Tech Giants: Performance Analysis of Major Indian Tech Stocks

In [1]:
```
!pip install yfinance
```

```
Requirement already satisfied: yfinance in /opt/anaconda3/lib/python3.11/site-packages (0.2.40)
Requirement already satisfied: pandas>=1.3.0 in /opt/anaconda3/lib/python3.11/site-packages (from
yfinance) (2.1.4)
Requirement already satisfied: numpy>=1.16.5 in /opt/anaconda3/lib/python3.11/site-packages (from
yfinance) (1.26.4)
Requirement already satisfied: requests>=2.31 in /opt/anaconda3/lib/python3.11/site-packages (fro
m yfinance) (2.31.0)
Requirement already satisfied: multitasking>=0.0.7 in /opt/anaconda3/lib/python3.11/site-packages
(from yfinance) (0.0.11)
Requirement already satisfied: lxml>=4.9.1 in /opt/anaconda3/lib/python3.11/site-packages (from y
finance) (4.9.3)
Requirement already satisfied: platformdirs>=2.0.0 in /opt/anaconda3/lib/python3.11/site-packages
(from yfinance) (3.10.0)
Requirement already satisfied: pytz>=2022.5 in /opt/anaconda3/lib/python3.11/site-packages (from
yfinance) (2023.3.post1)
Requirement already satisfied: frozendict>=2.3.4 in /opt/anaconda3/lib/python3.11/site-packages
(from yfinance) (2.4.4)
Requirement already satisfied: peewee>=3.16.2 in /opt/anaconda3/lib/python3.11/site-packages (fro
m yfinance) (3.17.5)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /opt/anaconda3/lib/python3.11/site-packa
ges (from yfinance) (4.12.2)
Requirement already satisfied: html5lib>=1.1 in /opt/anaconda3/lib/python3.11/site-packages (from
yfinance) (1.1)
Requirement already satisfied: soupsieve>1.2 in /opt/anaconda3/lib/python3.11/site-packages (from
beautifulsoup4>=4.11.1->yfinance) (2.5)
Requirement already satisfied: six>=1.9 in /opt/anaconda3/lib/python3.11/site-packages (from html
5lib>=1.1->yfinance) (1.16.0)
Requirement already satisfied: webencodings in /opt/anaconda3/lib/python3.11/site-packages (from
html5lib>=1.1->yfinance) (0.5.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/anaconda3/lib/python3.11/site-packa
ges (from pandas>=1.3.0->yfinance) (2.8.2)
Requirement already satisfied: tzdata>=2022.1 in /opt/anaconda3/lib/python3.11/site-packages (fro
m pandas>=1.3.0->yfinance) (2023.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/anaconda3/lib/python3.11/site-pac
kages (from requests>=2.31->yfinance) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in /opt/anaconda3/lib/python3.11/site-packages (from
requests>=2.31->yfinance) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/anaconda3/lib/python3.11/site-packages
(from requests>=2.31->yfinance) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /opt/anaconda3/lib/python3.11/site-packages
(from requests>=2.31->yfinance) (2024.2.2)
```

In [2]:
```
import pandas as pd
import numpy as np
import yfinance as yf
from datetime import date, timedelta
```

In [3]:
```
end_date = date.today().strftime("%Y-%m-%d")
start_date = (date.today() - timedelta(days=30)).strftime("%Y-%m-%d")
```

In [4]:
```
tickers = ['WIPRO.NS', 'TCS.NS', 'INFY.NS', 'HCLTECH.NS']
data = yf.download(tickers, start = start_date, end = end_date, progress = True)
```

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

In [5]:
```
data = data.reset_index()
data
```

Out[5]:

Price	Date	Adj Close						
Ticker		HCLTECH.NS	INFY.NS	TCS.NS	WIPRO.NS	HCLTECH.NS	INFY.NS	TCS.NS
0	2024-06-05	1343.699951	1430.099976	3746.449951	451.500000	1343.699951	1430.099976	3746.44995
1	2024-06-06	1397.500000	1472.250000	3830.399902	461.000000	1397.500000	1472.250000	3830.39990
2	2024-06-07	1431.500000	1533.599976	3893.949951	484.549988	1431.500000	1533.599976	3893.94995
3	2024-06-10	1418.750000	1499.750000	3858.699951	475.250000	1418.750000	1499.750000	3858.69995
4	2024-06-11	1428.800049	1495.750000	3852.100098	476.049988	1428.800049	1495.750000	3852.10009
5	2024-06-12	1438.750000	1485.199951	3831.649902	476.899994	1438.750000	1485.199951	3831.64990
6	2024-06-13	1444.150024	1493.949951	3878.149902	482.600006	1444.150024	1493.949951	3878.14990
7	2024-06-14	1431.050049	1488.900024	3832.050049	477.500000	1431.050049	1488.900024	3832.05004
8	2024-06-18	1437.199951	1498.199951	3815.100098	491.850006	1437.199951	1498.199951	3815.10009
9	2024-06-19	1445.849976	1511.349976	3801.699951	495.750000	1445.849976	1511.349976	3801.69995
10	2024-06-20	1443.449951	1515.400024	3787.250000	490.399994	1443.449951	1515.400024	3787.25000
11	2024-06-21	1447.849976	1532.699951	3810.750000	490.549988	1447.849976	1532.699951	3810.75000
12	2024-06-24	1440.849976	1527.150024	3816.800049	490.549988	1440.849976	1527.150024	3816.80004
13	2024-06-25	1447.949951	1541.949951	3838.449951	496.850006	1447.949951	1541.949951	3838.44995
14	2024-06-26	1443.699951	1540.699951	3855.850098	495.200012	1443.699951	1540.699951	3855.85009
15	2024-06-27	1454.900024	1573.349976	3934.149902	510.799988	1454.900024	1573.349976	3934.14990
16	2024-06-28	1459.599976	1566.750000	3904.149902	514.849976	1459.599976	1566.750000	3904.14990
17	2024-07-01	1468.849976	1590.800049	3978.199951	527.349976	1468.849976	1590.800049	3978.19995
18	2024-07-02	1480.800049	1621.050049	4017.399902	538.200012	1480.800049	1621.050049	4017.39990
19	2024-07-03	1481.000000	1627.400024	3965.250000	539.000000	1481.000000	1627.400024	3965.25000
20	2024-07-04	1522.349976	1650.650024	4020.949951	530.700012	1522.349976	1650.650024	4020.94995

21 rows × 25 columns

In [6]:
```
data_melt = data.melt(id_vars=['Date'], var_name=['Attribute','Ticker'])
data_melt.tail(50)
```

Out[6]:

	Date	Attribute	Ticker	value
454	2024-06-25	Volume	INFY.NS	4551525.0
455	2024-06-26	Volume	INFY.NS	4851004.0
456	2024-06-27	Volume	INFY.NS	14757754.0
457	2024-06-28	Volume	INFY.NS	8197544.0
458	2024-07-01	Volume	INFY.NS	6801771.0
459	2024-07-02	Volume	INFY.NS	10493173.0
460	2024-07-03	Volume	INFY.NS	7269965.0
461	2024-07-04	Volume	INFY.NS	8008311.0
462	2024-06-05	Volume	TCS.NS	2799670.0
463	2024-06-06	Volume	TCS.NS	4328036.0
464	2024-06-07	Volume	TCS.NS	4552445.0
465	2024-06-10	Volume	TCS.NS	1734661.0
466	2024-06-11	Volume	TCS.NS	1419898.0
467	2024-06-12	Volume	TCS.NS	2177001.0
468	2024-06-13	Volume	TCS.NS	1932323.0
469	2024-06-14	Volume	TCS.NS	1860730.0
470	2024-06-18	Volume	TCS.NS	1774045.0
471	2024-06-19	Volume	TCS.NS	1509050.0
472	2024-06-20	Volume	TCS.NS	2846526.0
473	2024-06-21	Volume	TCS.NS	4642195.0
474	2024-06-24	Volume	TCS.NS	1702154.0
475	2024-06-25	Volume	TCS.NS	1338808.0
476	2024-06-26	Volume	TCS.NS	1639845.0
477	2024-06-27	Volume	TCS.NS	4526556.0
478	2024-06-28	Volume	TCS.NS	2731571.0
479	2024-07-01	Volume	TCS.NS	2658723.0
480	2024-07-02	Volume	TCS.NS	2307449.0
481	2024-07-03	Volume	TCS.NS	1821198.0
482	2024-07-04	Volume	TCS.NS	2518001.0
483	2024-06-05	Volume	WIPRO.NS	6133714.0
484	2024-06-06	Volume	WIPRO.NS	7712784.0
485	2024-06-07	Volume	WIPRO.NS	36336220.0
486	2024-06-10	Volume	WIPRO.NS	9155876.0
487	2024-06-11	Volume	WIPRO.NS	4685857.0
488	2024-06-12	Volume	WIPRO.NS	6937558.0
489	2024-06-13	Volume	WIPRO.NS	6497966.0
490	2024-06-14	Volume	WIPRO.NS	5661298.0
491	2024-06-18	Volume	WIPRO.NS	15935492.0
492	2024-06-19	Volume	WIPRO.NS	10440142.0
493	2024-06-20	Volume	WIPRO.NS	6285563.0
494	2024-06-21	Volume	WIPRO.NS	47585588.0
495	2024-06-24	Volume	WIPRO.NS	4967259.0
496	2024-06-25	Volume	WIPRO.NS	4629105.0

	Date	Attribute	Ticker	value
497	2024-06-26	Volume	WIPRO.NS	6406045.0
498	2024-06-27	Volume	WIPRO.NS	21068569.0
499	2024-06-28	Volume	WIPRO.NS	8202916.0
500	2024-07-01	Volume	WIPRO.NS	15378764.0
501	2024-07-02	Volume	WIPRO.NS	18138484.0
502	2024-07-03	Volume	WIPRO.NS	8024692.0
503	2024-07-04	Volume	WIPRO.NS	9762316.0

In [7]:
```
data_pivot = data_melt.pivot_table(index=['Date', 'Ticker'], columns='Attribute', values='value',
data_pivot
```

Out[7]:

	Attribute	Adj Close	Close	High	Low	Open	Volume
Date	**Ticker**						
2024-06-05	**HCLTECH.NS**	1343.699951	1343.699951	1356.900024	1316.099976	1320.000000	3005139.0
	INFY.NS	1430.099976	1430.099976	1438.000000	1400.150024	1400.150024	9233424.0
	TCS.NS	3746.449951	3746.449951	3783.800049	3700.000000	3716.000000	2799670.0
	WIPRO.NS	451.500000	451.500000	455.000000	439.049988	439.700012	6133714.0
2024-06-06	**HCLTECH.NS**	1397.500000	1397.500000	1399.800049	1350.900024	1354.000000	5936709.0
...	**...**	...	...	...	...	...	...
2024-07-03	**WIPRO.NS**	539.000000	539.000000	545.450012	535.500000	541.900024	8024692.0
2024-07-04	**HCLTECH.NS**	1522.349976	1522.349976	1534.550049	1485.000000	1485.000000	6822557.0
	INFY.NS	1650.650024	1650.650024	1660.000000	1628.000000	1628.199951	8008311.0
	TCS.NS	4020.949951	4020.949951	4047.350098	3982.100098	3999.850098	2518001.0
	WIPRO.NS	530.700012	530.700012	548.799988	529.099976	541.400024	9762316.0

84 rows × 6 columns

In [8]:
```
stock_data = data_pivot.reset_index()
stock_data.head(20)
```

Out[8]:

Attribute	Date	Ticker	Adj Close	Close	High	Low	Open	Volu
0	2024-06-05	HCLTECH.NS	1343.699951	1343.699951	1356.900024	1316.099976	1320.000000	300513
1	2024-06-05	INFY.NS	1430.099976	1430.099976	1438.000000	1400.150024	1400.150024	923342
2	2024-06-05	TCS.NS	3746.449951	3746.449951	3783.800049	3700.000000	3716.000000	279967
3	2024-06-05	WIPRO.NS	451.500000	451.500000	455.000000	439.049988	439.700012	613371
4	2024-06-06	HCLTECH.NS	1397.500000	1397.500000	1399.800049	1350.900024	1354.000000	593670
5	2024-06-06	INFY.NS	1472.250000	1472.250000	1474.300049	1437.500000	1444.949951	1266093
6	2024-06-06	TCS.NS	3830.399902	3830.399902	3839.899902	3741.500000	3781.000000	432803
7	2024-06-06	WIPRO.NS	461.000000	461.000000	462.000000	452.600006	454.950012	771278
8	2024-06-07	HCLTECH.NS	1431.500000	1431.500000	1438.650024	1398.000000	1407.000000	588402
9	2024-06-07	INFY.NS	1533.599976	1533.599976	1539.699951	1477.250000	1481.000000	2407530
10	2024-06-07	TCS.NS	3893.949951	3893.949951	3915.000000	3837.300049	3837.300049	455244
11	2024-06-07	WIPRO.NS	484.549988	484.549988	486.399994	468.299988	470.000000	3633622
12	2024-06-10	HCLTECH.NS	1418.750000	1418.750000	1442.000000	1397.500000	1440.000000	329373
13	2024-06-10	INFY.NS	1499.750000	1499.750000	1529.800049	1497.300049	1525.300049	681060
14	2024-06-10	TCS.NS	3858.699951	3858.699951	3905.899902	3841.899902	3895.000000	173466
15	2024-06-10	WIPRO.NS	475.250000	475.250000	488.399994	473.350006	488.399994	915587
16	2024-06-11	HCLTECH.NS	1428.800049	1428.800049	1437.949951	1415.400024	1425.000000	226908
17	2024-06-11	INFY.NS	1495.750000	1495.750000	1506.449951	1493.949951	1500.050049	485305
18	2024-06-11	TCS.NS	3852.100098	3852.100098	3879.949951	3841.000000	3845.000000	141989
19	2024-06-11	WIPRO.NS	476.049988	476.049988	479.200012	474.350006	479.200012	468585

In [9]:

```
stock_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 84 entries, 0 to 83
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       84 non-null     datetime64[ns]
 1   Ticker     84 non-null     object
 2   Adj Close  84 non-null     float64
 3   Close      84 non-null     float64
 4   High       84 non-null     float64
 5   Low        84 non-null     float64
 6   Open       84 non-null     float64
 7   Volume     84 non-null     float64
dtypes: datetime64[ns](1), float64(6), object(1)
memory usage: 5.4+ KB
```

In [10]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

In [11]:
```
stock_data['Date'] = pd.to_datetime(stock_data['Date'])
stock_data.set_index('Date', inplace=True)
stock_data.reset_index(inplace=True)
```

In [12]:
```
fig = px.line(stock_data, x='Date', y='Adj Close', color='Ticker', markers=True,
              title='Adjusted Close Price Over Time',
              labels={'Date': 'Date', 'Adj Close': 'Adjusted Close Price'})

# Update layout for better readability
fig.update_layout(
    title={'text': 'Adjusted Close Price Over Time', 'x': 0.5, 'xanchor': 'center', 'font': {'siz
    xaxis_title={'text': 'Date', 'font': {'size': 14}},
    yaxis_title={'text': 'Adjusted Close Price', 'font': {'size': 14}},
    legend_title={'text': 'Ticker', 'font': {'size': 13}},
    legend={'font': {'size': 11}},
    xaxis={'tickangle': 45},
    template='plotly_white'
)

fig.show("svg")
```

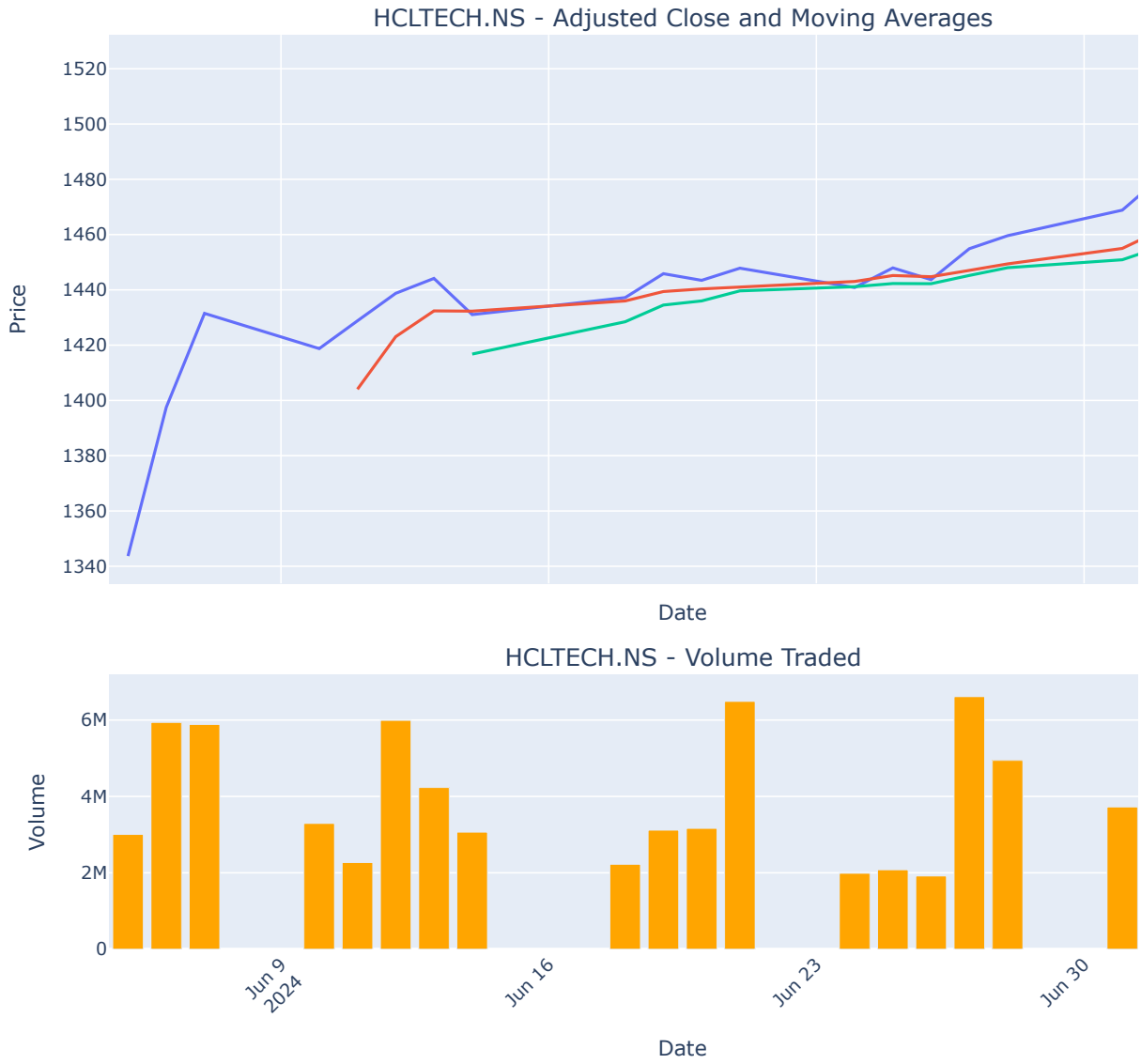
```
/opt/anaconda3/lib/python3.11/site-packages/_plotly_utils/basevalidators.py:106: FutureWarning: T
he behavior of DatetimeProperties.to_pydatetime is deprecated, in a future version this will retu
rn a Series containing python datetime objects instead of an ndarray. To retain the old behavior,
call `np.array` on the result
  v = v.dt.to_pydatetime()
```

In [13]:
```
short_window = 5
long_window = 8

data = stock_data.set_index('Date')
unique_tickers = data['Ticker'].unique()
```

In [14]:
```
ticker_data = data[data['Ticker'] == 'HCLTECH.NS'].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()
```

In [15]:
```
import plotly.graph_objs as go
import plotly.express as px
from plotly.subplots import make_subplots

def advanced_stock_plot(ticker_data, ticker):
    """
    Creates a combined plot of adjusted close, moving averages, and volume for a stock.

    Args:
        ticker_data (pandas.DataFrame): DataFrame containing stock data with columns
            for 'Adj Close', '50_MA', '200_MA', and 'Volume', along with a DatetimeIndex.
        ticker (str): The ticker symbol of the stock.

    Returns:
        None
    """

    # Plot Adjusted Close and Moving Averages
    fig = make_subplots(rows=2, cols=1, shared_xaxes=True, vertical_spacing=0.1,
                        subplot_titles=(f'{ticker} - Adjusted Close and Moving Averages', f'{tick
                        row_heights=[0.6, 0.3])

    fig.add_trace(go.Scatter(x=ticker_data.index, y=ticker_data['Adj Close'], mode='lines', name=
                  row=1, col=1)
    fig.add_trace(go.Scatter(x=ticker_data.index, y=ticker_data['50_MA'], mode='lines', name='50-
                  row=1, col=1)
    fig.add_trace(go.Scatter(x=ticker_data.index, y=ticker_data['200_MA'], mode='lines', name='20
                  row=1, col=1)

    # Plot Volume
    fig.add_trace(go.Bar(x=ticker_data.index, y=ticker_data['Volume'], name='Volume', marker_colo
                  row=2, col=1)

    # Update layout
    fig.update_layout(height=800, width=1000, title_text=f'{ticker} Stock Data',
                      xaxis1=dict(title='Date', tickangle=-45),
                      yaxis1=dict(title='Price'),
                      xaxis2=dict(title='Date', tickangle=-45),
                      yaxis2=dict(title='Volume'))

    fig.show("svg")

# Assuming you have your ticker_data loaded...
advanced_stock_plot(ticker_data, 'HCLTECH.NS')
```

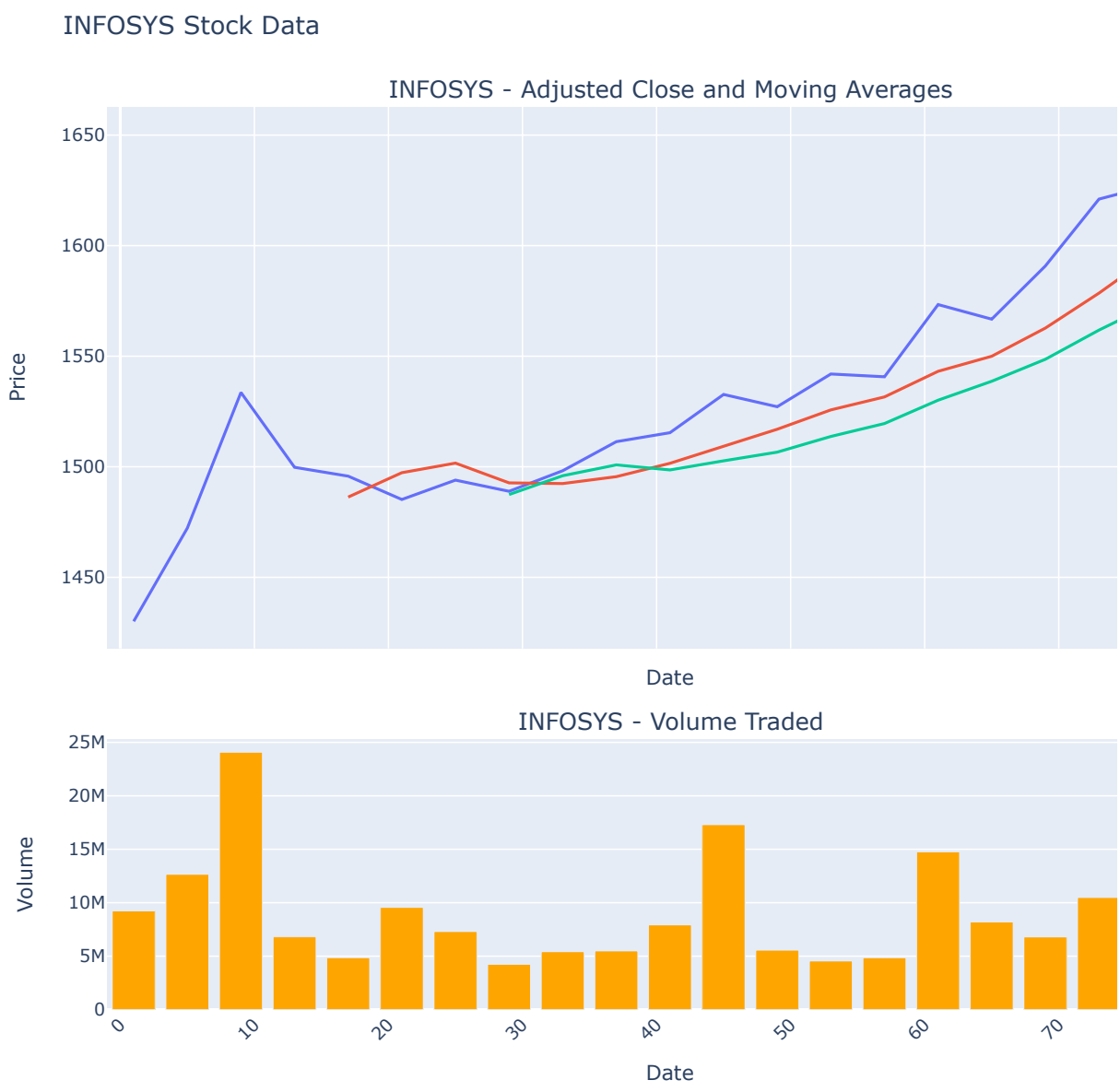
HCLTECH.NS Stock Data



In [16]:
```
ticker_data = stock_data[stock_data['Ticker'] == 'INFY.NS'].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()
```

In [17]:
```
import plotly.graph_objs as go
import plotly.express as px
from plotly.subplots import make_subplots

def advanced_stock_plot(ticker_data, ticker):
    """
    Creates a combined plot of adjusted close, moving averages, and volume for a stock.

    Args:
        ticker_data (pandas.DataFrame): DataFrame containing stock data with columns
            for 'Adj Close', '50_MA', '200_MA', and 'Volume', along with a DatetimeIndex.
        ticker (str): The ticker symbol of the stock.

    Returns:
        None
    """

    # Plot Adjusted Close and Moving Averages
    fig = make_subplots(rows=2, cols=1, shared_xaxes=True, vertical_spacing=0.1,
                        subplot_titles=(f'{ticker} - Adjusted Close and Moving Averages', f'{tick
                        row_heights=[0.6, 0.3])
```

```
    fig.add_trace(go.Scatter(x=ticker_data.index, y=ticker_data['Adj Close'], mode='lines', name=
                  row=1, col=1)
    fig.add_trace(go.Scatter(x=ticker_data.index, y=ticker_data['50_MA'], mode='lines', name='50-
                  row=1, col=1)
    fig.add_trace(go.Scatter(x=ticker_data.index, y=ticker_data['200_MA'], mode='lines', name='20
                  row=1, col=1)

    # Plot Volume
    fig.add_trace(go.Bar(x=ticker_data.index, y=ticker_data['Volume'], name='Volume', marker_colo
                  row=2, col=1)

    # Update layout
    fig.update_layout(height=800, width=1000, title_text=f'{ticker} Stock Data',
                      xaxis1=dict(title='Date', tickangle=-45),
                      yaxis1=dict(title='Price'),
                      xaxis2=dict(title='Date', tickangle=-45),
                      yaxis2=dict(title='Volume'))

    fig.show("svg")

# Assuming you have your ticker_data loaded...
advanced_stock_plot(ticker_data, 'INFOSYS')
```

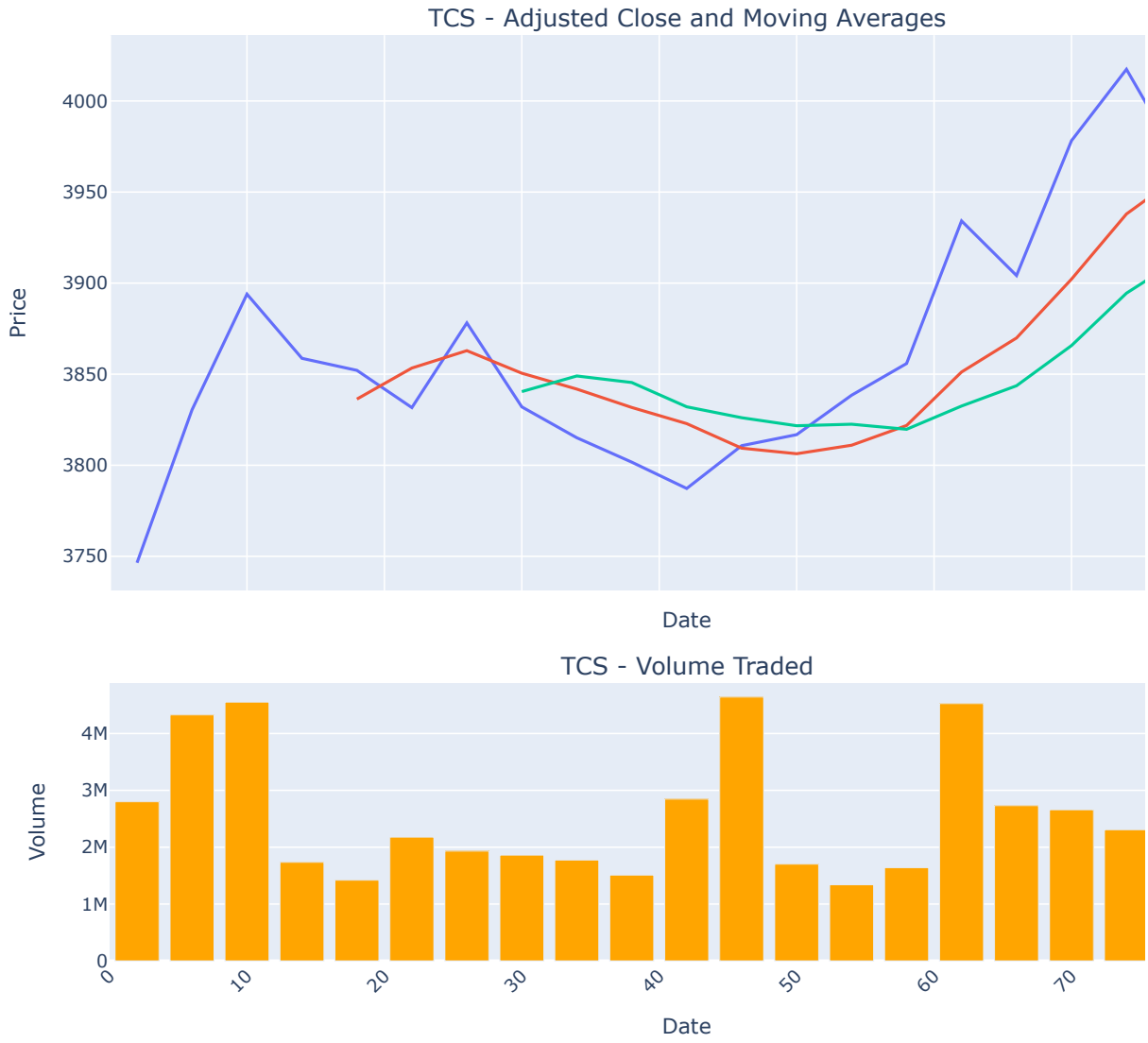


In [18]:
```
ticker_data = stock_data[stock_data['Ticker'] == 'TCS.NS'].copy()
ticker_data['50_MA'] = ticker_data['Adj Close'].rolling(window=short_window).mean()
ticker_data['200_MA'] = ticker_data['Adj Close'].rolling(window=long_window).mean()
```

In [19]:

```
import plotly.graph_objs as go
import plotly.express as px
from plotly.subplots import make_subplots

def advanced_stock_plot(ticker_data, ticker):
    """
    Creates a combined plot of adjusted close, moving averages, and volume for a stock.

    Args:
        ticker_data (pandas.DataFrame): DataFrame containing stock data with columns
            for 'Adj Close', '50_MA', '200_MA', and 'Volume', along with a DatetimeIndex.
        ticker (str): The ticker symbol of the stock.

    Returns:
        None
    """

    # Plot Adjusted Close and Moving Averages
    fig = make_subplots(rows=2, cols=1, shared_xaxes=True, vertical_spacing=0.1,
                        subplot_titles=(f'{ticker} - Adjusted Close and Moving Averages', f'{tick
                        row_heights=[0.6, 0.3])

    fig.add_trace(go.Scatter(x=ticker_data.index, y=ticker_data['Adj Close'], mode='lines', name=
                  row=1, col=1)
    fig.add_trace(go.Scatter(x=ticker_data.index, y=ticker_data['50_MA'], mode='lines', name='50-
                  row=1, col=1)
    fig.add_trace(go.Scatter(x=ticker_data.index, y=ticker_data['200_MA'], mode='lines', name='20
                  row=1, col=1)

    # Plot Volume
    fig.add_trace(go.Bar(x=ticker_data.index, y=ticker_data['Volume'], name='Volume', marker_colo
                  row=2, col=1)

    # Update layout
    fig.update_layout(height=800, width=1000, title_text=f'{ticker} Stock Data',
                      xaxis1=dict(title='Date', tickangle=-45),
                      yaxis1=dict(title='Price'),
                      xaxis2=dict(title='Date', tickangle=-45),
                      yaxis2=dict(title='Volume'))

    fig.show("svg")

# Assuming you have your ticker_data loaded...
advanced_stock_plot(ticker_data, 'TCS')
```

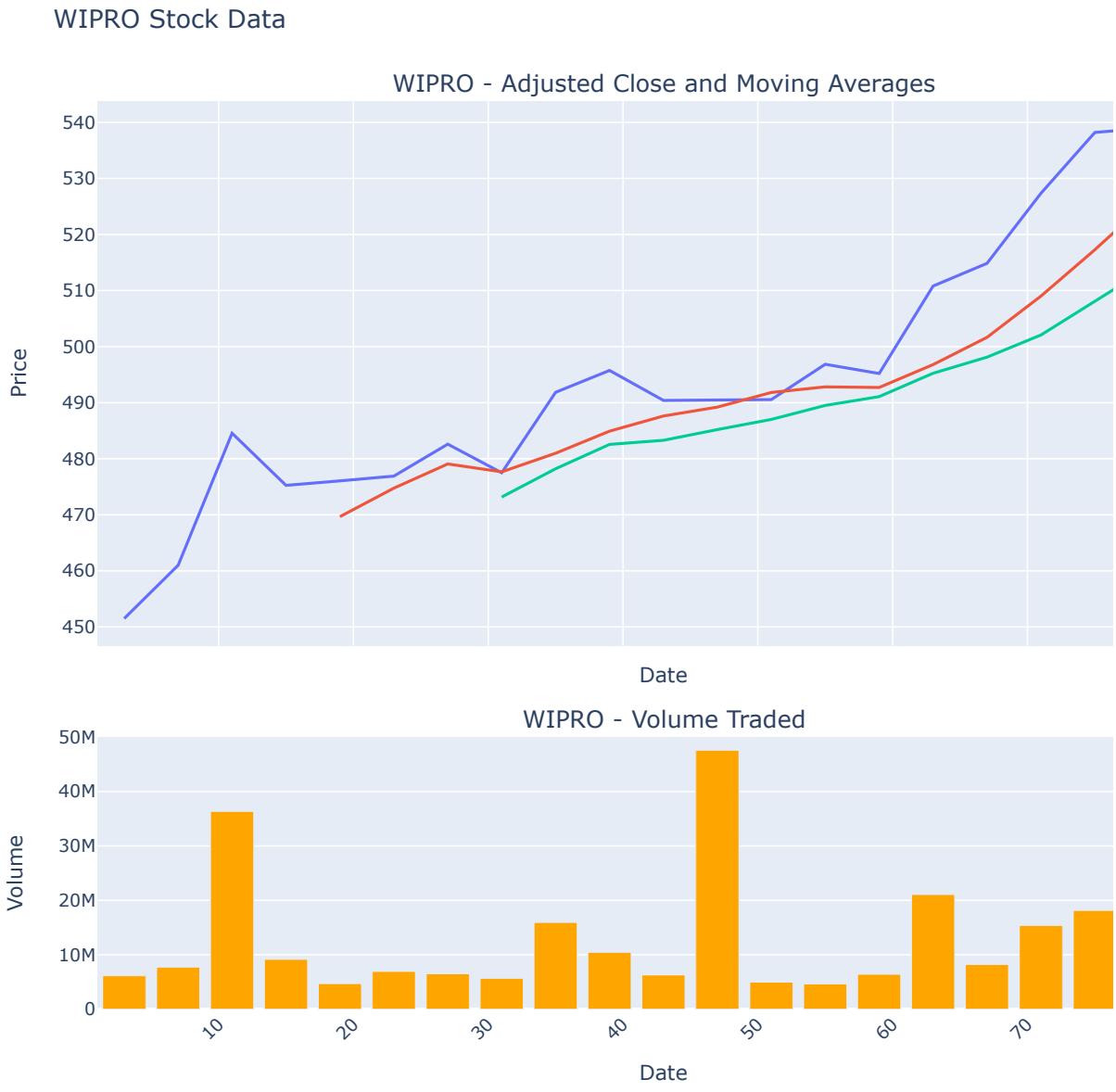
TCS Stock Data



In [20]:
```
ticker_data = stock_data[stock_data['Ticker'] == 'WIPRO.NS'].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()
```

In [21]:
```
import plotly.graph_objs as go
import plotly.express as px
from plotly.subplots import make_subplots

def advanced_stock_plot(ticker_data, ticker):
    """
    Creates a combined plot of adjusted close, moving averages, and volume for a stock.

    Args:
        ticker_data (pandas.DataFrame): DataFrame containing stock data with columns
            for 'Adj Close', '50_MA', '200_MA', and 'Volume', along with a DatetimeIndex.
        ticker (str): The ticker symbol of the stock.

    Returns:
        None
    """

    # Plot Adjusted Close and Moving Averages
    fig = make_subplots(rows=2, cols=1, shared_xaxes=True, vertical_spacing=0.1,
                        subplot_titles=(f'{ticker} - Adjusted Close and Moving Averages', f'{tick
                        row_heights=[0.6, 0.3])
```

```
    fig.add_trace(go.Scatter(x=ticker_data.index, y=ticker_data['Adj Close'], mode='lines', name=
                  row=1, col=1)
    fig.add_trace(go.Scatter(x=ticker_data.index, y=ticker_data['50_MA'], mode='lines', name='50-
                  row=1, col=1)
    fig.add_trace(go.Scatter(x=ticker_data.index, y=ticker_data['200_MA'], mode='lines', name='20
                  row=1, col=1)

    # Plot Volume
    fig.add_trace(go.Bar(x=ticker_data.index, y=ticker_data['Volume'], name='Volume', marker_colo
                  row=2, col=1)

    # Update layout
    fig.update_layout(height=800, width=1000, title_text=f'{ticker} Stock Data',
                      xaxis1=dict(title='Date', tickangle=-45),
                      yaxis1=dict(title='Price'),
                      xaxis2=dict(title='Date', tickangle=-45),
                      yaxis2=dict(title='Volume'))

    fig.show("svg")

# Assuming you have your ticker_data loaded...
advanced_stock_plot(ticker_data, 'WIPRO')
```



In [22]:
```
stock_data['Daily Return'] = stock_data.groupby('Ticker')['Adj Close'].pct_change()
```

In [23]:
```
stock_data
```

Out[23]:

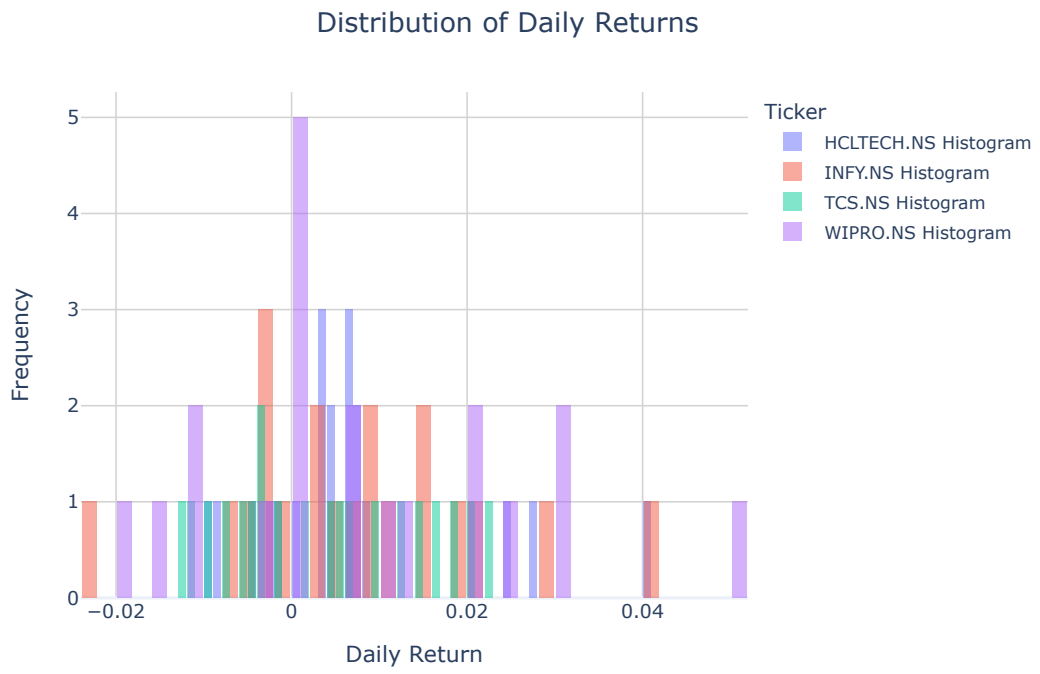
Attribute	Date	Ticker	Adj Close	Close	High	Low	Open	Volum
0	2024-06-05	HCLTECH.NS	1343.699951	1343.699951	1356.900024	1316.099976	1320.000000	3005139.
1	2024-06-05	INFY.NS	1430.099976	1430.099976	1438.000000	1400.150024	1400.150024	9233424.
2	2024-06-05	TCS.NS	3746.449951	3746.449951	3783.800049	3700.000000	3716.000000	2799670.
3	2024-06-05	WIPRO.NS	451.500000	451.500000	455.000000	439.049988	439.700012	6133714.
4	2024-06-06	HCLTECH.NS	1397.500000	1397.500000	1399.800049	1350.900024	1354.000000	5936709.
...	...	...	...	...	...	...	...	...
79	2024-07-03	WIPRO.NS	539.000000	539.000000	545.450012	535.500000	541.900024	8024692.
80	2024-07-04	HCLTECH.NS	1522.349976	1522.349976	1534.550049	1485.000000	1485.000000	6822557.
81	2024-07-04	INFY.NS	1650.650024	1650.650024	1660.000000	1628.000000	1628.199951	8008311.
82	2024-07-04	TCS.NS	4020.949951	4020.949951	4047.350098	3982.100098	3999.850098	2518001.
83	2024-07-04	WIPRO.NS	530.700012	530.700012	548.799988	529.099976	541.400024	9762316.

84 rows × 9 columns

In [24]:

```
import plotly.express as px
import plotly.graph_objects as go
import pandas as pd

# Create a figure
fig = go.Figure()

# Loop through each ticker
for ticker in unique_tickers:
    ticker_data = stock_data[stock_data['Ticker'] == ticker]

    # Add histogram
    fig.add_trace(go.Histogram(
        x=ticker_data['Daily Return'].dropna(),
        nbinsx=50,
        name=f'{ticker} Histogram',
        opacity=0.5
    ))

    # Add KDE (Kernel Density Estimate)


# Update layout
fig.update_layout(
    title={'text': 'Distribution of Daily Returns', 'x': 0.5, 'xanchor': 'center', 'yanchor': 'to
    xaxis_title='Daily Return',
    yaxis_title='Frequency',
    legend_title={'text': 'Ticker', 'font': {'size': 13}},
    legend={'font': {'size': 11}},
    template='plotly_white',
    barmode='overlay',
    bargap=0.2,
    xaxis={'gridcolor': 'lightgray'},
    yaxis={'gridcolor': 'lightgray'}
)
```

```
# Show plot
fig.show("svg")
```



In [25]:
```
daily_returns = stock_data.pivot_table(index='Date', columns='Ticker', values='Daily Return')
correlation_matrix = daily_returns.corr()
```

In [26]:
```
daily_returns
```

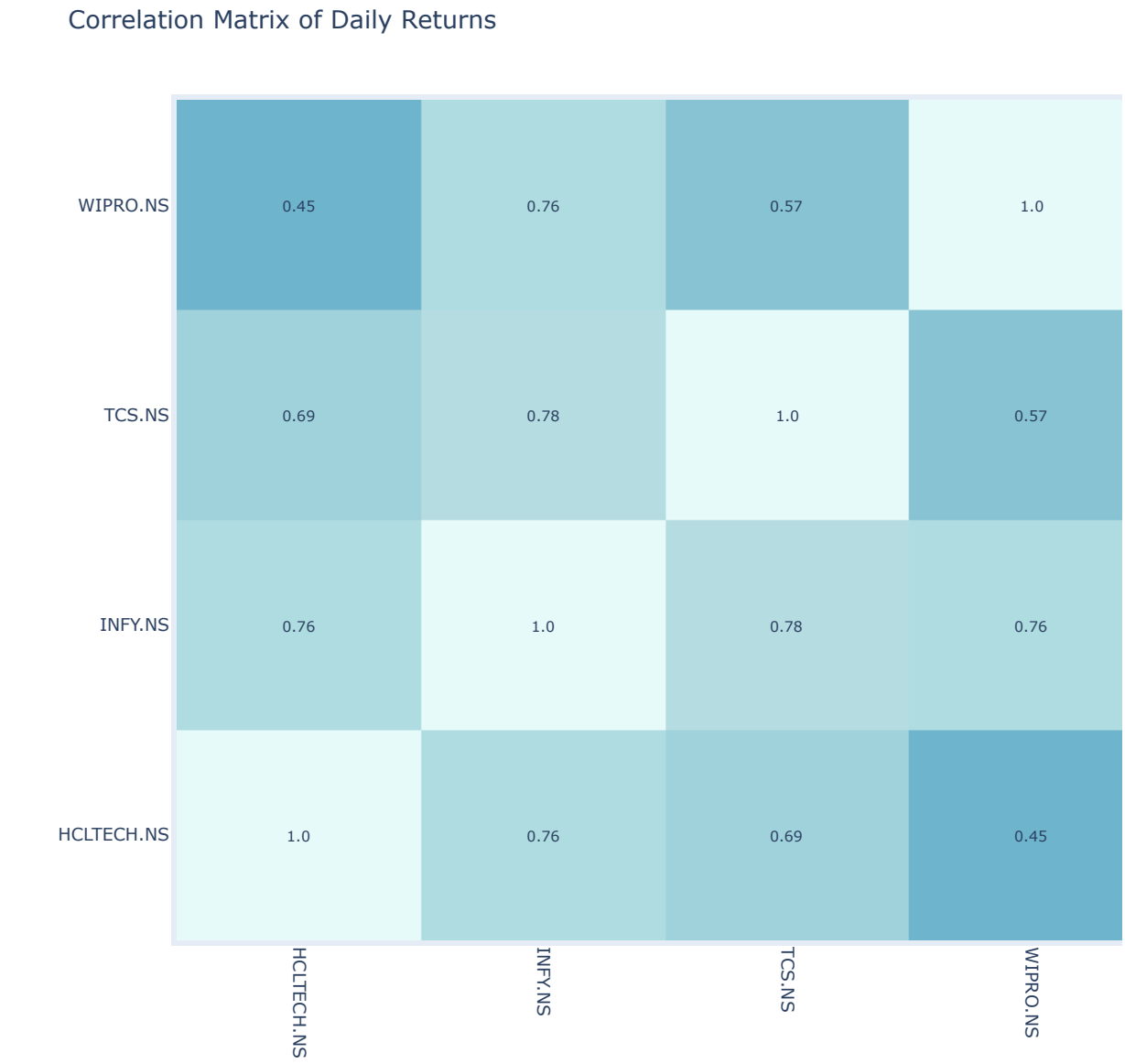
Out[26]:

Ticker	HCLTECH.NS	INFY.NS	TCS.NS	WIPRO.NS
Date				
2024-06-06	0.040039	0.029473	0.022408	0.021041
2024-06-07	0.024329	0.041671	0.016591	0.051085
2024-06-10	-0.008907	-0.022072	-0.009053	-0.019193
2024-06-11	0.007084	-0.002667	-0.001710	0.001683
2024-06-12	0.006964	-0.007053	-0.005309	0.001786
2024-06-13	0.003753	0.005891	0.012136	0.011952
2024-06-14	-0.009071	-0.003380	-0.011887	-0.010568
2024-06-18	0.004297	0.006246	-0.004423	0.030052
2024-06-19	0.006019	0.008777	-0.003512	0.007929
2024-06-20	-0.001660	0.002680	-0.003801	-0.010792
2024-06-21	0.003048	0.011416	0.006205	0.000306
2024-06-24	-0.004835	-0.003621	0.001588	0.000000
2024-06-25	0.004928	0.009691	0.005672	0.012843
2024-06-26	-0.002935	-0.000811	0.004533	-0.003321
2024-06-27	0.007758	0.021192	0.020307	0.031502
2024-06-28	0.003230	-0.004195	-0.007626	0.007929
2024-07-01	0.006337	0.015350	0.018967	0.024279
2024-07-02	0.008136	0.019016	0.009854	0.020575
2024-07-03	0.000135	0.003917	-0.012981	0.001486
2024-07-04	0.027920	0.014287	0.014047	-0.015399

In [27]:

```
# Create heatmap
fig = go.Figure(data=go.Heatmap(
    z=correlation_matrix.values,
    x=correlation_matrix.columns,
    y=correlation_matrix.index,
    colorscale='ice',
    zmin=-1, zmax=1,
    text=np.round(correlation_matrix.values, 2),  # Add annotations
    hoverinfo='text',
    showscale=True
))

# Add annotations
annotations = []
for i in range(correlation_matrix.shape[0]):
    for j in range(correlation_matrix.shape[1]):
        annotations.append(
            go.layout.Annotation(
                text=str(np.round(correlation_matrix.values[i, j], 2)),
                x=correlation_matrix.columns[j],
                y=correlation_matrix.index[i],
                xref='x1', yref='y1',
                showarrow=False,
                font=dict(size=10)
            )
        )

# Update layout
fig.update_layout(
    title='Correlation Matrix of Daily Returns',
    xaxis_nticks=len(correlation_matrix.columns),
    yaxis_nticks=len(correlation_matrix.index),
    xaxis_title='',
    yaxis_title='',
    annotations=annotations,
    autosize=False,
    width=900,
```

```
    height=800,
    margin=dict(l=40, r=40, b=85, t=100, pad=4),
    xaxis=dict(tickangle=90)
)

# Show plot
fig.show("svg")
```



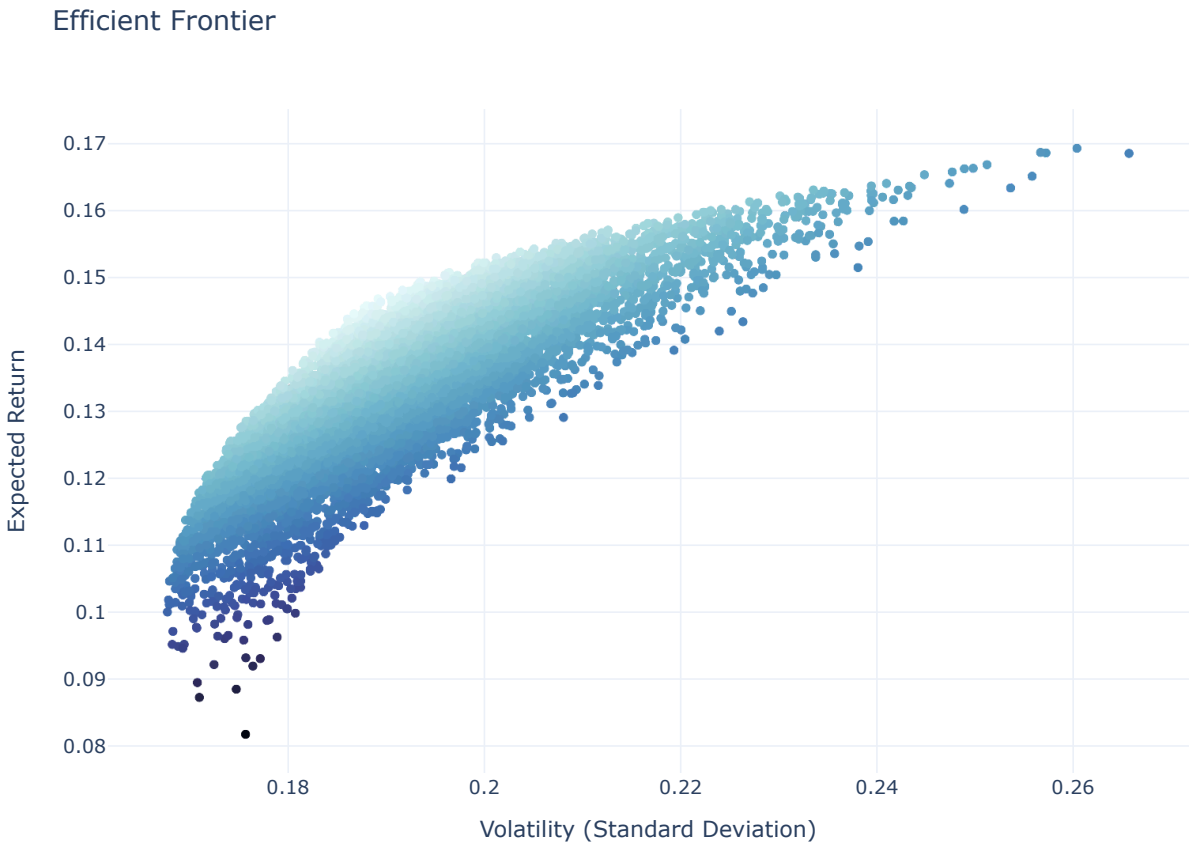
In [28]:
```
import numpy as np

expected_returns = daily_returns.mean() * 21  # annualize the returns
volatility = daily_returns.std() * np.sqrt(21)  # annualize the volatility

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

stock_stats
```

Out[28]:

Ticker	Expected Return	Volatility
HCLTECH.NS	0.132898	0.055111
INFY.NS	0.153099	0.064610
TCS.NS	0.075606	0.050997
WIPRO.NS	0.173434	0.080637

In [29]:
```python
# 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

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

In [30]:
```python
import plotly.express as px
import plotly.graph_objects as go
import numpy as np

# Create the scatter plot
fig = px.scatter(
    x=results[1, :],  # Volatility (Standard Deviation)
    y=results[0, :],  # Expected Return
    color=results[2, :],  # Sharpe Ratio
    labels={
        'x': 'Volatility (Standard Deviation)',
        'y': 'Expected Return',
        'color': 'Sharpe Ratio'
    },
    title='Efficient Frontier',
    color_continuous_scale='ice'
)

# Customize layout
fig.update_layout(
    width=900,
    height=600,
    xaxis=dict(title='Volatility (Standard Deviation)'),
    yaxis=dict(title='Expected Return'),
    coloraxis_colorbar=dict(title='Sharpe Ratio'),
    template='plotly_white',
    showlegend=False
)

# Show plot
fig.show("svg")
```



In [31]:
```
max_sharpe_idx = np.argmax(results[2])
max_sharpe_return = results[0, max_sharpe_idx]
max_sharpe_volatility = results[1, max_sharpe_idx]
max_sharpe_ratio = results[2, max_sharpe_idx]

max_sharpe_return, max_sharpe_volatility, max_sharpe_ratio
```

Out[31]: (0.14683808937319137, 0.1891899833168986, 0.7761409288103451)

In [32]:
```
max_sharpe_weights = np.zeros(len(unique_tickers))

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

    if results[2, i] == max_sharpe_ratio:
        max_sharpe_weights = weights
        break

portfolio_weights_df = pd.DataFrame({
    'Ticker': unique_tickers,
    'Weight': max_sharpe_weights
})

portfolio_weights_df
```

Out[32]:

	Ticker	Weight
0	HCLTECH.NS	0.386736
1	INFY.NS	0.340602
2	TCS.NS	0.209266
3	WIPRO.NS	0.063395