Here in this we would be selecting the best combination of stocks to maximize returns and minimizing the risk.

It would be based on historical data and financial metrics

This optimization involves: analyze price trends, calculate expected returns, calculate price volatility, determine correlation between different stocks to see diversification.

we use Modern Portfolio Theory (MPT) for making investment portfolios to maximize expected return based on given level of market risk

MPT -> pratical method for selecting investments in order to maximise their overall results within acceptable level of risks

Result given by stock market portfolio optimization identifies the portfolio with highest Sharpe Ratio which provide a clear allocation strategy for the selected stocks to achieve long term investment goals

Data Collection

for stock market optimization we need data about stock market performance per time. So we will be using real time stock market data using 'yfinance' API

yfinance is used for performing financial analysis, backtesting trading strategies and to develop financial application

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

```
Requirement already satisfied: yfinance in d:\anaconda3\lib\site-packages (0.2.40)
Requirement already satisfied: pandas>=1.3.0 in d:\anaconda3\lib\site-packages (from yfinance) (2.1.4)
Requirement already satisfied: numpy>=1.16.5 in d:\anaconda3\lib\site-packages (from yfinance) (1.26.4)
Requirement already satisfied: requests>=2.31 in d:\anaconda3\lib\site-packages (from yfinance) (2.31.0)
Requirement already satisfied: multitasking>=0.0.7 in d:\anaconda3\lib\site-packages (from yfinance) (0.0.11)
Requirement already satisfied: lxml>=4.9.1 in d:\anaconda3\lib\site-packages (from yfinance) (4.9.3)
Requirement already satisfied: platformdirs>=2.0.0 in d:\anaconda3\lib\site-packages (from yfinance) (3.10.0)
Requirement already satisfied: pytz>=2022.5 in d:\anaconda3\lib\site-packages (from yfinance) (2023.3.post1)
Requirement already satisfied: frozendict>=2.3.4 in d:\anaconda3\lib\site-packages (from yfinance) (2.4.4)
Requirement already satisfied: peewee>=3.16.2 in d:\anaconda3\lib\site-packages (from yfinance) (3.17.5)
Requirement already satisfied: beautifulsoup4>=4.11.1 in d:\anaconda3\lib\site-packages (from yfinance) (4.12.2)
Requirement already satisfied: html5lib>=1.1 in d:\anaconda3\lib\site-packages (from yfinance) (1.1)
Requirement already satisfied: soupsieve>1.2 in d:\anaconda3\lib\site-packages (from beautifulsoup4>=4.11.1->yfi
nance) (2.5)
Requirement already satisfied: six>=1.9 in d:\anaconda3\lib\site-packages (from html5lib>=1.1->yfinance) (1.16.0
)
Requirement already satisfied: webencodings in d:\anaconda3\lib\site-packages (from html5lib>=1.1->yfinance) (0.
5.1)
Requirement already satisfied: python-dateutil>=2.8.2 in d:\anaconda3\lib\site-packages (from pandas>=1.3.0->yfi
nance) (2.8.2)
Requirement already satisfied: tzdata>=2022.1 in d:\anaconda3\lib\site-packages (from pandas>=1.3.0->yfinance) (
2023.3)
Requirement already satisfied: charset-normalizer<4,>=2 in d:\anaconda3\lib\site-packages (from requests>=2.31->
yfinance) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in d:\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (3
.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in d:\anaconda3\lib\site-packages (from requests>=2.31->yfinan
ce) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in d:\anaconda3\lib\site-packages (from requests>=2.31->yfinan
ce) (2024.2.2)
Note: you may need to restart the kernel to use updated packages.
```

Collect stock market data of popular indian companies

In [2]:
```
import pandas as pd
import yfinance as yf
from datetime import date, timedelta
#date used for manipulating and working with specific dates
#timedelta used for difference b/w 2 dates or time,we can (+,-) a time
```

In [3]:
```
#time period for date
end_date=date.today()
```

In [4]:
```
print(end_date)
```

```
2024-07-14
```

In [5]:
```
type(end_date)
```

Out[5]:
```
datetime.date
```

In [6]:
```
start_date=end_date - timedelta(days=365)
```

In [7]:

```
type(start_date)
```

Out[7]:

```
datetime.date
```

In [8]:

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

In [9]:

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

In [10]:

```
data
```

Out[10]:

Price	Adj Close				Close				
Ticker	HDFCBANK.NS	INFY.NS	RELIANCE.NS	TCS.NS	HDFCBANK.NS	INFY.NS	RELIANCE.NS	TCS.NS	HDFCB
Date									
2023-07-17	1656.282715	1396.894409	2572.266846	3432.684814	1678.900024	1422.949951	2581.353271	3491.699951	1682
2023-07-18	1654.901611	1448.187622	2594.110840	3437.747803	1677.500000	1475.199951	2603.274414	3496.850098	1704
2023-07-19	1662.399170	1447.942261	2613.793457	3411.400635	1685.099976	1474.949951	2623.026611	3470.050049	1688
2023-07-20	1666.000000	1422.958252	2610.628174	3413.618408	1688.750000	1449.500000	2619.850098	3463.300049	1692
2023-07-21	1653.175171	1307.217163	2529.813477	3319.981445	1675.750000	1331.599976	2538.750000	3368.300049	1689
...	...	...	...	...	...	...	...	...	
2024-07-08	1635.349976	1661.650024	3201.800049	3993.199951	1635.349976	1661.650024	3201.800049	3993.199951	1654
2024-07-09	1636.500000	1657.150024	3180.550049	3985.500000	1636.500000	1657.150024	3180.550049	3985.500000	1646
2024-07-10	1626.099976	1648.250000	3168.449951	3909.149902	1626.099976	1648.250000	3168.449951	3909.149902	1640
2024-07-11	1621.900024	1652.699951	3161.300049	3923.699951	1621.900024	1652.699951	3161.300049	3923.699951	1625
2024-07-12	1622.699951	1711.750000	3193.449951	4183.950195	1622.699951	1711.750000	3193.449951	4183.950195	1638

243 rows × 24 columns

Data Preparation

In [11]:

```
data=data.reset_index() #added a index col(easy to say in this way)
```

In [12]:

```
data
```

Out[12]:

Price	Date	Adj Close							Close
Ticker		HDFCBANK.NS	INFY.NS	RELIANCE.NS	TCS.NS	HDFCBANK.NS	INFY.NS	RELIANCE.NS	TCS.NS
0	2023-07-17	1656.282715	1396.894409	2572.266846	3432.684814	1678.900024	1422.949951	2581.353271	3491.699951
1	2023-07-18	1654.901611	1448.187622	2594.110840	3437.747803	1677.500000	1475.199951	2603.274414	3496.850098
2	2023-07-19	1662.399170	1447.942261	2613.793457	3411.400635	1685.099976	1474.949951	2623.026611	3470.050049
3	2023-07-20	1666.000000	1422.958252	2610.628174	3413.618408	1688.750000	1449.500000	2619.850098	3463.300049
4	2023-07-21	1653.175171	1307.217163	2529.813477	3319.981445	1675.750000	1331.599976	2538.750000	3368.300049
...	...	...	...	...	...	...	...	...	...
238	2024-07-08	1635.349976	1661.650024	3201.800049	3993.199951	1635.349976	1661.650024	3201.800049	3993.199951
239	2024-07-09	1636.500000	1657.150024	3180.550049	3985.500000	1636.500000	1657.150024	3180.550049	3985.500000
240	2024-07-10	1626.099976	1648.250000	3168.449951	3909.149902	1626.099976	1648.250000	3168.449951	3909.149902
241	2024-07-11	1621.900024	1652.699951	3161.300049	3923.699951	1621.900024	1652.699951	3161.300049	3923.699951
242	2024-07-12	1622.699951	1711.750000	3193.449951	4183.950195	1622.699951	1711.750000	3193.449951	4183.950195

243 rows × 25 columns

In [13]:
```
#melt used to convert wide format into long format data(?)
data_melted=data.melt(id_vars=['Date'], var_name=['Attribute','Ticker'])
#id_vars -> mention those col's which remains unchanged/unmelted
#var_name -> defines the names for the new col's that will store the melted var names
```

In [14]:
```
data_melted
```

Out[14]:

	Date	Attribute	Ticker	value
0	2023-07-17	Adj Close	HDFCBANK.NS	1.656283e+03
1	2023-07-18	Adj Close	HDFCBANK.NS	1.654902e+03
2	2023-07-19	Adj Close	HDFCBANK.NS	1.662399e+03
3	2023-07-20	Adj Close	HDFCBANK.NS	1.666000e+03
4	2023-07-21	Adj Close	HDFCBANK.NS	1.653175e+03
...	...	...	...	...
5827	2024-07-08	Volume	TCS.NS	1.758882e+06
5828	2024-07-09	Volume	TCS.NS	1.305801e+06
5829	2024-07-10	Volume	TCS.NS	2.669716e+06
5830	2024-07-11	Volume	TCS.NS	4.872189e+06
5831	2024-07-12	Volume	TCS.NS	1.350916e+07

5832 rows × 4 columns

In [15]:
```
#pivot the dataframe to have attributes(open,high,low,etc.) as col(?)
data_pivoted=data_melted.pivot_table(index=['Date','Ticker'],columns='Attribute',values='value',aggfunc='first'
```

In [17]:
```
data_pivoted
```

Out[17]:

Attribute		Adj Close	Close	High	Low	Open	Volume
Date	**Ticker**						
2023-07-17	**HDFCBANK.NS**	1656.282715	1678.900024	1682.000000	1633.000000	1650.000000	24626464.0
	INFY.NS	1396.894409	1422.949951	1458.949951	1414.300049	1425.949951	11569884.0
	RELIANCE.NS	2572.266846	2581.353271	2598.290283	2517.943115	2535.480225	11110020.0
	TCS.NS	3432.684814	3491.699951	3549.899902	3477.050049	3510.000000	2743228.0
2023-07-18	**HDFCBANK.NS**	1654.901611	1677.500000	1704.000000	1670.000000	1698.000000	40538409.0
...	**...**	...	...	...	...	...	...
2024-07-11	**TCS.NS**	3923.699951	3923.699951	3980.000000	3895.600098	3931.000000	4872189.0
2024-07-12	**HDFCBANK.NS**	1622.699951	1622.699951	1638.400024	1611.150024	1622.000000	28024980.0
	INFY.NS	1711.750000	1711.750000	1719.750000	1666.650024	1680.000000	17078316.0
	RELIANCE.NS	3193.449951	3193.449951	3210.300049	3149.000000	3169.000000	6462392.0
	TCS.NS	4183.950195	4183.950195	4199.950195	3971.300049	3980.000000	13509164.0

972 rows × 6 columns

In [18]:

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

In [18]:

```
stock_data
```

Out[18]:

Attribute	Date	Ticker	Adj Close	Close	High	Low	Open	Volume
0	2023-07-10	HDFCBANK.NS	1634.135132	1656.449951	1676.750000	1649.699951	1661.000000	19199221.0
1	2023-07-10	INFY.NS	1304.812012	1329.150024	1341.900024	1319.300049	1336.550049	3940315.0
2	2023-07-10	RELIANCE.NS	2515.564209	2524.450195	2543.787109	2469.024170	2481.853760	16620008.0
3	2023-07-10	TCS.NS	3216.648926	3271.949951	3324.750000	3265.199951	3324.750000	1407431.0
4	2023-07-11	HDFCBANK.NS	1626.193604	1648.400024	1676.000000	1645.500000	1663.000000	25335213.0
...	...	...	...	...	...	...	...	...
967	2024-07-04	TCS.NS	4020.949951	4020.949951	4047.350098	3982.100098	3999.850098	2518001.0
968	2024-07-05	HDFCBANK.NS	1648.099976	1648.099976	1685.000000	1642.199951	1685.000000	41121274.0
969	2024-07-05	INFY.NS	1647.449951	1647.449951	1665.849976	1633.349976	1651.449951	7065022.0
970	2024-07-05	RELIANCE.NS	3177.250000	3177.250000	3197.000000	3096.000000	3107.649902	6134855.0
971	2024-07-05	TCS.NS	4011.800049	4011.800049	4026.750000	3988.000000	4010.000000	1668616.0

972 rows × 8 columns

So in data collection and preparation following steps are involved:

downloading the data of list of tickers, reset the index, melted the data(?), pivot the data(?), again reset the data,

Visualize the data

look for performance of these companies in the sock market over time

In [19]:

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

In [20]:

```
stock_data['Date']
```

Out[20]:

```
0      2023-07-17
1      2023-07-17
2      2023-07-17
3      2023-07-17
4      2023-07-18
          ...
967    2024-07-11
968    2024-07-12
969    2024-07-12
970    2024-07-12
971    2024-07-12
Name: Date, Length: 972, dtype: datetime64[ns]
```

In [21]:

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

```
# converting various data types (such as str or array) into pandas specialized datetime objects (Timestamp)
```

this above step is necessary while working with time series data (e.g., stock prices, sensor readings, or event timestamps)and Pandas integrates well with lib like Matplotlib. When your data is in datetime format, you can create informative time-based plots and explore trends visually, plus for indexing also when use choose datetime col as index of your dataframe. cases where you might not need specialized datetime objects: 1. you're only interested in extracting specific components (like year, month, day) you can keep the data as strings or integers. 2.your analysis doesn't involve time-based operations, can use original form

In [22]:
```
stock_data.set_index('Date',inplace=True)
```

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

Out[23]:

Attribute	Ticker	Adj Close	Close	High	Low	Open	Volume
Date							
2023-07-17	HDFCBANK.NS	1656.282715	1678.900024	1682.000000	1633.000000	1650.000000	24626464.0
2023-07-17	INFY.NS	1396.894409	1422.949951	1458.949951	1414.300049	1425.949951	11569884.0
2023-07-17	RELIANCE.NS	2572.266846	2581.353271	2598.290283	2517.943115	2535.480225	11110020.0
2023-07-17	TCS.NS	3432.684814	3491.699951	3549.899902	3477.050049	3510.000000	2743228.0
2023-07-18	HDFCBANK.NS	1654.901611	1677.500000	1704.000000	1670.000000	1698.000000	40538409.0
...	...	...	...	...	...	...	...
2024-07-11	TCS.NS	3923.699951	3923.699951	3980.000000	3895.600098	3931.000000	4872189.0
2024-07-12	HDFCBANK.NS	1622.699951	1622.699951	1638.400024	1611.150024	1622.000000	28024980.0
2024-07-12	INFY.NS	1711.750000	1711.750000	1719.750000	1666.650024	1680.000000	17078316.0
2024-07-12	RELIANCE.NS	3193.449951	3193.449951	3210.300049	3149.000000	3169.000000	6462392.0
2024-07-12	TCS.NS	4183.950195	4183.950195	4199.950195	3971.300049	3980.000000	13509164.0

972 rows × 7 columns

In [24]:
```
stock_data.reset_index(inplace=True)
```

In [25]:
```
stock_data
```

Out[25]:

Attribute	Date	Ticker	Adj Close	Close	High	Low	Open	Volume
0	2023-07-17	HDFCBANK.NS	1656.282715	1678.900024	1682.000000	1633.000000	1650.000000	24626464.0
1	2023-07-17	INFY.NS	1396.894409	1422.949951	1458.949951	1414.300049	1425.949951	11569884.0
2	2023-07-17	RELIANCE.NS	2572.266846	2581.353271	2598.290283	2517.943115	2535.480225	11110020.0
3	2023-07-17	TCS.NS	3432.684814	3491.699951	3549.899902	3477.050049	3510.000000	2743228.0
4	2023-07-18	HDFCBANK.NS	1654.901611	1677.500000	1704.000000	1670.000000	1698.000000	40538409.0
...	...	...	...	...	...	...	...	...
967	2024-07-11	TCS.NS	3923.699951	3923.699951	3980.000000	3895.600098	3931.000000	4872189.0
968	2024-07-12	HDFCBANK.NS	1622.699951	1622.699951	1638.400024	1611.150024	1622.000000	28024980.0
969	2024-07-12	INFY.NS	1711.750000	1711.750000	1719.750000	1666.650024	1680.000000	17078316.0
970	2024-07-12	RELIANCE.NS	3193.449951	3193.449951	3210.300049	3149.000000	3169.000000	6462392.0
971	2024-07-12	TCS.NS	4183.950195	4183.950195	4199.950195	3971.300049	3980.000000	13509164.0

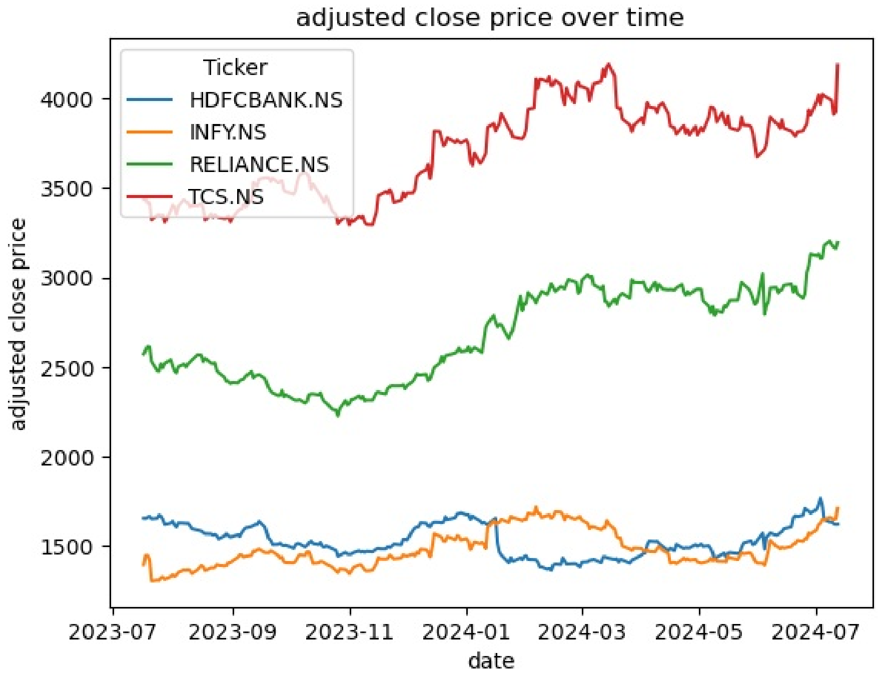
972 rows × 8 columns

In [26]:
```
plt.figure(figsize=(14,7))
```

Out[26]:
```
<Figure size 1400x700 with 0 Axes>
<Figure size 1400x700 with 0 Axes>
```

In [27]:
```
#sns.set(style='whitegrid')
```

In [28]:
```
sns.lineplot(data=stock_data,x='Date',y='Adj Close',hue='Ticker')
plt.title('adjusted close price over time')
plt.xlabel('date')
plt.ylabel('adjusted close price')
#plt.legend(title='Ticker')
plt.show()
```



get 50-day and 200-day moving averages(statistic that captures the average change in a data series over time) and plot them along with the Adjusted Close price for each stock(?)

In [29]:
```
short_window= 50
```

In [30]:
```
long_window=200
```

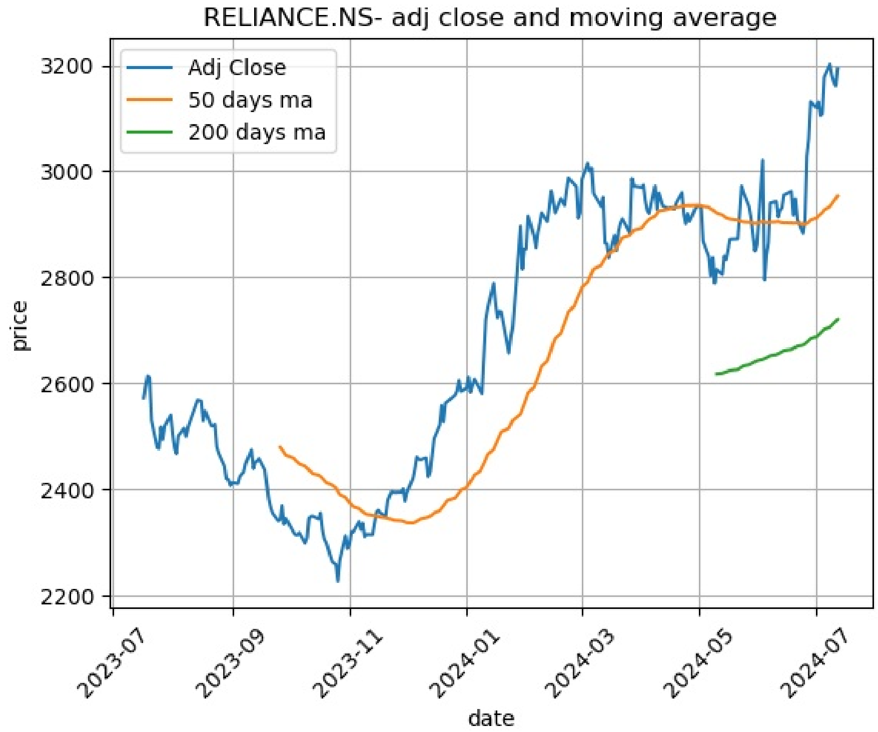
In [31]:
```
stock_data.set_index('Date',inplace=True)
```

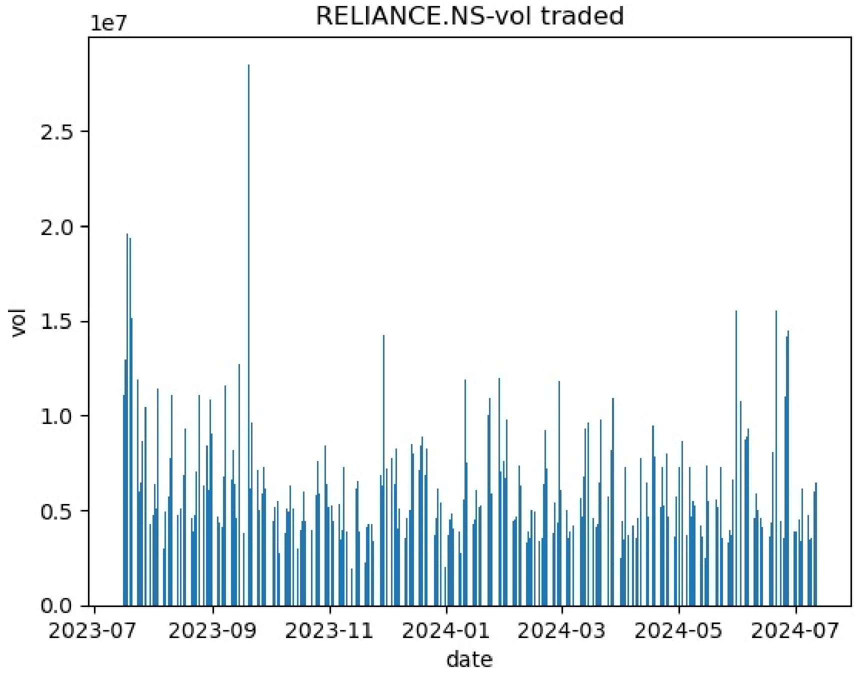
In [33]:
```
unique_tickers=stock_data['Ticker'].unique()
```

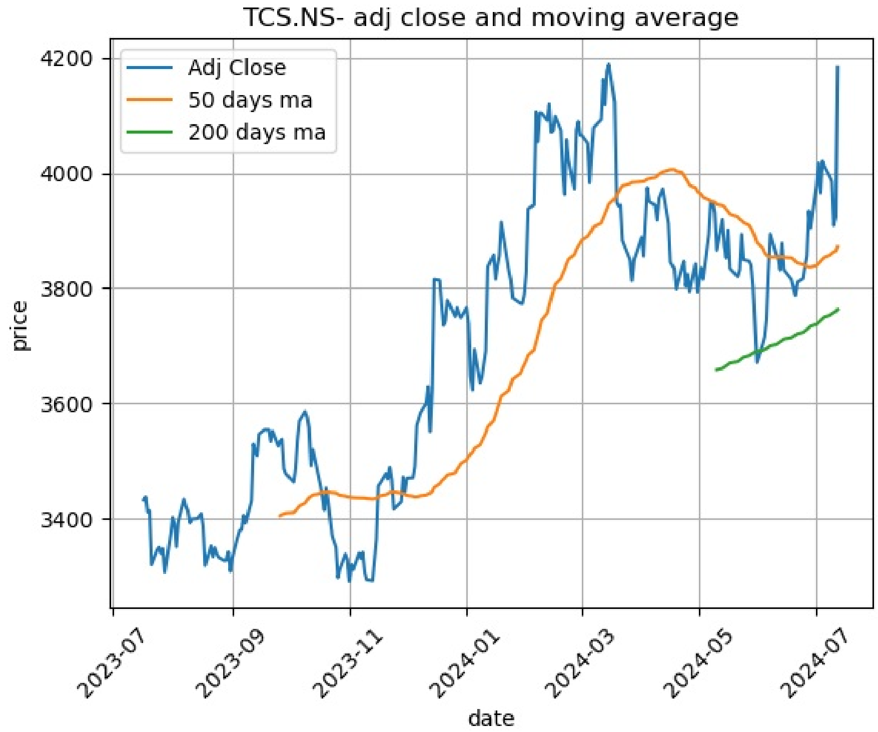
In [35]:
```
tickers
```

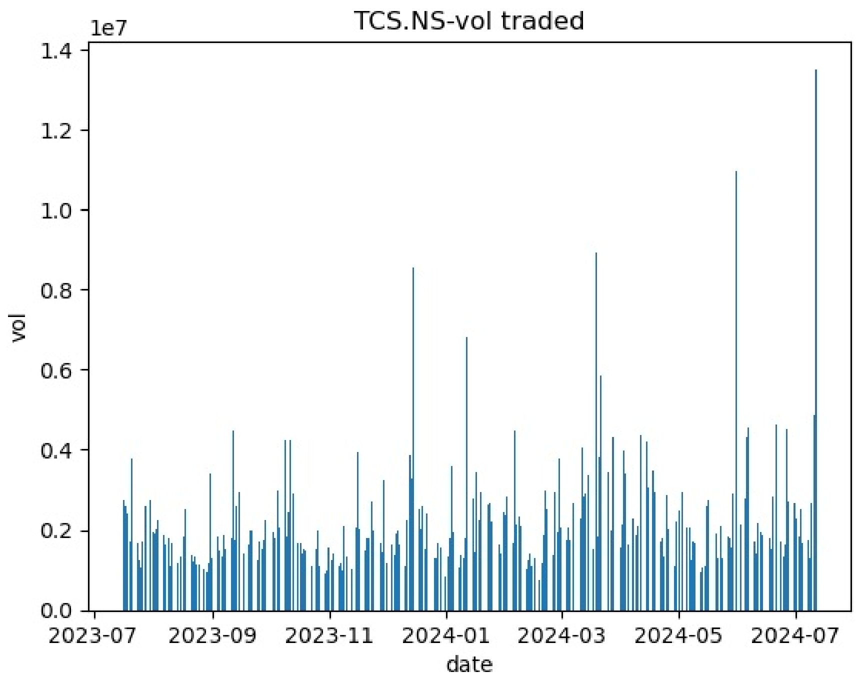
Out[35]:
```
['RELIANCE.NS', 'TCS.NS', 'INFY.NS', 'HDFCBANK.NS']
```

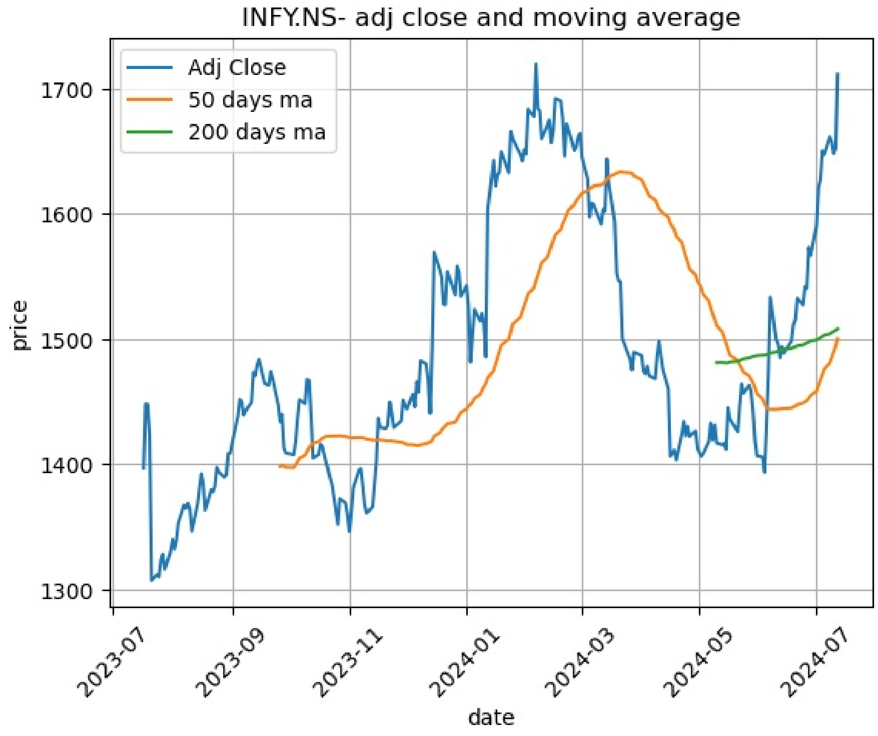
In [36]:
```
for i in tickers:
    ticker_data=stock_data[stock_data['Ticker']==i].copy()
    ticker_data['50ma']= ticker_data['Adj Close'].rolling(window=short_window).mean()
    ticker_data['200ma']= ticker_data['Adj Close'].rolling(window=long_window).mean()

    plt.plot(ticker_data.index,ticker_data['Adj Close'],label='Adj Close')
    plt.plot(ticker_data.index,ticker_data['50ma'],label='50 days ma')
    plt.plot(ticker_data.index,ticker_data['200ma'],label='200 days ma')
    plt.title(f'{i}- adj close and moving average')
    plt.xlabel('date')
    plt.ylabel('price')
    plt.legend()
    plt.grid(True)
    plt.xticks(rotation=45)
    #plt.tight_layout()
    plt.show()

    plt.bar(ticker_data.index,ticker_data['Volume'],label='volume')
    plt.title(f'{i}-vol traded')
    plt.xlabel('date')
    plt.ylabel('vol')
    plt.show()
```

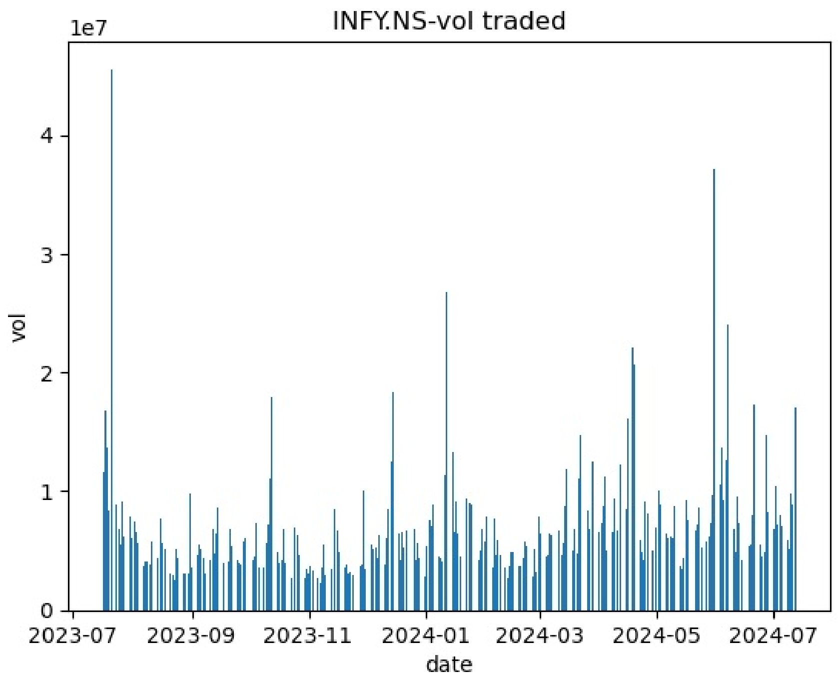

RELIANCE.NS- adj close and moving average
Adj Close
50 days ma
200 days ma
price
3200
3000
2800
2600
2400
2200
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
date

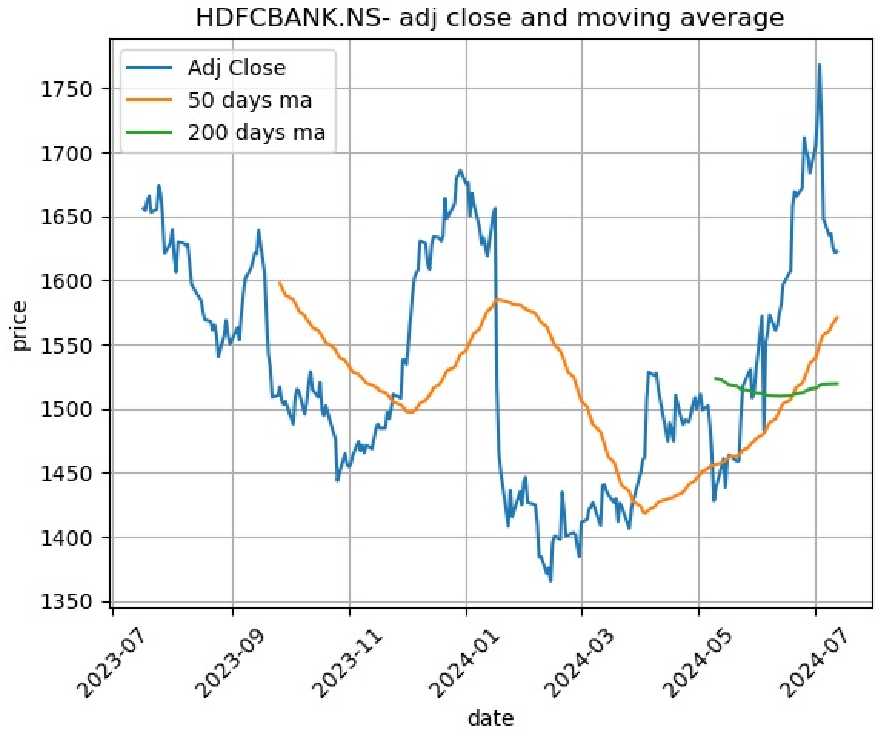

RELIANCE.NS-vol traded
1e7
vol
2.5
2.0
1.5
1.0
0.5
0.0
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07
date

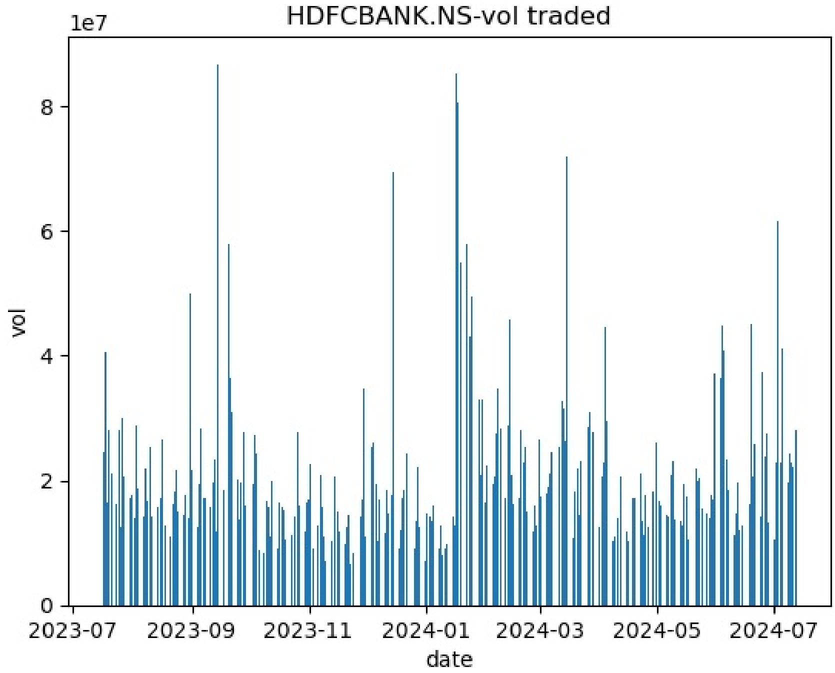

TCS.NS- adj close and moving average
Adj Close
50 days ma
200 days ma
price
date
4200
4000
3800
3600
3400
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07


TCS.NS-vol traded
1e7
vol
date
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07


INFY.NS- adj close and moving average
Adj Close
50 days ma
200 days ma
price
1300
1400
1500
1600
1700
date
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07


INFY.NS-vol traded
1e7
vol
0
1
2
3
4
date
2023-07
2023-09
2023-11
2024-01
2024-03
2024-05
2024-07





see the distribution of daily returns of these stocks(?)

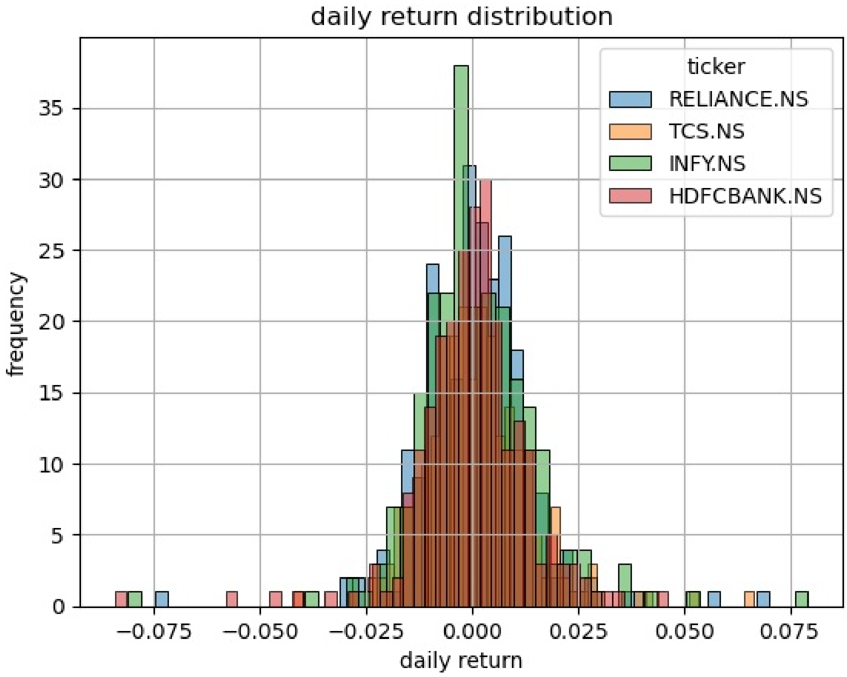
In [37]:
```
stock_data['Daily Return']= stock_data.groupby('Ticker')['Adj Close'].pct_change()
```

In [38]:
```
stock_data['Daily Return']
```

Out[38]:
```
Date
2023-07-17         NaN
2023-07-17         NaN
2023-07-17         NaN
2023-07-17         NaN
2023-07-18   -0.000834
                 ...
2024-07-11    0.003722
2024-07-12    0.000493
2024-07-12    0.035729
2024-07-12    0.010170
2024-07-12    0.066328
Name: Daily Return, Length: 972, dtype: float64
```

In [39]:
```
for i in tickers:
    ticker_data=stock_data[stock_data['Ticker']== i]
    sns.histplot(ticker_data['Daily Return'].dropna(),bins=50,label=i,alpha=0.5)
plt.title('daily return distribution')
plt.xlabel('daily return')
plt.ylabel('frequency')
```

```python
plt.legend(title='ticker')
plt.grid(True)

plt.show()
```



as dsitribution is normal (centered around zero) this shows that most of the daily returns close to avg return

now see correlation b/w these stocks

In [40]:
```python
daily_ret = stock_data.pivot_table(index='Date',columns='Ticker',values='Daily Return')
```

In [41]:
```python
daily_ret
```

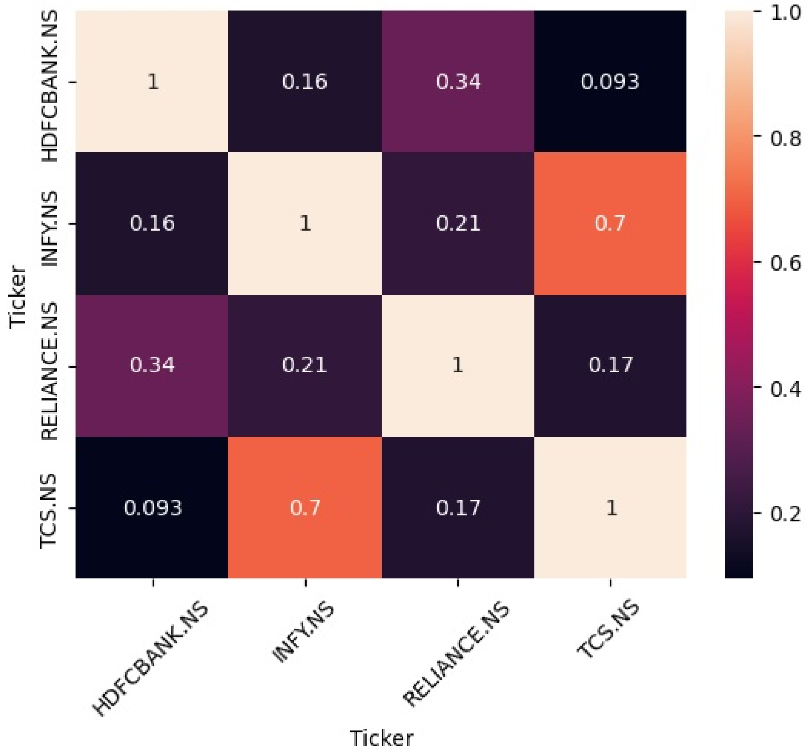
Out[41]:

Ticker	HDFCBANK.NS	INFY.NS	RELIANCE.NS	TCS.NS
Date				
2023-07-18	-0.000834	0.036719	0.008492	0.001475
2023-07-19	0.004531	-0.000169	0.007587	-0.007664
2023-07-20	0.002166	-0.017255	-0.001211	0.000650
2023-07-21	-0.007698	-0.081338	-0.030956	-0.027430
2023-07-24	0.001581	0.003755	-0.020226	0.007853
...	...	...	...	...
2024-07-08	-0.007736	0.008619	0.007727	-0.004636
2024-07-09	0.000703	-0.002708	-0.006637	-0.001928
2024-07-10	-0.006355	-0.005371	-0.003804	-0.019157
2024-07-11	-0.002583	0.002700	-0.002257	0.003722
2024-07-12	0.000493	0.035729	0.010170	0.066328

242 rows × 4 columns

In [42]:
```python
corr_matrix=daily_ret.corr()
```

In [44]:
```python
sns.heatmap(corr_matrix,annot=True)
plt.xticks(rotation=45)
plt.show()
#cmap=coolwarm
```



observation: above shows that combining stocks with lower correlation can reduce overall portfolio risk, aiming for the mix of stocks with different correlations can enhance the stability

Portfolio Optimization

using MPT can construct efficient portfolio by balancing risk and return: 1.calculate expected return and risk of each stock(volatility) 2.generate some random portfolios to identify efficient frontier 3.optimize portfolio to maxi. sharpe ratio(compare return of an investment with its risk)

In [45]:
```
#calculate expected return and volatility of stock
import numpy as np
exp_return=daily_ret.mean()* 252
# 252 shows the no. of trading days in a yr.
```

In [46]:
```
exp_return
```

Out[46]:
```
Ticker
HDFCBANK.NS    0.001264
INFY.NS        0.238076
RELIANCE.NS    0.247364
TCS.NS         0.226247
dtype: float64
```

In [47]:
```
volatility= daily_ret.std()*np.sqrt(252)
#Volatility measures the variability or dispersion of returns for a financial asset.
#It indicates how much an asset’s price fluctuates over a specific period.
#In finance, volatility is often expressed as the standard deviation of returns.
```

In [48]:
```
stock_stats=pd.DataFrame({'expected return':exp_return,'volatility':volatility})
```

In [49]:
```
stock_stats
```

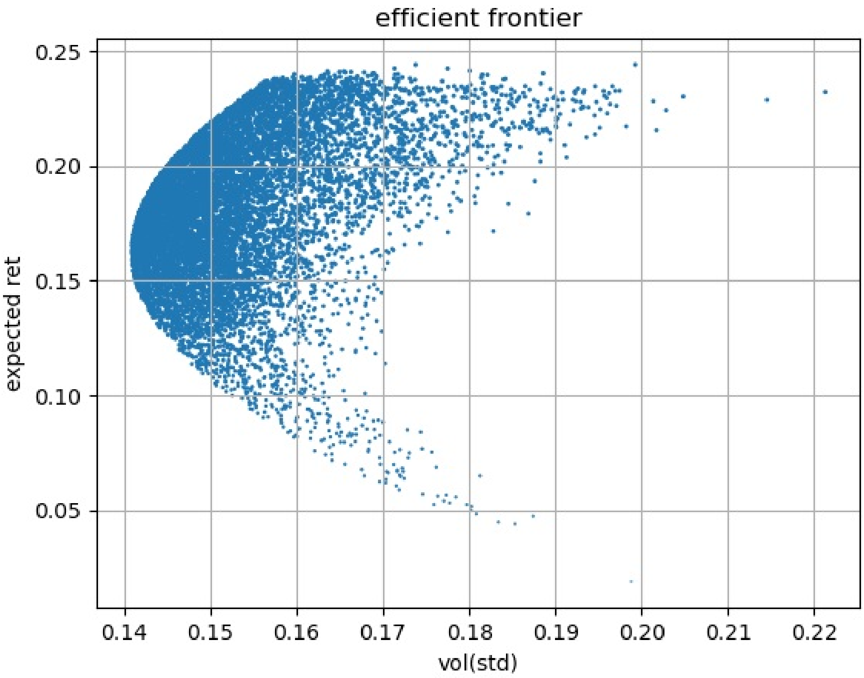
Out[49]:

Ticker	expected return	volatility
HDFCBANK.NS	0.001264	0.211722
INFY.NS	0.238076	0.230187
RELIANCE.NS	0.247364	0.210386
TCS.NS	0.226247	0.201598

Generate a large number of random portfolio weights. Calculate the expected return and volatility for each portfolio. Plot these portfolios to visualize the efficient frontier.

In [50]:
```
#function to calculate portfolio performance(?)
def portfolio_perf(weight,returns,cov_matrix):
    #dot product of weight of assest in portfolio & exp. return from each assest
    portfolio_ret=np.dot(weight,returns) #shows the expected return of the entire portfolio
    portfolio_volatility=np.sqrt(np.dot(weight.T,np.dot(cov_matrix,weight)))
    return portfolio_ret,portfolio_volatility
```

```python
#no. of portfolios to stimulate
no_portfolio=10000
#array to store result
result=np.zeros((3,no_portfolio))
#annualized cov matrix
cov_matrix=daily_ret.cov()*252
#cov matrix used to: Understand asset correlations,Construct diversified portfolios,Estimate portfolio risk and

np.random.seed(42)

for i in range(no_portfolio):
    weight=np.random.random(len(tickers))
    weight/=np.sum(weight)

    portfolio_ret,portfolio_volatility=portfolio_perf(weight,exp_return,cov_matrix)
    result[0,i]=portfolio_ret
    result[1,i]=portfolio_volatility
    result[2,i]=portfolio_ret/portfolio_volatility #sharpe ratio

plt.scatter(result[1,:],result[0,:],result[2,:])
plt.title('efficient frontier')
plt.xlabel('vol(std)')
plt.ylabel('expected ret')
plt.grid(True)
plt.show()
```



In [51]:
```python
# identify portfolio with maximum sharpe ratio:
sharpe_idx=np.argmax(result[2])
sharpe_return=result[0,sharpe_idx]
sharpe_volatility=result[1,sharpe_idx]
sharpe_ratio=result[2,sharpe_idx]
```

In [52]:
```python
sharpe_return, sharpe_volatility, sharpe_ratio
```

Out[52]:
```
(0.2364569332438419, 0.15626732371094604, 1.5131566064395254)
```

observation portfolio with maxi. sharpe ratio(a measure of risk-adjusted return) has following: expected return -> ~26% volatility -> ~15 % sharpe ratio -> 1.68

In [55]:
```python
#identify weights of the stocks in the portfolio that has maxi. sharpe ratio
max_sharpe_weight=np.zeros(len(tickers))
for i in range(no_portfolio):
    weight=np.random.random(len(tickers))
    weight /= np.sum(weight)

    portfolio_return, portfolio_volatility = portfolio_perf(weight,exp_return,cov_matrix )
    if result[2,i] == sharpe_ratio:
        max_sharpe_weight=weight
        break

portfolio_weight_df=pd.DataFrame({'Ticker':tickers,'Weight':max_sharpe_weight})
```

In [61]:
```python
portfolio_weight_df.sort_values('Weight',ascending=False)
```

Out[61]:

	Ticker	Weight
0	RELIANCE.NS	0.457059
2	INFY.NS	0.244102
1	TCS.NS	0.212806
3	HDFCBANK.NS	0.086033

Observation:- Reliance-> 45.70% INFY-> 24.41% TCS-> 21.28% HDFC-> 8.60% Reliance has the highest allocation, showing it's significant contribution to portfolio performance and HDFC has smallest allocation. This allocation aims to maxi. return and mini. risk

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js