```
from google.colab import drive
```

+ Code + Text

```
pip install yfinance
```

```
Requirement already satisfied: yfinance in /usr/local/lib/python3.10/dist-packages (0.2.40)
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.0.3)
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.25.2)
Requirement already satisfied: requests>=2.31 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.31.0)
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.10/dist-packages (from yfinance) (0.0.11)
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.9.4)
Requirement already satisfied: platformdirs>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.2.2)
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2023.4)
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.10/dist-packages (from yfinance) (2.4.4)
Requirement already satisfied: peewee>=3.16.2 in /usr/local/lib/python3.10/dist-packages (from yfinance) (3.17.6)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (4.12.3)
Requirement already satisfied: html5lib>=1.1 in /usr/local/lib/python3.10/dist-packages (from yfinance) (1.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4>=4.11.1->yfinance) (2.5
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.10/dist-packages (from html5lib>=1.1->yfinance) (1.16.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.3.0->yfinance) (2.8
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.3.0->yfinance) (2024.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (2.0.7
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.31->yfinance) (2024.7
```

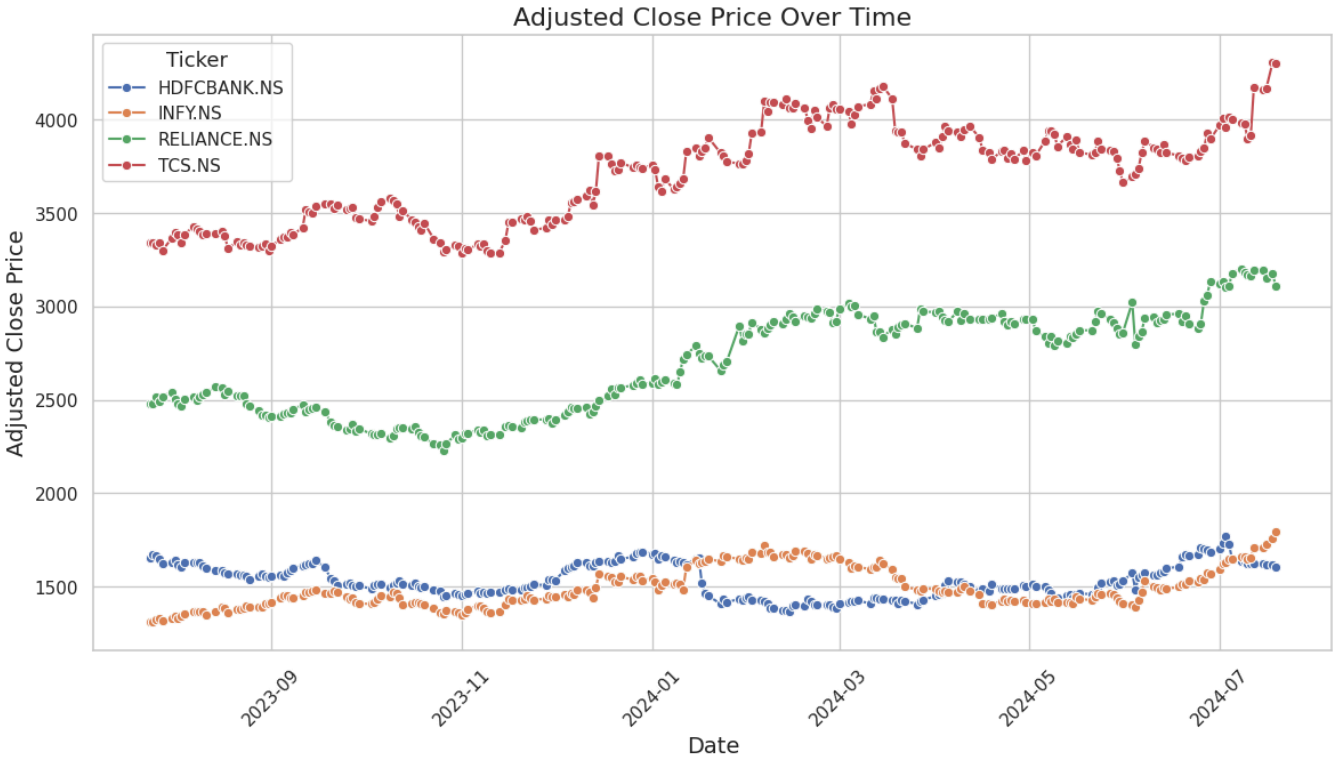
```
import pandas as pd
import yfinance as yf
from datetime import date, timedelta

# define the time period for the data
end_date = date.today().strftime("%Y-%m-%d")
start_date = (date.today() - timedelta(days=365)).strftime("%Y-%m-%d")

# list of stock tickers to download
tickers = ['RELIANCE.NS', 'TCS.NS', 'INFY.NS', 'HDFCBANK.NS']
```

```
data = yf.download(tickers, start=start_date, end=end_date, progress=False)

# reset index to bring Date into the columns for the melt function
data = data.reset_index()

# melt the DataFrame to make it long format where each row is a unique combination of Date, Ticker, and attributes
data_melted = data.melt(id_vars=['Date'], var_name=['Attribute', 'Ticker'])

# pivot the melted DataFrame to have the attributes (Open, High, Low, etc.) as columns
data_pivoted = data_melted.pivot_table(index=['Date', 'Ticker'], columns='Attribute', values='value', aggfunc='first')

# reset index to turn multi-index into columns
stock_data = data_pivoted.reset_index()

print(stock_data.head())
```

```
Attribute       Date       Ticker    Adj Close        Close         High  \
0         2023-07-24  HDFCBANK.NS  1655.789429  1678.400024  1684.650024
1         2023-07-24      INFY.NS  1312.125610  1336.599976  1349.900024
2         2023-07-24  RELIANCE.NS  2478.644287  2487.399902  2514.949951
3         2023-07-24       TCS.NS  3338.298096  3394.750000  3413.449951
4         2023-07-25  HDFCBANK.NS  1673.744263  1696.599976  1699.000000

Attribute          Low         Open      Volume
0          1670.099976  1678.500000  16089722.0
1          1334.250000  1341.000000   8859789.0
2          2469.300049  2481.000000  11863933.0
3          3372.100098  3381.000000   1680132.0
4          1678.400024  1684.650024  27996298.0
```

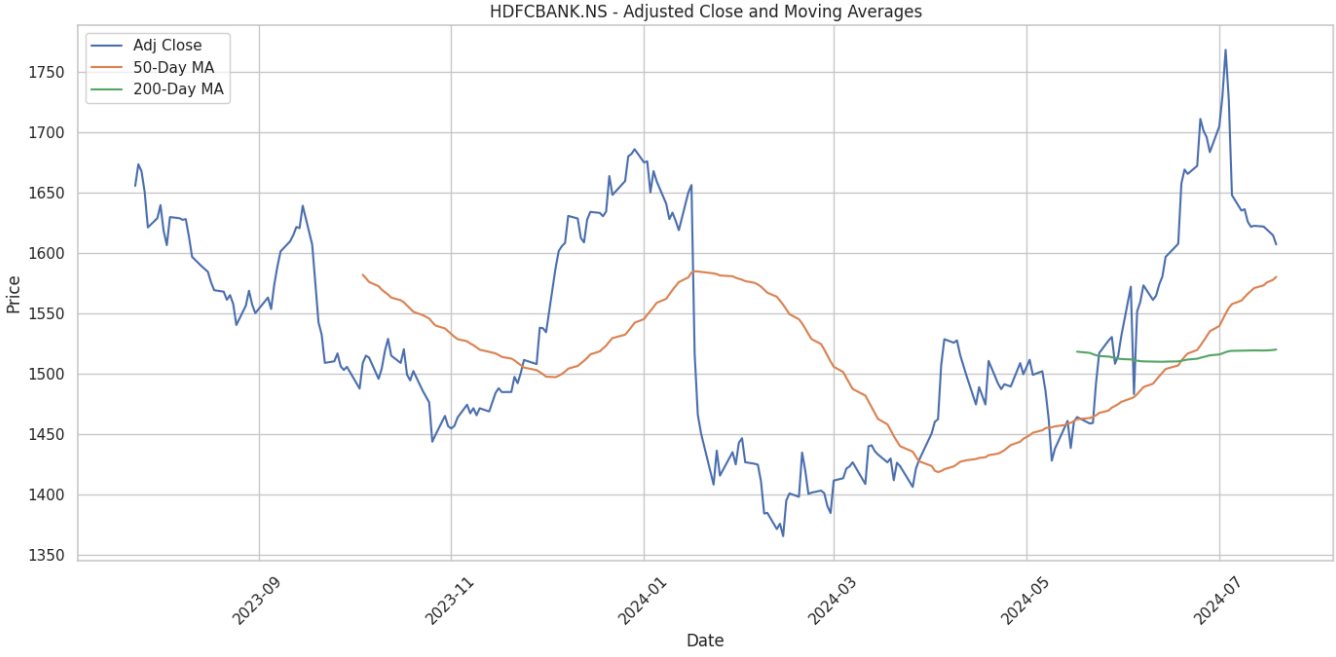
```
import matplotlib.pyplot as plt
import seaborn as sns

stock_data['Date'] = pd.to_datetime(stock_data['Date'])

stock_data.set_index('Date', inplace=True)
stock_data.reset_index(inplace=True)
plt.figure(figsize=(14, 7))
sns.set(style='whitegrid')

sns.lineplot(data=stock_data, x='Date', y='Adj Close', hue='Ticker', marker='o')

plt.title('Adjusted Close Price Over Time', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Adjusted Close Price', fontsize=14)
plt.legend(title='Ticker', title_fontsize='13', fontsize='11')
plt.grid(True)

plt.xticks(rotation=45)

plt.show()
```

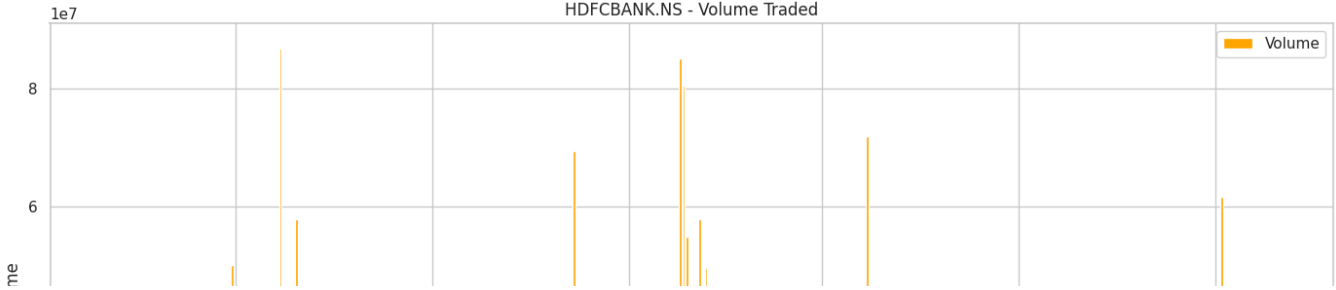
[+]

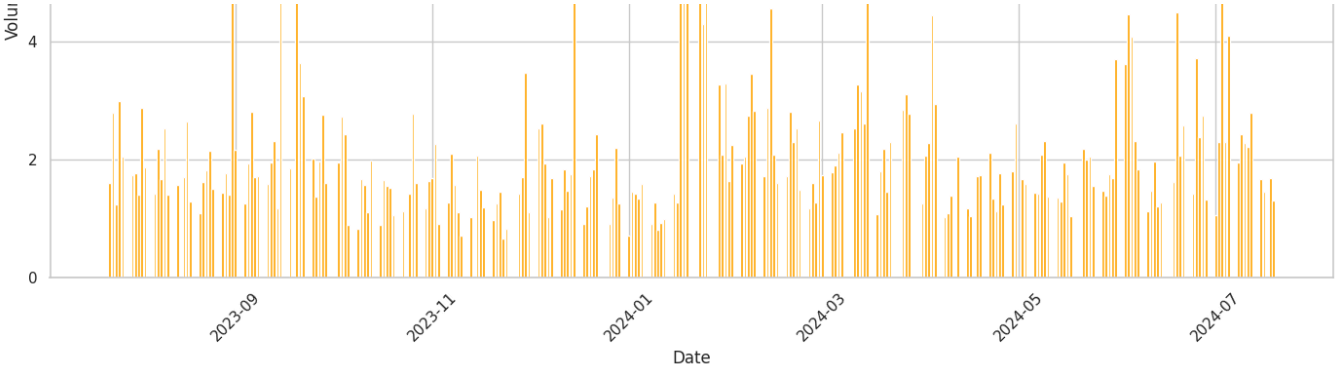

Adjusted Close Price Over Time
Ticker
HDFCBANK.NS
INFY.NS
RELIANCE.NS
TCS.NS
Adjusted Close Price
4000
3500
3000
2500
2000
1500
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
Date

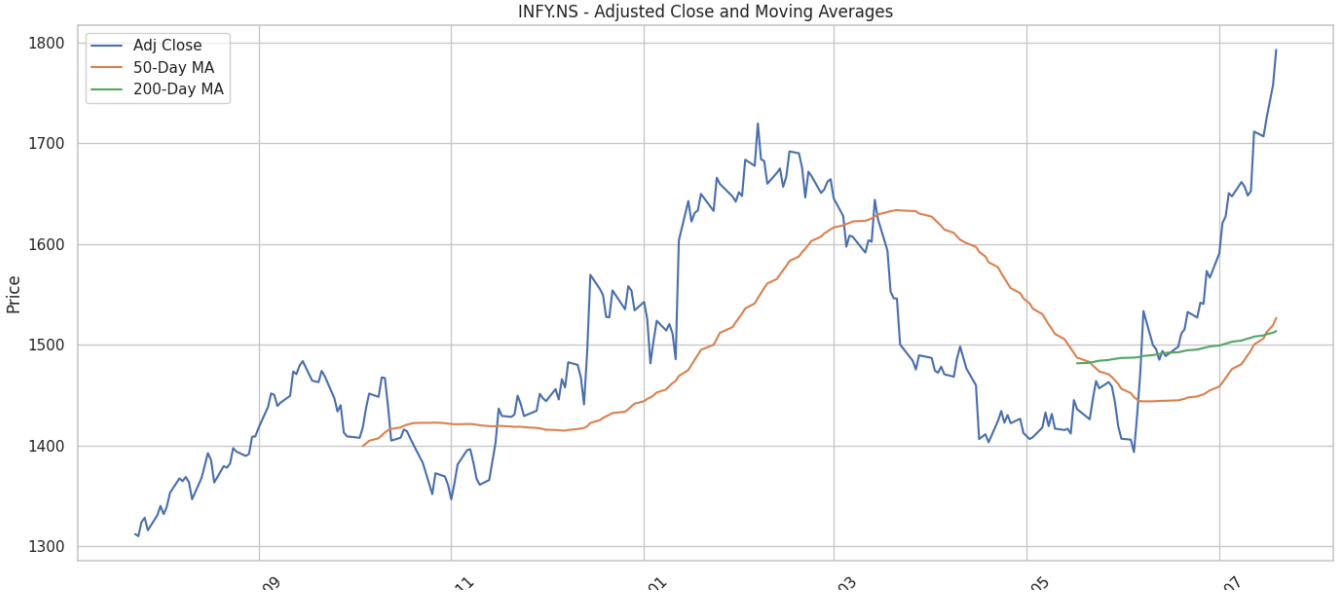
```
short_window = 50
long_window = 200

stock_data.set_index('Date', inplace=True)
unique_tickers = stock_data['Ticker'].unique()

for ticker in unique_tickers:
    ticker_data = stock_data[stock_data['Ticker'] == ticker].copy()
    ticker_data['50_MA'] = ticker_data['Adj Close'].rolling(window=short_window).mean()
    ticker_data['200_MA'] = ticker_data['Adj Close'].rolling(window=long_window).mean()

    plt.figure(figsize=(14, 7))
    plt.plot(ticker_data.index, ticker_data['Adj Close'], label='Adj Close')
    plt.plot(ticker_data.index, ticker_data['50_MA'], label='50-Day MA')
    plt.plot(ticker_data.index, ticker_data['200_MA'], label='200-Day MA')
    plt.title(f'{ticker} - Adjusted Close and Moving Averages')
    plt.xlabel('Date')
    plt.ylabel('Price')
    plt.legend()
    plt.grid(True)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(14, 7))
    plt.bar(ticker_data.index, ticker_data['Volume'], label='Volume', color='orange')
    plt.title(f'{ticker} - Volume Traded')
    plt.xlabel('Date')
    plt.ylabel('Volume')
    plt.legend()
    plt.grid(True)
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

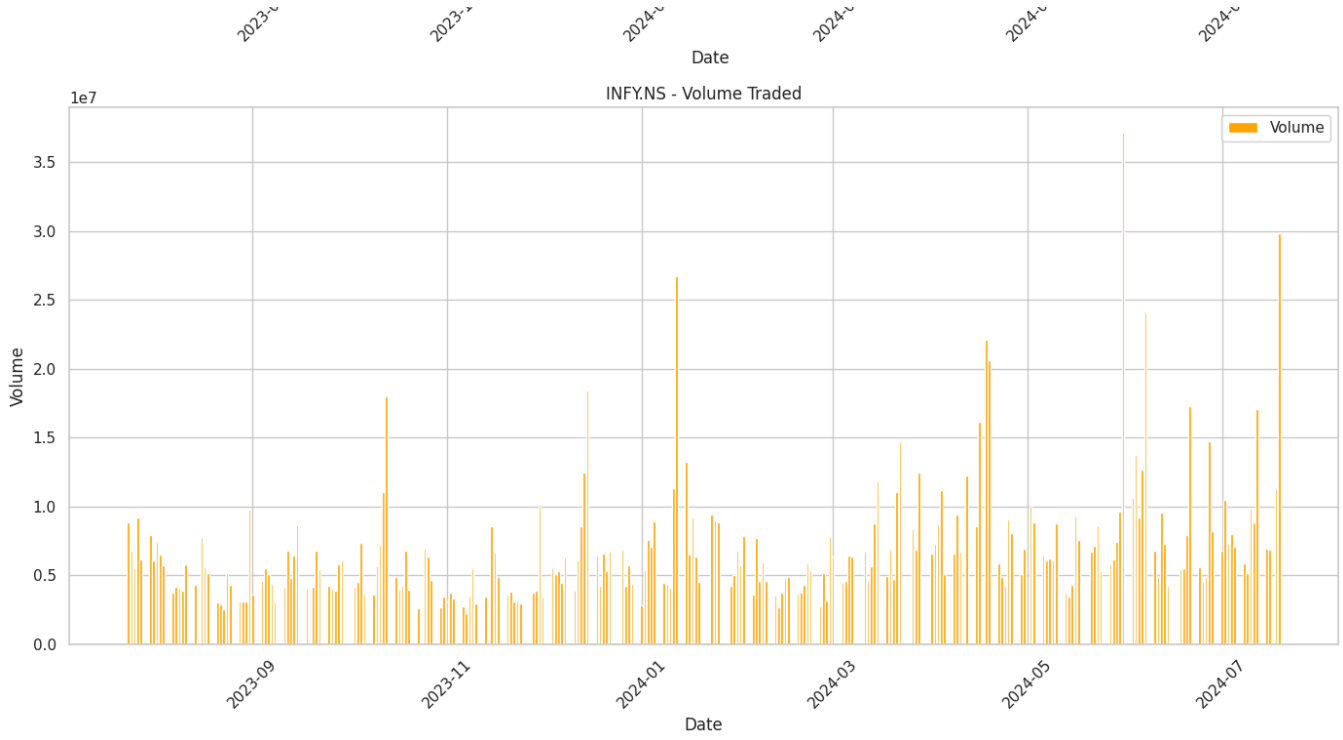
[4]

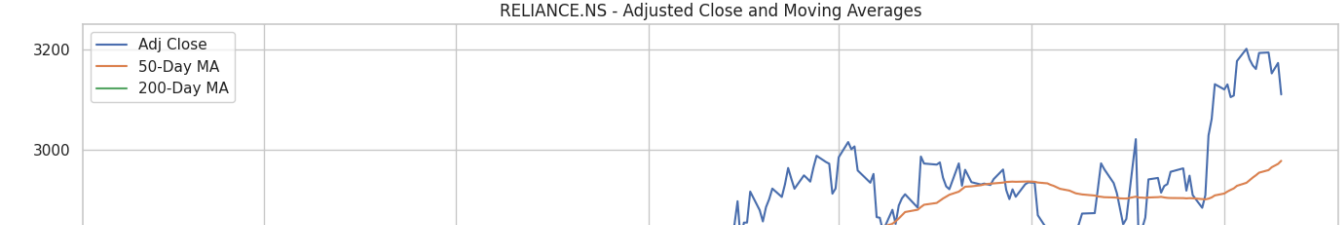

HDFCBANK.NS - Adjusted Close and Moving Averages
Adj Close
50-Day MA
200-Day MA
Price
1750
1700
1650
1600
1550
1500
1450
1400
1350
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
Date

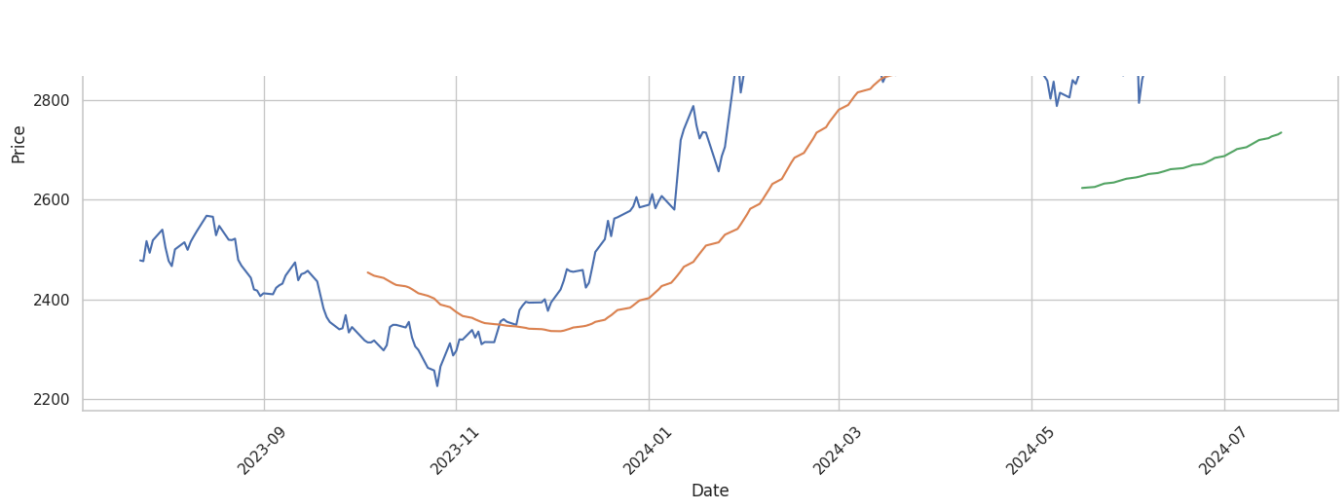

HDFCBANK.NS - Volume Traded
1e7
Volume
8
6

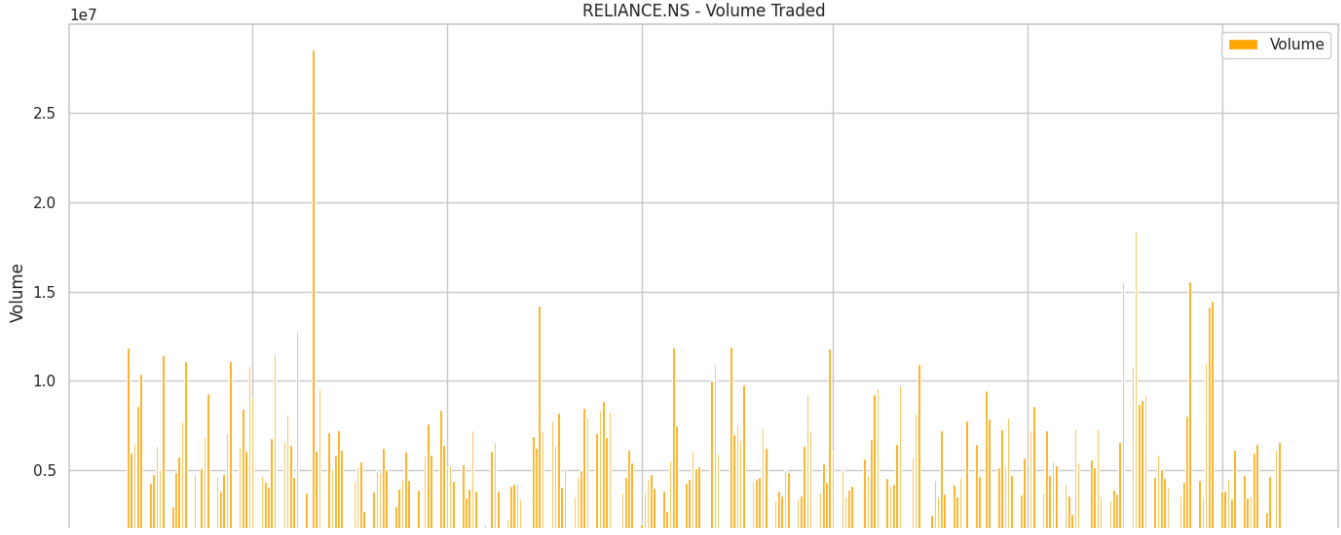

Volu
4
2
0
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
Date


INFY.NS - Adjusted Close and Moving Averages
Adj Close
50-Day MA
200-Day MA
Price
1800
1700
1600
1500
1400
1300


Date
INFY.NS - Volume Traded
1e7
Volume
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
Date

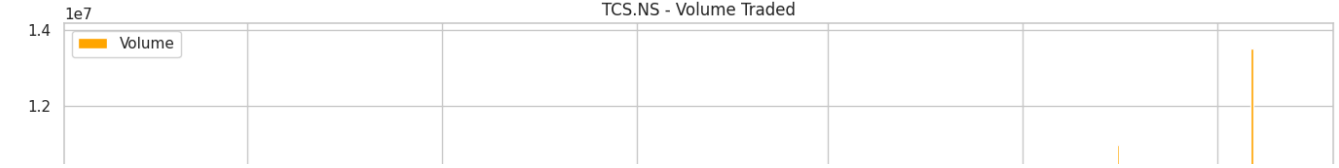

RELIANCE.NS - Adjusted Close and Moving Averages
Adj Close
50-Day MA
200-Day MA
3200
3000

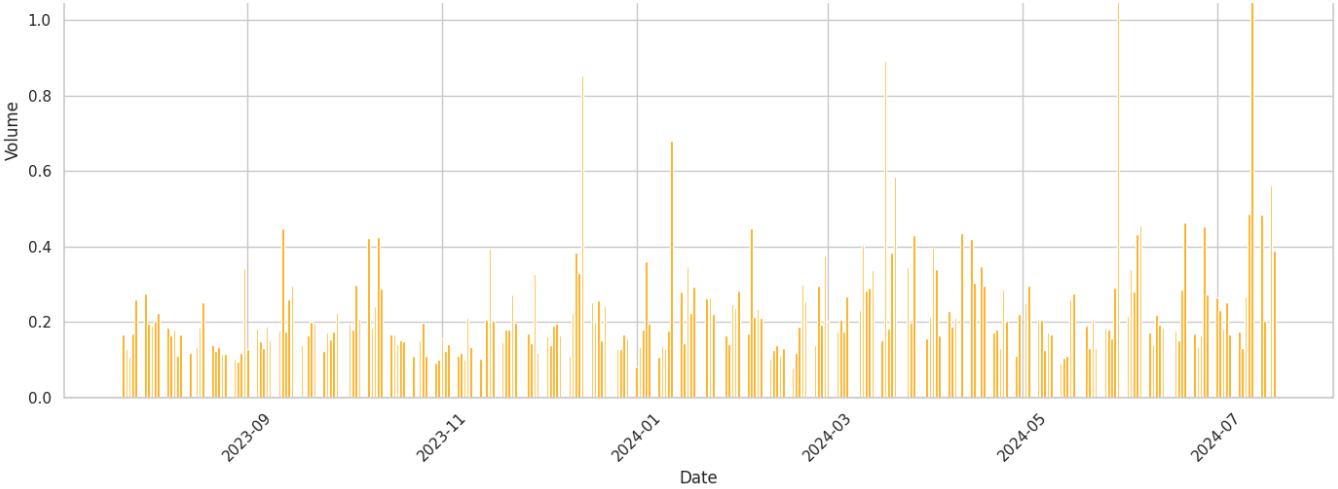

Price
2800
2600
2400
2200
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
Date


RELIANCE.NS - Volume Traded
1e7
Volume
2.5
2.0
1.5
1.0
0.5
Volume


0.0
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
Date

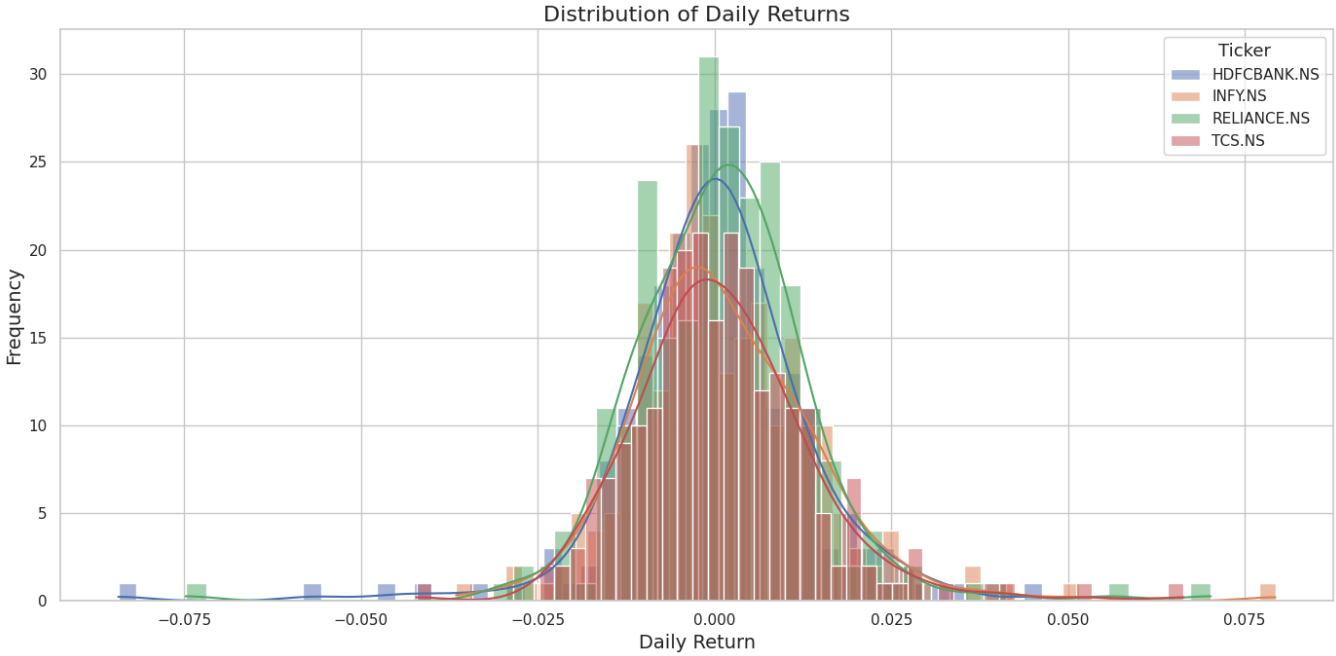

TCS.NS - Adjusted Close and Moving Averages
Adj Close
50-Day MA
200-Day MA
Price
4200
4000
3800
3600
3400
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
Date


TCS.NS - Volume Traded
1e7
1.4
1.2
Volume


Volume
1.0
0.8
0.6
0.4
0.2
0.0
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
Date

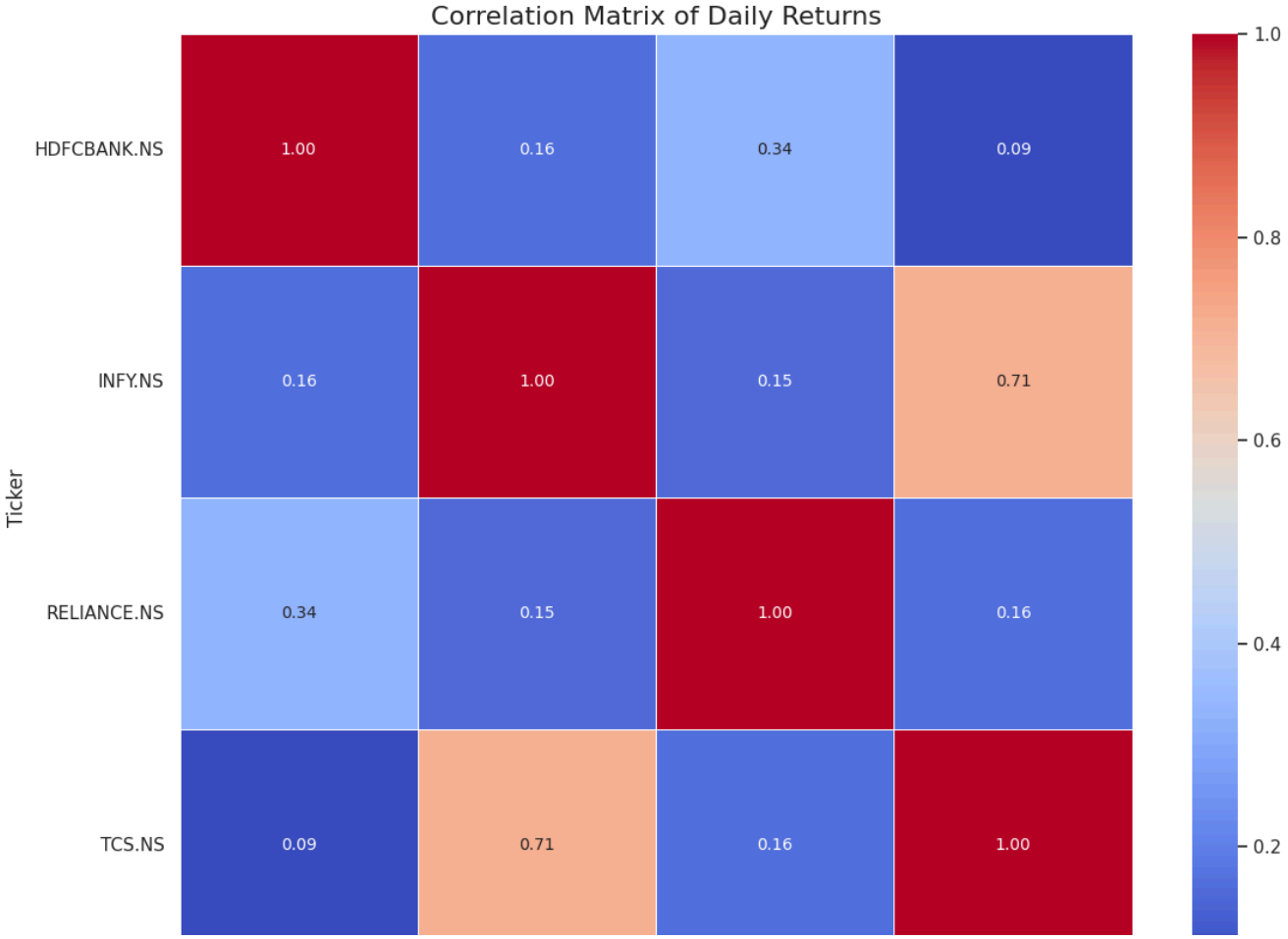
```
stock_data['Daily Return'] = stock_data.groupby('Ticker')['Adj Close'].pct_change()

plt.figure(figsize=(14, 7))
sns.set(style='whitegrid')

for ticker in unique_tickers:
    ticker_data = stock_data[stock_data['Ticker'] == ticker]
    sns.histplot(ticker_data['Daily Return'].dropna(), bins=50, kde=True, label=ticker, alpha=0.5)

plt.title('Distribution of Daily Returns', fontsize=16)
plt.xlabel('Daily Return', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.legend(title='Ticker', title_fontsize='13', fontsize='11')
plt.grid(True)
plt.tight_layout()
plt.show()
```

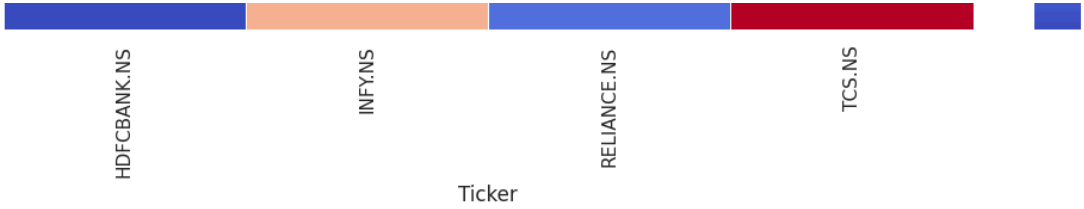
[+]


Distribution of Daily Returns
Ticker
HDFCBANK.NS
INFY.NS
RELIANCE.NS
TCS.NS
Frequency
Daily Return
0
5
10
15
20
25
30
−0.075
−0.050
−0.025
0.000
0.025
0.050
0.075

```
daily_returns = stock_data.pivot_table(index='Date', columns='Ticker', values='Daily Return')
correlation_matrix = daily_returns.corr()

plt.figure(figsize=(12, 10))
sns.set(style='whitegrid')

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=.5, fmt='.2f', annot_kws={"size": 10})
plt.title('Correlation Matrix of Daily Returns', fontsize=16)
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

[+]


Correlation Matrix of Daily Returns
Ticker
HDFCBANK.NS
INFY.NS
RELIANCE.NS
TCS.NS
1.00
0.16
0.34
0.09
0.16
1.00
0.15
0.71
0.34
0.15
1.00
0.16
0.09
0.71
0.16
1.00
1.0
0.8
0.6
0.4
0.2




```
import numpy as np

expected_returns = daily_returns.mean() * 252  # annualize the returns
volatility = daily_returns.std() * np.sqrt(252)  # annualize the volatility

stock_stats = pd.DataFrame({
    'Expected Return': expected_returns,
    'Volatility': volatility
})

stock_stats
```

Ticker	Expected Return	Volatility
HDFCBANK.NS	-0.008419	0.212027
INFY.NS	0.348923	0.212535
RELIANCE.NS	0.259098	0.208548
TCS.NS	0.285627	0.202289

Next steps: Generate code with `stock_stats` View recommended plots

```
# function to calculate portfolio performance
def portfolio_performance(weights, returns, cov_matrix):
    portfolio_return = np.dot(weights, returns)
    portfolio_volatility = np.sqrt(np.dot(weights.T, np.dot(cov_matrix, weights)))
    return portfolio_return, portfolio_volatility

# number of portfolios to simulate
num_portfolios = 10000

# arrays to store the results
results = np.zeros((3, num_portfolios))

# annualized covariance matrix
cov_matrix = daily_returns.cov() * 252

np.random.seed(42)

for i in range(num_portfolios):
    weights = np.random.random(len(unique_tickers))
    weights /= np.sum(weights)

    portfolio_return, portfolio_volatility = portfolio_performance(weights, expected_returns, cov_matrix)

    results[0,i] = portfolio_return
    results[1,i] = portfolio_volatility
    results[2,i] = portfolio_return / portfolio_volatility  # Sharpe Ratio

plt.figure(figsize=(10, 7))
plt.scatter(results[1,:], results[0,:], c=results[2,:], cmap='YlGnBu', marker='o')
plt.title('Efficient Frontier')
plt.xlabel('Volatility (Standard Deviation)')
plt.ylabel('Expected Return')
plt.colorbar(label='Sharpe Ratio')
plt.grid(True)
plt.show()
```

```
max_sharpe_idx = np.argmax(results[2])
```