.dropna('columns')
    df = df[[col for col in df if df[col].nunique() > 1]]
    if df.shape[1] < 2:
        print(f'No correlation plots shown: The number of non-NaN or constant
columns ({df.shape[1]}) is less than 2')
        return
    corr = df.corr()
    plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, facecolor='w',
edgecolor='k')
    corrMat = plt.matshow(corr, fignum=1)
    plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
    plt.yticks(range(len(corr.columns)), corr.columns)
    plt.gca().xaxis.tick_bottom()
    plt.colorbar(corrMat)
    plt.title(f'Correlation Matrix for {filename}', fontsize=15)
    plt.show()
file_path = 'MSFT.csv'
df = pd.read_csv(file_path)
df.dataframeName = 'Your DataFrame Name'
graphWidth = 7
plotCorrelationMatrix(df, graphWidth)
```

**Output:**


Correlation Matrix for Your DataFrame Name
Open
High
Low
Close
Adj Close
Volume
Open
High
Low
Close
Adj Close
Volume
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

**SCATTER PLOT AND DENSITY PLOT:**

The provided code defines a Python function called plotScatterMatrix and uses it to create a scatter matrix plot with density plots for numerical columns in a given DataFrame.

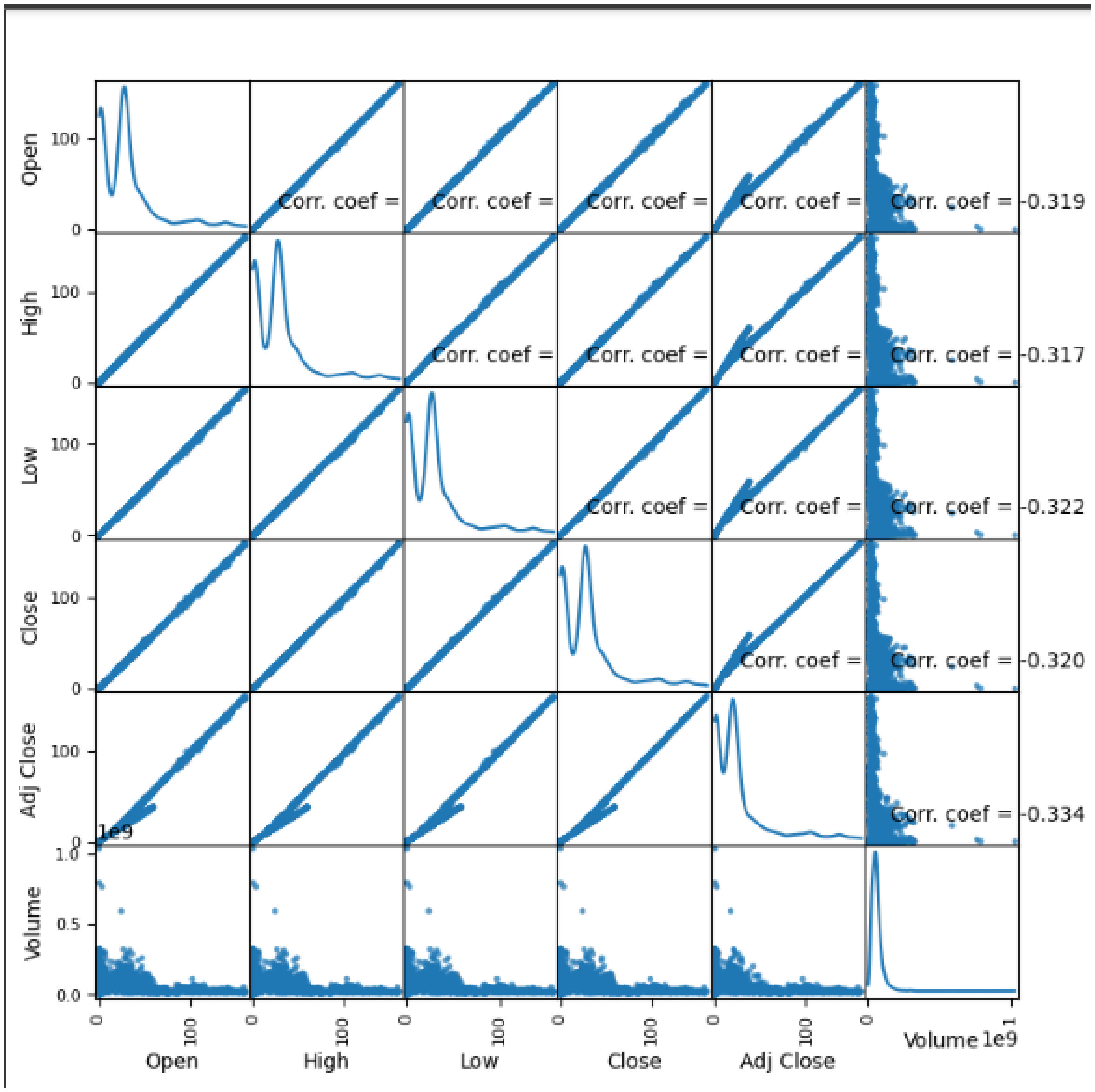
**CODE;**

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
def plotScatterMatrix(df, plotSize, textSize):
    df = df.select_dtypes(include=[np.number])
    df = df.dropna('columns')
    df = df[[col for col in df if df[col].nunique() > 1]]

    columnNames = list(df)
    if len(columnNames) > 10:
        columnNames = columnNames[:10]

    df = df[columnNames]
    ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSize, plotSize],
diagonal='kde')
    corrs = df.corr().values
    for i, j in zip(*np.triu_indices_from(ax, k=1)):
        ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.8, 0.2), xycoords='axes
fraction', ha='center', va='center', size=textSize)
    plt.suptitle('Scatter and density Plot')
    plt.show()
file_path = 'MSFT.csv'
df = pd.read_csv(file_path)
plotSize = 8
textSize = 10
plotScatterMatrix(df, plotSize, textSize)
```

**Output:**



Open
High
Low
Close
Adj Close
Volume
Corr. coef = -0.319
Corr. coef = -0.317
Corr. coef = -0.322
Corr. coef = -0.320
Corr. coef = -0.334
1e9

## VIOLIN PLOT:

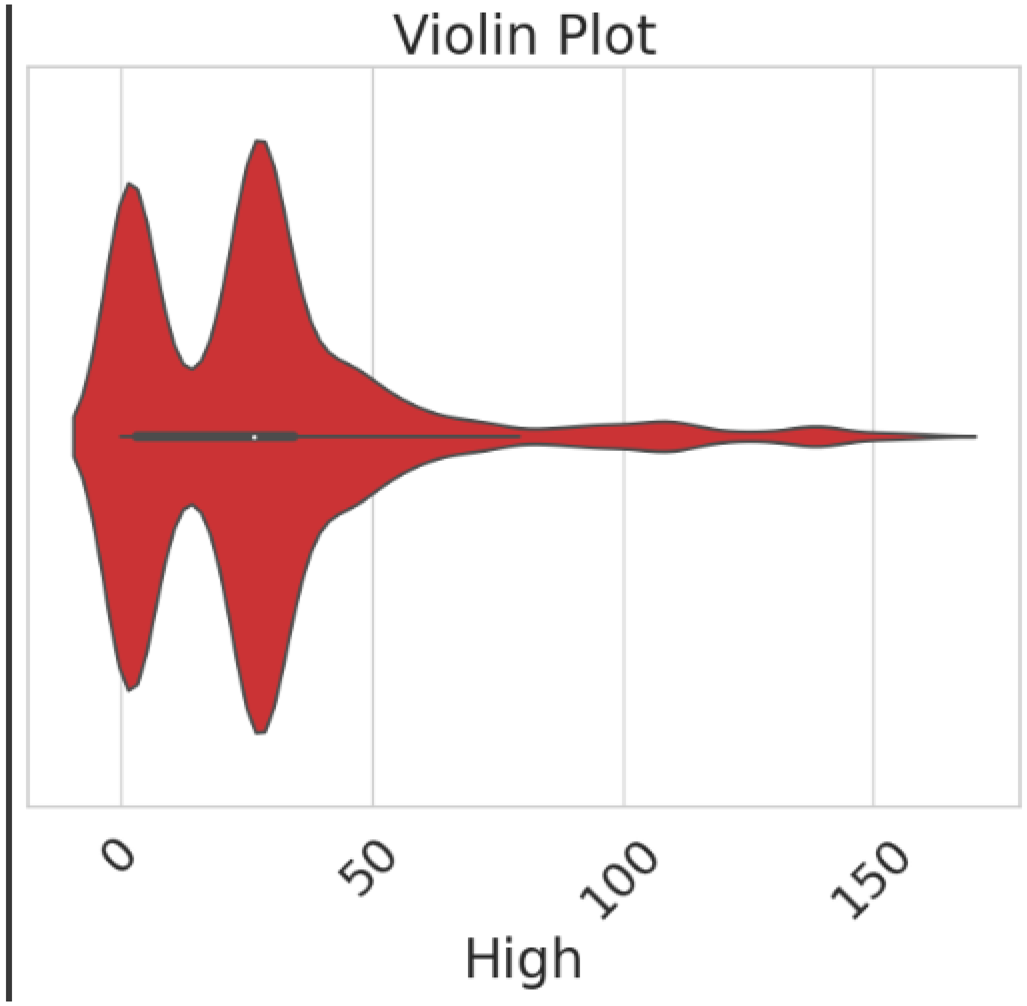
A violin plot is a data visualization that combines elements of a box plot and a kernel density plot. It is used to represent the distribution of a dataset, showing both the summary statistics (similar to a box plot) and the probability density of the data at different values.

**Code:**

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
def plotScatterMatrix(df, plotSize, textSize):

    numeric_df = df.select_dtypes(include=[np.number])
    numeric_df = numeric_df.dropna('columns')
    numeric_df = numeric_df.loc[:, numeric_df.nunique() > 1]
    if numeric_df.shape[1] > 10:
        numeric_df = numeric_df.iloc[:, :10]
    plt.figure(figsize=(plotSize, plotSize))
    sns.set(font_scale=textSize)
    sns.set_style("whitegrid")
    sns.pairplot(numeric_df, diag_kind='kde', kind='scatter', plot_kws={'alpha':
0.75})
    corrs = numeric_df.corr().values
    for i, j in zip(*np.triu_indices_from(corrs, k=1)):
    plt.suptitle('Scatter and Density Plot')
    plt.show()
```

Output:


Violin Plot
High