Time Series Analysis of stock prices - Atharva Rodge

In [1]:
```
import pandas as pd # Pandas for analyzing, cleaning, exploring and manipulating the data
import numpy as np # Numpy to work with arrays
import matplotlib.pyplot as plt # Data visualization library
import seaborn as sns # advance data visualization

import warnings
from warnings import filterwarnings
filterwarnings('ignore')
```

In [2]:
```
import glob # package used to search files with extension
```

In [3]:
```
glob.glob(r'C:\Users\Atharva\Desktop\Data Analysis Course\Stock analysis\S&P_resources\individual_stocks_5yr\*csv')
```

```
ta.csv',
 'C:\\Users\\Atharva\\Desktop\\Data Analysis Course\\Stock analysis\\S&P_resources\\individual_stocks_5yr\\ECL_dat
a.csv',
 'C:\\Users\\Atharva\\Desktop\\Data Analysis Course\\Stock analysis\\S&P_resources\\individual_stocks_5yr\\ED_dat
a.csv',
 'C:\\Users\\Atharva\\Desktop\\Data Analysis Course\\Stock analysis\\S&P_resources\\individual_stocks_5yr\\EFX_dat
a.csv',
 'C:\\Users\\Atharva\\Desktop\\Data Analysis Course\\Stock analysis\\S&P_resources\\individual_stocks_5yr\\EIX_dat
a.csv',
 'C:\\Users\\Atharva\\Desktop\\Data Analysis Course\\Stock analysis\\S&P_resources\\individual_stocks_5yr\\EL_dat
a.csv',
 'C:\\Users\\Atharva\\Desktop\\Data Analysis Course\\Stock analysis\\S&P_resources\\individual_stocks_5yr\\EMN_dat
a.csv',
 'C:\\Users\\Atharva\\Desktop\\Data Analysis Course\\Stock analysis\\S&P_resources\\individual_stocks_5yr\\EMR_dat
a.csv',
 'C:\\Users\\Atharva\\Desktop\\Data Analysis Course\\Stock analysis\\S&P_resources\\individual_stocks_5yr\\EOG_dat
a.csv',
 'C:\\Users\\Atharva\\Desktop\\Data Analysis Course\\Stock analysis\\S&P_resources\\individual_stocks_5yr\\EQIX_da
ta.csv',
 'C:\\Users\\Atharva\\Desktop\\Data Analysis Course\\Stock analysis\\S&P_resources\\individual_stocks_5yr\\EQR_dat
```

In [4]:
```
len(glob.glob(r'C:\Users\Atharva\Desktop\Data Analysis Course\Stock analysis\S&P_resources\individual_stocks_5yr\*csv'
```

Out[4]: 505

- Using glob function we got all the dataset present in our folder and we can use the data as per our requirment. in this project we are going to use stock data of 4 companies - Amazon , Apple , Google , Microsoft

In [5]:
```
company_list = [
    r'C:\\Users\\Atharva\\Desktop\\Data Analysis Course\\Stock analysis\\S&P_resources\\individual_stocks_5yr\\AAPL_da
    r'C:\\Users\\Atharva\\Desktop\\Data Analysis Course\\Stock analysis\\S&P_resources\\individual_stocks_5yr\\AMZN_da
    r'C:\\Users\\Atharva\\Desktop\\Data Analysis Course\\Stock analysis\\S&P_resources\\individual_stocks_5yr\\GOOG_da
    r'C:\\Users\\Atharva\\Desktop\\Data Analysis Course\\Stock analysis\\S&P_resources\\individual_stocks_5yr\\MSFT_da
]
```

Combining Datasets using append function

In [6]:
```
# Creating a empty data frame to combine all the datasets using append function
all_stock_data = pd.DataFrame()

for file in company_list:

    current_df = pd.read_csv(file)
    all_stock_data = current_df.append(all_stock_data, ignore_index= True)
```

In [7]:
```
all_stock_data.head()
```

Out[7]:

	date	open	high	low	close	volume	Name
0	2013-02-08	27.35	27.71	27.31	27.55	33318306	MSFT
1	2013-02-11	27.65	27.92	27.50	27.86	32247549	MSFT
2	2013-02-12	27.88	28.00	27.75	27.88	35990829	MSFT
3	2013-02-13	27.93	28.11	27.88	28.03	41715530	MSFT
4	2013-02-14	27.92	28.06	27.87	28.04	32663174	MSFT

Combining datasets using pandas concat function

In [8]:
```
alternative_method = pd.DataFrame()

for file in company_list:
    df = pd.read_csv(file)
    alternative_method = pd.concat([alternative_method , df])
```

In [9]:
```
alternative_method.head()
```

Out[9]:

	date	open	high	low	close	volume	Name
0	2013-02-08	67.7142	68.4014	66.8928	67.8542	158168416	AAPL
1	2013-02-11	68.0714	69.2771	67.6071	68.5614	129029425	AAPL
2	2013-02-12	68.5014	68.9114	66.8205	66.8428	151829363	AAPL
3	2013-02-13	66.7442	67.6628	66.1742	66.7156	118721995	AAPL
4	2013-02-14	66.3599	67.3771	66.2885	66.6556	88809154	AAPL

We will use 'all_stock_data' for our analysis

In [10]:
```
all_stock_data.head(6)
```

Out[10]:

	date	open	high	low	close	volume	Name
0	2013-02-08	27.35	27.71	27.310	27.55	33318306	MSFT
1	2013-02-11	27.65	27.92	27.500	27.86	32247549	MSFT
2	2013-02-12	27.88	28.00	27.750	27.88	35990829	MSFT
3	2013-02-13	27.93	28.11	27.880	28.03	41715530	MSFT
4	2013-02-14	27.92	28.06	27.870	28.04	32663174	MSFT
5	2013-02-15	28.04	28.16	27.875	28.01	49650538	MSFT

In [11]:
```
all_stock_data['Name'].unique()
```

Out[11]:
```
array(['MSFT', 'GOOG', 'AMZN', 'AAPL'], dtype=object)
```

- Unique() function returns all the unique values present in Name column

In [12]:
```
all_stock_data.isnull().sum() # checking null values
```

Out[12]:
```
date      0
open      0
high      0
low       0
close     0
volume    0
Name      0
dtype: int64
```

In [13]:
```
all_stock_data.dtypes # Checking data types of all the features
```

Out[13]:
```
date       object
open      float64
high      float64
low       float64
close     float64
volume      int64
Name       object
dtype: object
```

- Date column is in object format but as we are performing time series analysis we need this in datetime64[ns] format to do that we will use to_datetime() function in pandas

In [14]:
```
all_stock_data['date'] = pd.to_datetime(all_stock_data['date'])
```

In [15]:
```
all_stock_data['date']
```

Out[15]:
```
0       2013-02-08
1       2013-02-11
2       2013-02-12
3       2013-02-13
4       2013-02-14
           ...
4747    2018-02-01
4748    2018-02-02
4749    2018-02-05
4750    2018-02-06
4751    2018-02-07
Name: date, Length: 4752, dtype: datetime64[ns]
```

In [16]:
```
tech_list = all_stock_data['Name'].unique() # Creating a list with all the unique tech stock present in our data
```

In [17]:
```
tech_list
```

Out[17]:
```
array(['MSFT', 'GOOG', 'AMZN', 'AAPL'], dtype=object)
```

In [18]:
```
plt.figure(figsize=(20,12))

for index , company in enumerate (tech_list, 1):
    plt.subplot(2,2, index)
    filter1 = all_stock_data['Name'] == company
    df = all_stock_data[filter1]
    plt.plot(df['date'],df['close'])
    plt.title(company)
```

Moving Average of various stocks

In [19]:
```
all_stock_data.head(15)
```

Out[19]:

	date	open	high	low	close	volume	Name
0	2013-02-08	27.3500	27.71	27.310	27.550	33318306	MSFT
1	2013-02-11	27.6500	27.92	27.500	27.860	32247549	MSFT
2	2013-02-12	27.8800	28.00	27.750	27.880	35990829	MSFT
3	2013-02-13	27.9300	28.11	27.880	28.030	41715530	MSFT
4	2013-02-14	27.9200	28.06	27.870	28.040	32663174	MSFT
5	2013-02-15	28.0400	28.16	27.875	28.010	49650538	MSFT
6	2013-02-19	27.8801	28.09	27.800	28.045	38804616	MSFT
7	2013-02-20	28.1300	28.20	27.830	27.870	44109412	MSFT
8	2013-02-21	27.7400	27.74	27.230	27.490	49078338	MSFT
9	2013-02-22	27.6800	27.76	27.480	27.760	31425726	MSFT
10	2013-02-25	27.9700	28.05	27.370	27.370	48011248	MSFT
11	2013-02-26	27.3800	27.60	27.340	27.370	49917353	MSFT
12	2013-02-27	27.4200	28.00	27.330	27.810	36390889	MSFT
13	2013-02-28	27.8800	27.97	27.740	27.800	35836861	MSFT
14	2013-03-01	27.7200	27.98	27.520	27.950	34849287	MSFT

In [20]:
```
all_stock_data['close'].rolling(window = 10).mean().head(14)
```

Out[20]:
```
0         NaN
1         NaN
2         NaN
3         NaN
4         NaN
5         NaN
6         NaN
7         NaN
8         NaN
9     27.8535
10    27.8355
11    27.7865
12    27.7795
13    27.7565
Name: close, dtype: float64
```

- The rolling method creates a rolling window object. The window=10 parameter specifies the size of the window to be 10. This means that calculations will be based on the most recent 10 data points in the series. In this context, it is used to calculate a moving average.
- After creating the rolling window object, the mean() method calculates the mean (average) of the values within each window. This will produce a new series where each value is the average of the current and previous 9 closing prices.

In [21]:
```
new_data = all_stock_data.copy() # Creating a copy of a data so that our main data remains unchanged
```

In [22]:
```
ma_days = [10,20,50]

for ma in ma_days:
    new_data['close_'+str(ma)] = new_data['close'].rolling(ma).mean()
```

In [23]:
```
new_data
```

Out[23]:

	date	open	high	low	close	volume	Name	close_10	close_20	close_50
0	2013-02-08	27.350	27.71	27.3100	27.55	33318306	MSFT	NaN	NaN	NaN
1	2013-02-11	27.650	27.92	27.5000	27.86	32247549	MSFT	NaN	NaN	NaN
2	2013-02-12	27.880	28.00	27.7500	27.88	35990829	MSFT	NaN	NaN	NaN
3	2013-02-13	27.930	28.11	27.8800	28.03	41715530	MSFT	NaN	NaN	NaN
4	2013-02-14	27.920	28.06	27.8700	28.04	32663174	MSFT	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...
4747	2018-02-01	167.165	168.62	166.7600	167.78	47230787	AAPL	171.948	173.8700	172.8252
4748	2018-02-02	166.000	166.80	160.1000	160.50	86593825	AAPL	170.152	173.2435	172.6356
4749	2018-02-05	159.100	163.88	156.0000	156.49	72738522	AAPL	168.101	172.3180	172.3026
4750	2018-02-06	154.830	163.72	154.0000	163.03	68243838	AAPL	166.700	171.7520	172.0640
4751	2018-02-07	163.085	163.40	159.0685	159.54	51608580	AAPL	165.232	171.0125	171.7554

4752 rows × 10 columns

In [24]:
```
new_data.tail(7)
```

Out[24]:

	date	open	high	low	close	volume	Name	close_10	close_20	close_50
4745	2018-01-30	165.525	167.3700	164.7000	166.97	46048185	AAPL	174.263	174.3340	172.9460
4746	2018-01-31	166.870	168.4417	166.5000	167.43	32478930	AAPL	173.096	174.0925	172.8726
4747	2018-02-01	167.165	168.6200	166.7600	167.78	47230787	AAPL	171.948	173.8700	172.8252
4748	2018-02-02	166.000	166.8000	160.1000	160.50	86593825	AAPL	170.152	173.2435	172.6356
4749	2018-02-05	159.100	163.8800	156.0000	156.49	72738522	AAPL	168.101	172.3180	172.3026
4750	2018-02-06	154.830	163.7200	154.0000	163.03	68243838	AAPL	166.700	171.7520	172.0640
4751	2018-02-07	163.085	163.4000	159.0685	159.54	51608580	AAPL	165.232	171.0125	171.7554

In [25]:
```
new_data.set_index('date', inplace = True)
```

- You can see the code line 23 where the index is number
- The set_index method in pandas is used to set one of the DataFrame's columns as the index.
- In this case, the 'date' column is chosen to be the new index of the DataFrame.
- This means the DataFrame will now use the 'date' column for its row labels instead of the default integer index.

In [26]:
```
new_data.tail()
```

Out[26]:

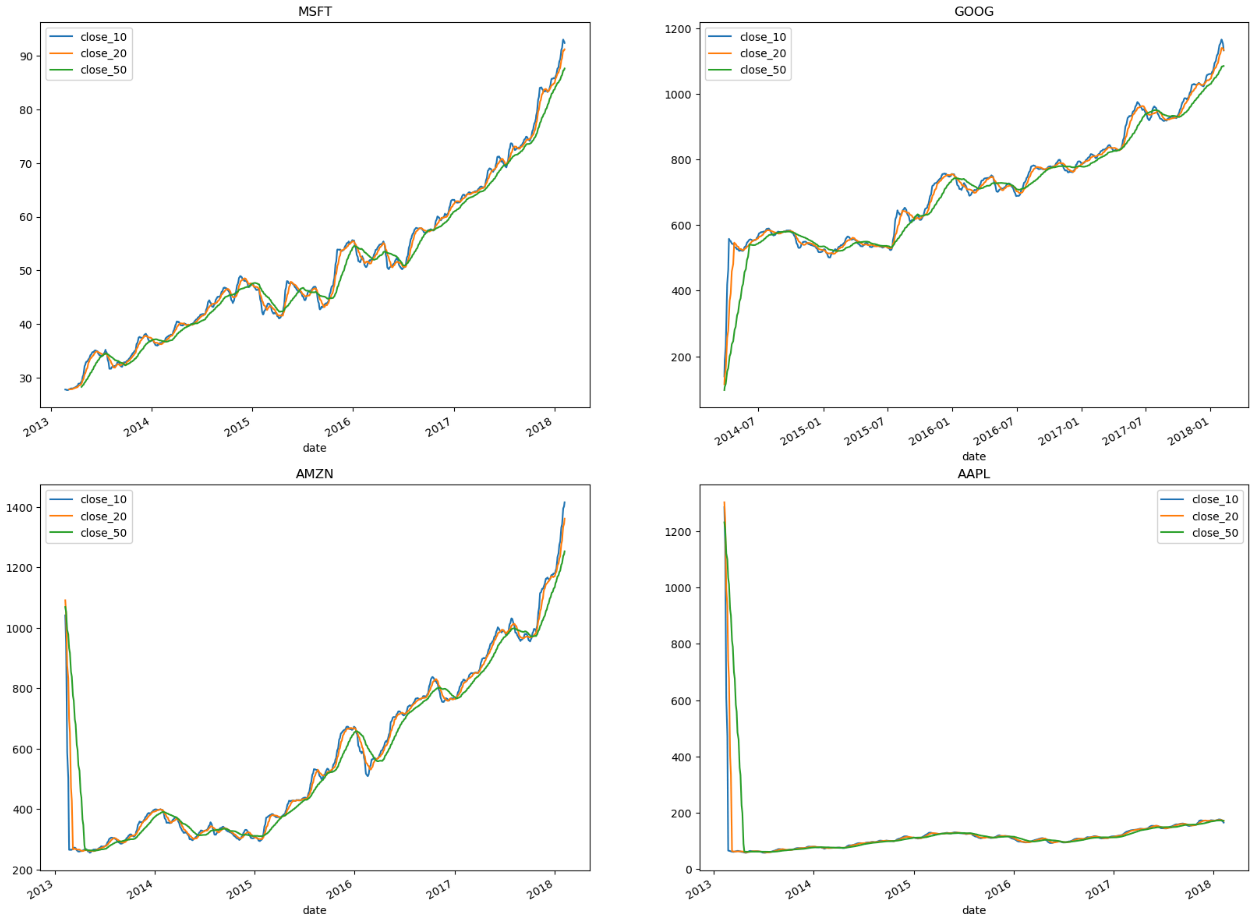
	open	high	low	close	volume	Name	close_10	close_20	close_50
date									
2018-02-01	167.165	168.62	166.7600	167.78	47230787	AAPL	171.948	173.8700	172.8252
2018-02-02	166.000	166.80	160.1000	160.50	86593825	AAPL	170.152	173.2435	172.6356
2018-02-05	159.100	163.88	156.0000	156.49	72738522	AAPL	168.101	172.3180	172.3026
2018-02-06	154.830	163.72	154.0000	163.03	68243838	AAPL	166.700	171.7520	172.0640
2018-02-07	163.085	163.40	159.0685	159.54	51608580	AAPL	165.232	171.0125	171.7554

In [27]:
```
new_data.columns
```

Out[27]:
```
Index(['open', 'high', 'low', 'close', 'volume', 'Name', 'close_10',
       'close_20', 'close_50'],
      dtype='object')
```

In [28]:
```python
plt.figure(figsize = (20,16))

for index, company in enumerate (tech_list,1):
    plt.subplot(2,2,index)
    filter_ = new_data['Name'] == company
    df = new_data[filter_]
    df[['close_10', 'close_20', 'close_50']].plot(ax = plt.gca())
    plt.title(company)
```



Observing closing price percentage change in apple stock

In [29]:
```python
company_list
```

Out[29]:
```
['C:\\\\Users\\\\Atharva\\\\Desktop\\\\Data Analysis Course\\\\Stock analysis\\\\S&P_resources\\\\individual_stocks_5
yr\\\\AAPL_data.csv',
 'C:\\\\Users\\\\Atharva\\\\Desktop\\\\Data Analysis Course\\\\Stock analysis\\\\S&P_resources\\\\individual_stocks_5
yr\\\\AMZN_data.csv',
 'C:\\\\Users\\\\Atharva\\\\Desktop\\\\Data Analysis Course\\\\Stock analysis\\\\S&P_resources\\\\individual_stocks_5
yr\\\\GOOG_data.csv',
 'C:\\\\Users\\\\Atharva\\\\Desktop\\\\Data Analysis Course\\\\Stock analysis\\\\S&P_resources\\\\individual_stocks_5
yr\\\\MSFT_data.csv']
```

In [30]:
```python
apple = pd.read_csv(company_list[0]) # We are only consideering Apple stock
```

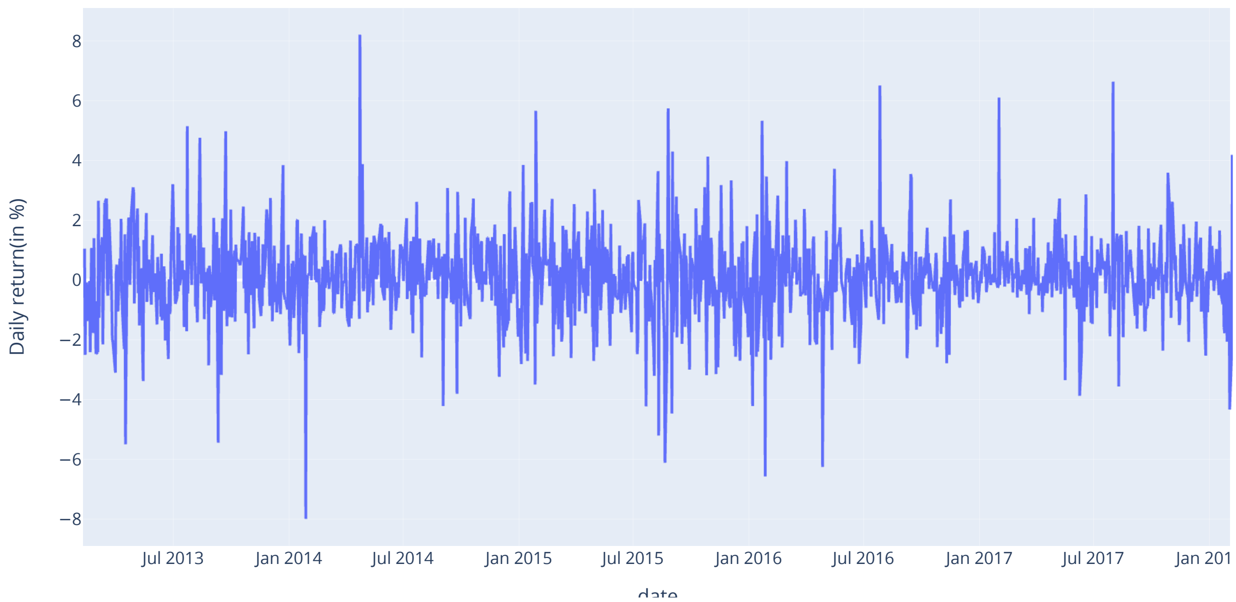
In [31]:
```python
apple.head()
```

Out[31]:

	date	open	high	low	close	volume	Name
0	2013-02-08	67.7142	68.4014	66.8928	67.8542	158168416	AAPL
1	2013-02-11	68.0714	69.2771	67.6071	68.5614	129029425	AAPL
2	2013-02-12	68.5014	68.9114	66.8205	66.8428	151829363	AAPL
3	2013-02-13	66.7442	67.6628	66.1742	66.7156	118721995	AAPL
4	2013-02-14	66.3599	67.3771	66.2885	66.6556	88809154	AAPL

In [32]:
```
apple['close']
```
Out[32]:
```
0        67.8542
1        68.5614
2        66.8428
3        66.7156
4        66.6556
          ...
1254    167.7800
1255    160.5000
1256    156.4900
1257    163.0300
1258    159.5400
Name: close, Length: 1259, dtype: float64
```
In [33]:
```
apple['Daily return(in %)'] = apple['close'].pct_change() * 100
```
In [34]:
```
apple.head()
```
Out[34]:

	date	open	high	low	close	volume	Name	Daily return(in %)
0	2013-02-08	67.7142	68.4014	66.8928	67.8542	158168416	AAPL	NaN
1	2013-02-11	68.0714	69.2771	67.6071	68.5614	129029425	AAPL	1.042235
2	2013-02-12	68.5014	68.9114	66.8205	66.8428	151829363	AAPL	-2.506658
3	2013-02-13	66.7442	67.6628	66.1742	66.7156	118721995	AAPL	-0.190297
4	2013-02-14	66.3599	67.3771	66.2885	66.6556	88809154	AAPL	-0.089934

In [35]:
```
import plotly.express as px
```
In [36]:
```
px.line(apple, x = 'date', y = 'Daily return(in %)')
```


From the above plot we can depict that the highest positive percent change was on 24th april 2014 & highest negative percent change was on 28th january 2014 this might be because of any news or and feature anouncement. we can gain such insights from the plots above

Performing resampling analysis of closing price

Resampling is a data preprocessing technique in time series analysis that involves changing the frequency of time series data. This can mean aggregating data to a higher-level frequency (downsampling) or interpolating data to a lower-level frequency (upsampling). Resampling helps in summarizing or restructuring time series data for better analysis and visualization.

Why Resample?

- Aggregation: To summarize data by aggregating over a specified time period (e.g., daily to monthly).
- Interpolation: To fill in missing data points by interpolating between existing data points (e.g., hourly to minute-level data).

- Smoothing: To smooth out short-term fluctuations and highlight longer-term trends or cycles.
- Frequency Matching: To align time series data of different frequencies for comparison or merging.

In [37]:
```
apple.dtypes
```

Out[37]:
```
date                  object
open                 float64
high                 float64
low                  float64
close                float64
volume                 int64
Name                  object
Daily return(in %)   float64
dtype: object
```

In [38]:
```
apple['date'] = pd.to_datetime(apple['date'])
```

In [39]:
```
apple.dtypes
```

Out[39]:
```
date                  datetime64[ns]
open                         float64
high                         float64
low                          float64
close                        float64
volume                         int64
Name                          object
Daily return(in %)           float64
dtype: object
```

In [40]:
```
apple.head()
```

Out[40]:

	date	open	high	low	close	volume	Name	Daily return(in %)
0	2013-02-08	67.7142	68.4014	66.8928	67.8542	158168416	AAPL	NaN
1	2013-02-11	68.0714	69.2771	67.6071	68.5614	129029425	AAPL	1.042235
2	2013-02-12	68.5014	68.9114	66.8205	66.8428	151829363	AAPL	-2.506658
3	2013-02-13	66.7442	67.6628	66.1742	66.7156	118721995	AAPL	-0.190297
4	2013-02-14	66.3599	67.3771	66.2885	66.6556	88809154	AAPL	-0.089934

In [41]:
```
apple.set_index('date', inplace = True)
```

In [42]:
```
apple.head()
```

Out[42]:

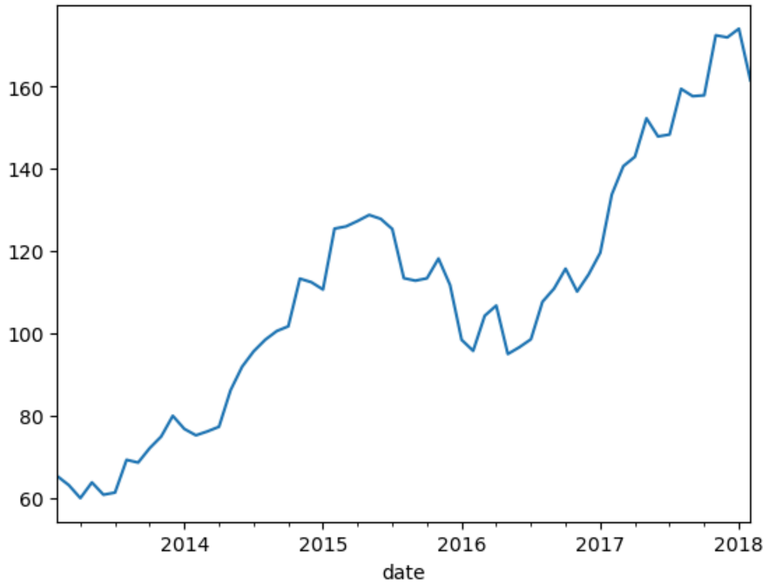
	open	high	low	close	volume	Name	Daily return(in %)
date							
2013-02-08	67.7142	68.4014	66.8928	67.8542	158168416	AAPL	NaN
2013-02-11	68.0714	69.2771	67.6071	68.5614	129029425	AAPL	1.042235
2013-02-12	68.5014	68.9114	66.8205	66.8428	151829363	AAPL	-2.506658
2013-02-13	66.7442	67.6628	66.1742	66.7156	118721995	AAPL	-0.190297
2013-02-14	66.3599	67.3771	66.2885	66.6556	88809154	AAPL	-0.089934

In [43]:
```
apple['close'].resample('M').mean() # 'M' denotes Month
```

Out[43]:
```
date
2013-02-28     65.306264
2013-03-31     63.120110
2013-04-30     59.966432
2013-05-31     63.778927
2013-06-30     60.791120
                 ...
2017-10-31    157.817273
2017-11-30    172.406190
2017-12-31    171.891500
2018-01-31    174.005238
2018-02-28    161.468000
Freq: M, Name: close, Length: 61, dtype: float64
```

In [44]:
```
apple['close'].resample('M').mean().plot()
```

Out[44]: <Axes: xlabel='date'>

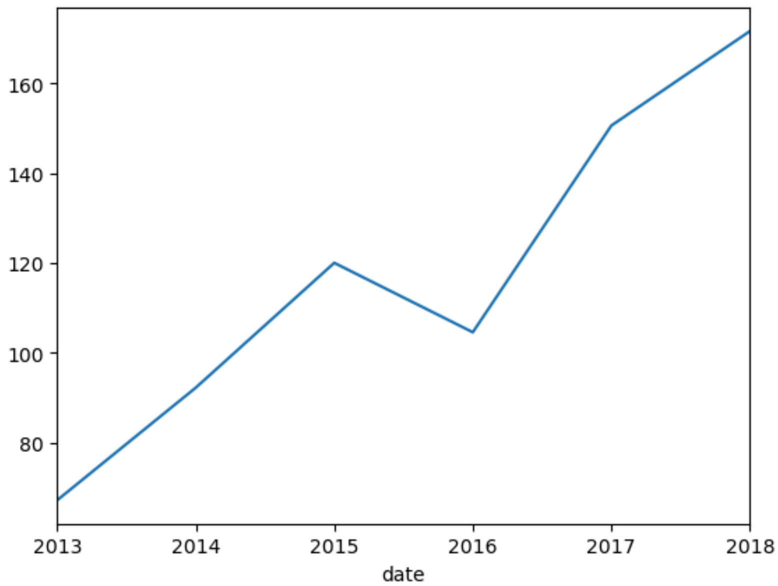


In [45]:
```
apple['close'].resample('Y').mean() # 'Y' denotes Year
```

Out[45]:
```
date
2013-12-31     67.237839
2014-12-31     92.264531
2015-12-31    120.039861
2016-12-31    104.604008
2017-12-31    150.585080
2018-12-31    171.594231
Freq: A-DEC, Name: close, dtype: float64
```

In [46]:
```
apple['close'].resample('Y').mean().plot().plot()
```

Out[46]: []

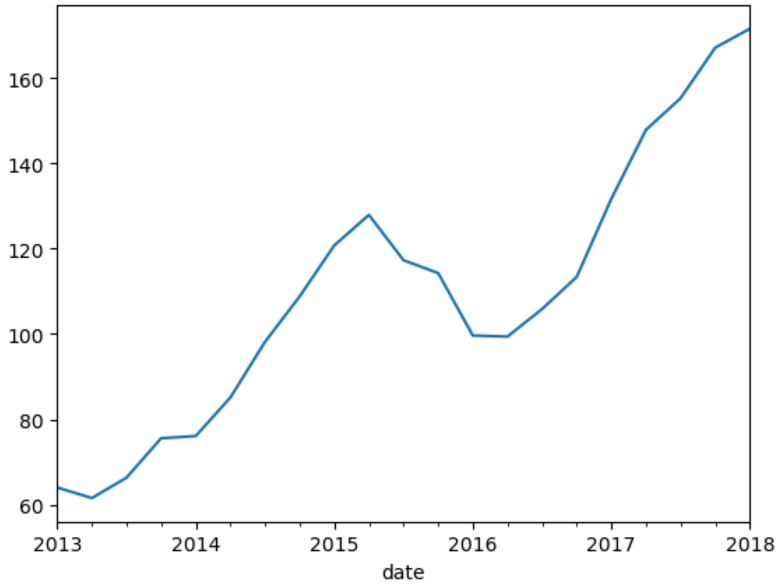


In [47]:
```python
apple['close'].resample('Q').mean() # 'Q' Denotes Quater
```

Out[47]:
```
date
2013-03-31     64.020291
2013-06-30     61.534692
2013-09-30     66.320670
2013-12-31     75.567478
2014-03-31     76.086293
2014-06-30     85.117475
2014-09-30     98.163311
2014-12-31    108.821016
2015-03-31    120.776721
2015-06-30    127.937937
2015-09-30    117.303438
2015-12-31    114.299297
2016-03-31     99.655082
2016-06-30     99.401250
2016-09-30    105.866094
2016-12-31    113.399048
2017-03-31    131.712500
2017-06-30    147.875397
2017-09-30    155.304603
2017-12-31    167.148254
2018-03-31    171.594231
Freq: Q-DEC, Name: close, dtype: float64
```

In [48]:
```python
apple['close'].resample('Q').mean().plot()
```

Out[48]: <Axes: xlabel='date'>

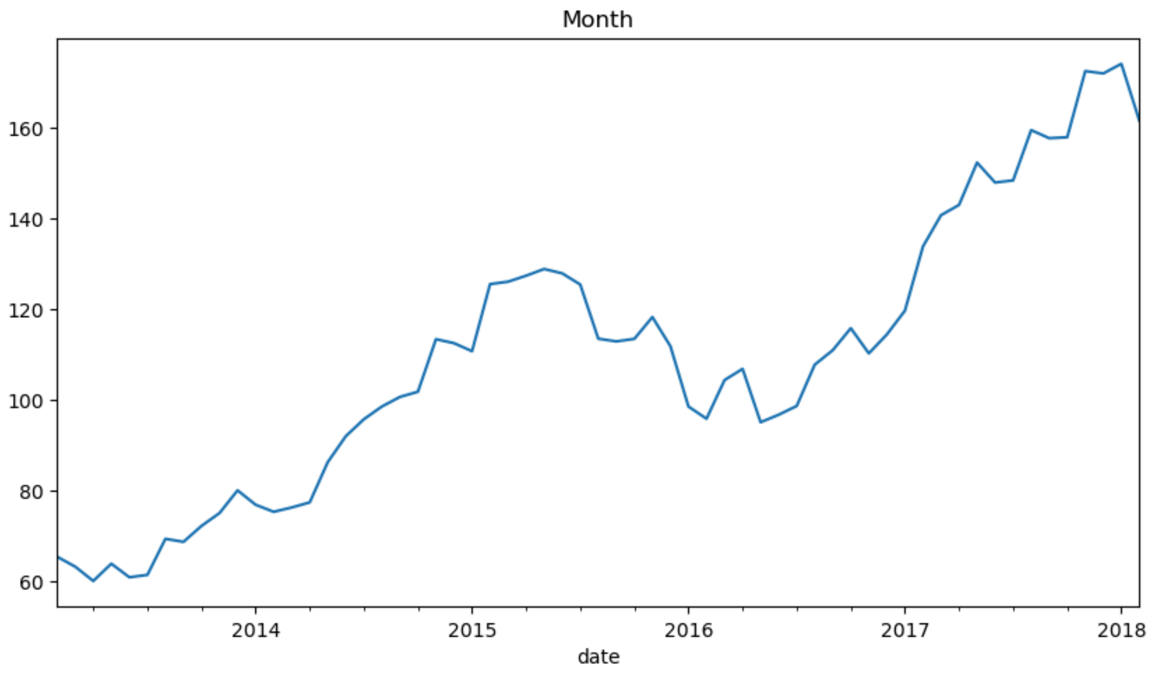


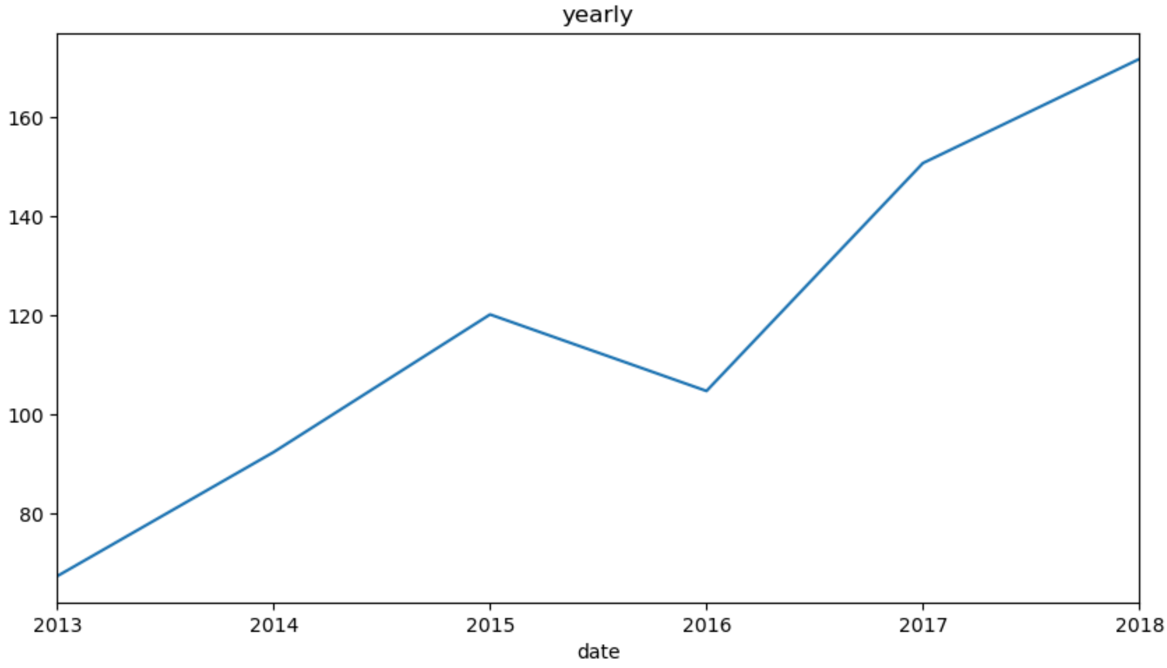
Ploting all the graphs together

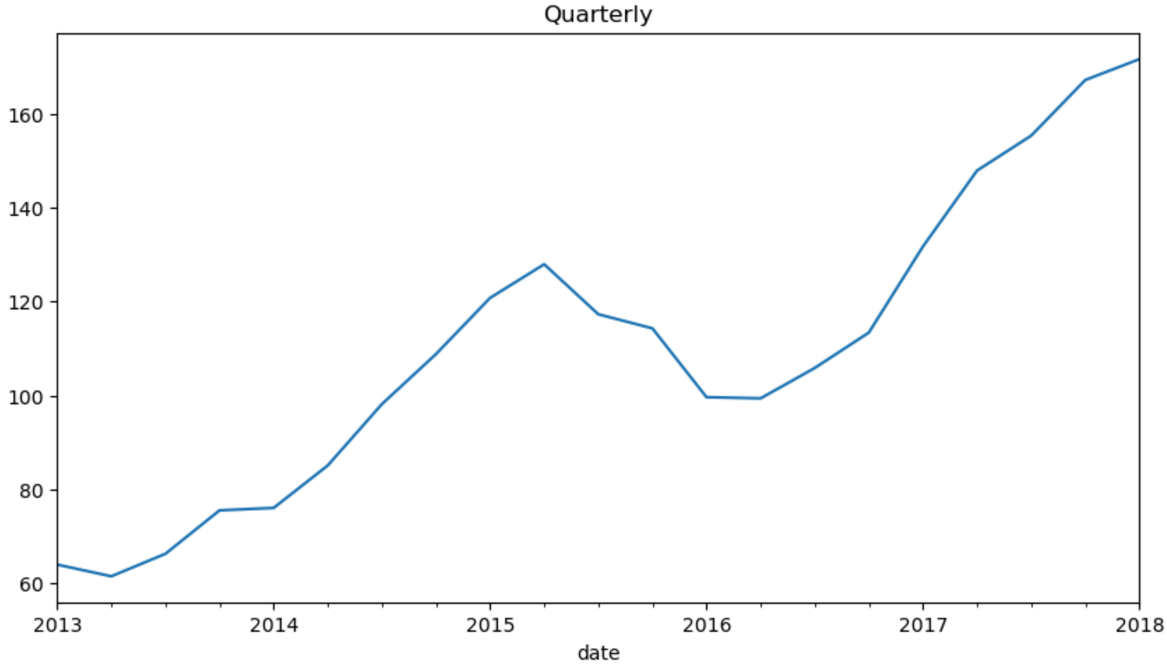
In [49]:
```
resampling_list = ['M','Y','Q']

plt.figure(figsize = (10,18))

for index , sample in enumerate (resampling_list , 1):
    plt.subplot(3,1,index)
    apple['close'].resample(f'{sample}').mean().plot()
    if sample == 'M':
        plt.title('Month')
    elif sample == 'Y':
        plt.title('yearly')
    else: plt.title('Quarterly')
```


Month
160
140
120
100
80
60
2014
2015
2016
2017
2018
date


yearly
160
140
120
100
80
2013
2014
2015
2016
2017
2018
date


Quarterly
160
140
120
100
80
60
2013
2014
2015
2016
2017
2018
date

Performing multivariate analysis to understand co-relation

- checking if the closing prices of these tech companies (amazon , apple , google , microsoft) are correlated with each other

In [50]:
```
company_list
```

Out[50]:
```
['C:\\\\Users\\\\Atharva\\\\Desktop\\\\Data Analysis Course\\\\Stock analysis\\\\S&P_resources\\\\individual_stocks_5
yr\\\\AAPL_data.csv',
 'C:\\\\Users\\\\Atharva\\\\Desktop\\\\Data Analysis Course\\\\Stock analysis\\\\S&P_resources\\\\individual_stocks_5
yr\\\\AMZN_data.csv',
 'C:\\\\Users\\\\Atharva\\\\Desktop\\\\Data Analysis Course\\\\Stock analysis\\\\S&P_resources\\\\individual_stocks_5
yr\\\\GOOG_data.csv',
 'C:\\\\Users\\\\Atharva\\\\Desktop\\\\Data Analysis Course\\\\Stock analysis\\\\S&P_resources\\\\individual_stocks_5
yr\\\\MSFT_data.csv']
```

In [51]:
```
app = pd.read_csv(company_list[0])
```

In [52]:
```
amzn = pd.read_csv(company_list[1])
```

In [53]:
```
google = pd.read_csv(company_list[2])
```

In [54]:
```
msft = pd.read_csv(company_list[3])
```

In [55]:
```
closing_p = pd.DataFrame()
```

In [56]:
```
closing_p['apple_close'] = app['close']
closing_p['amzn_close'] = amzn['close']
closing_p['google_close'] = google['close']
closing_p['msft_close'] = msft['close']
```

In [57]:
```
closing_p
```

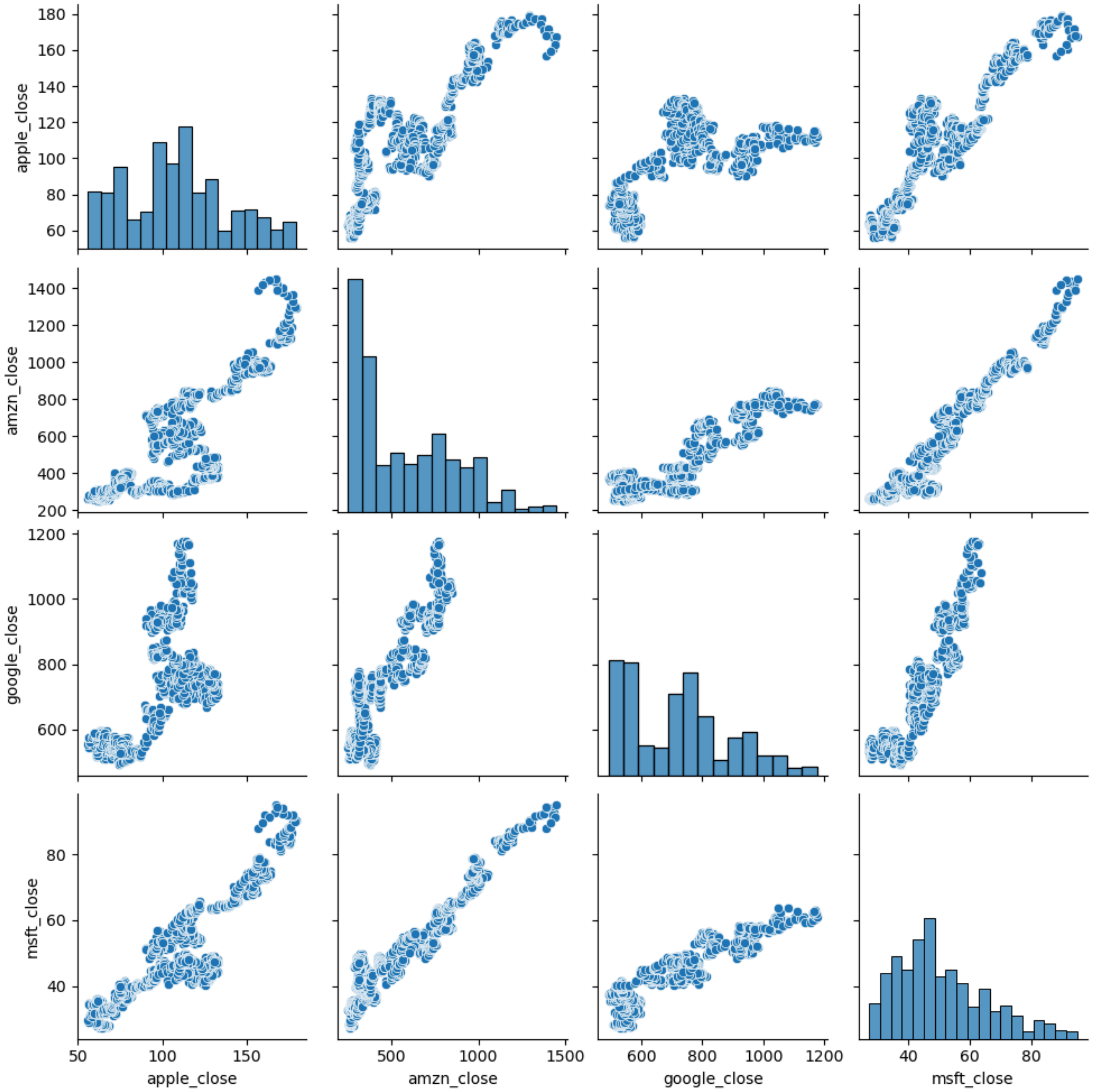
Out[57]:

	apple_close	amzn_close	google_close	msft_close
0	67.8542	261.95	558.46	27.55
1	68.5614	257.21	559.99	27.86
2	66.8428	258.70	556.97	27.88
3	66.7156	269.47	567.16	28.03
4	66.6556	269.24	567.00	28.04
...	...	...	...	...
1254	167.7800	1390.00	NaN	94.26
1255	160.5000	1429.95	NaN	91.78
1256	156.4900	1390.00	NaN	88.00
1257	163.0300	1442.84	NaN	91.33
1258	159.5400	1416.78	NaN	89.61

1259 rows × 4 columns

In [58]:
```
sns.pairplot(closing_p)
```

Out[58]: <seaborn.axisgrid.PairGrid at 0x290f08ec370>



- The sns.pairplot(closing_p) function in the Seaborn library is used to create a grid of scatter plots and histograms for the pairwise relationships between the columns in a DataFrame. This is a form of exploratory data analysis (EDA) that helps visualize the relationships between multiple variables in a dataset.

In [59]:
```
closing_p.corr()
```

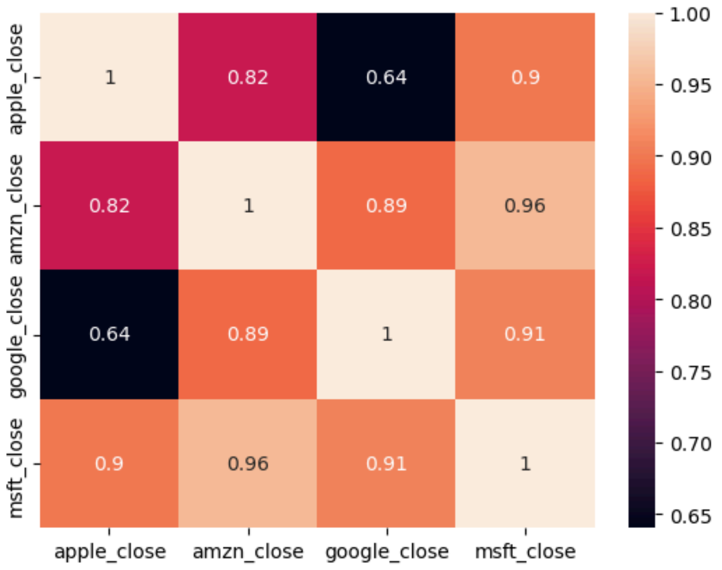
Out[59]:

	apple_close	amzn_close	google_close	msft_close
apple_close	1.000000	0.819078	0.640522	0.899689
amzn_close	0.819078	1.000000	0.888456	0.955977
google_close	0.640522	0.888456	1.000000	0.907011
msft_close	0.899689	0.955977	0.907011	1.000000

- Correlation is used to depict the correlation between two variables

In [60]:
```python
sns.heatmap(closing_p.corr(), annot= True)
```

Out[60]: <Axes: >



Performing Correlation analysis

- Analyze whether daily change in closing price od stock or daily returns in stock are correlated or not

In [61]:
```python
company_close_pct = pd.DataFrame()
```

In [62]:
```python
company_close_pct['AAPL_close_pct_change'] = app['close'].pct_change() * 100
company_close_pct['AMZN_close_pct_change'] = amzn['close'].pct_change() * 100
company_close_pct['GOOG_close_pct_change'] = google['close'].pct_change() * 100
company_close_pct['MSFT_close_pct_change'] = msft['close'].pct_change() * 100
```

In [63]:
```python
company_close_pct.head()
```

Out[63]:

	AAPL_close_pct_change	AMZN_close_pct_change	GOOG_close_pct_change	MSFT_close_pct_change
0	NaN	NaN	NaN	NaN
1	1.042235	-1.809506	0.273968	1.125227
2	-2.506658	0.579293	-0.539295	0.071788
3	-0.190297	4.163123	1.829542	0.538020
4	-0.089934	-0.085353	-0.028211	0.035676

Different approach to calculate percent change

In [64]:
```python
closing_p['apple_close']
```

Out[64]:
```
0        67.8542
1        68.5614
2        66.8428
3        66.7156
4        66.6556
          ...
1254    167.7800
1255    160.5000
1256    156.4900
1257    163.0300
1258    159.5400
Name: apple_close, Length: 1259, dtype: float64
```

In [65]:
```
closing_p['apple_close'].shift(1)
```

Out[65]:
```
0            NaN
1        67.8542
2        68.5614
3        66.8428
4        66.7156
          ...
1254    167.4300
1255    167.7800
1256    160.5000
1257    156.4900
1258    163.0300
Name: apple_close, Length: 1259, dtype: float64
```

In [66]:
```
(closing_p['apple_close'] - closing_p['apple_close'].shift(1)) / closing_p['apple_close'].shift(1) * 100
```

Out[66]:
```
0            NaN
1       1.042235
2      -2.506658
3      -0.190297
4      -0.089934
          ...
1254    0.209043
1255   -4.339015
1256   -2.498442
1257    4.179181
1258   -2.140710
Name: apple_close, Length: 1259, dtype: float64
```

In [67]:
```
for col in closing_p.columns:
    closing_p[col+'_pct_change'] = (closing_p[col] - closing_p[col].shift(1)) / closing_p[col].shift(1) * 100
```

In [68]:
```
closing_p.columns
```

Out[68]:
```
Index(['apple_close', 'amzn_close', 'google_close', 'msft_close',
       'apple_close_pct_change', 'amzn_close_pct_change',
       'google_close_pct_change', 'msft_close_pct_change'],
      dtype='object')
```

In [69]:
```
close_p = closing_p[['apple_close_pct_change', 'amzn_close_pct_change',
       'google_close_pct_change', 'msft_close_pct_change']]
```

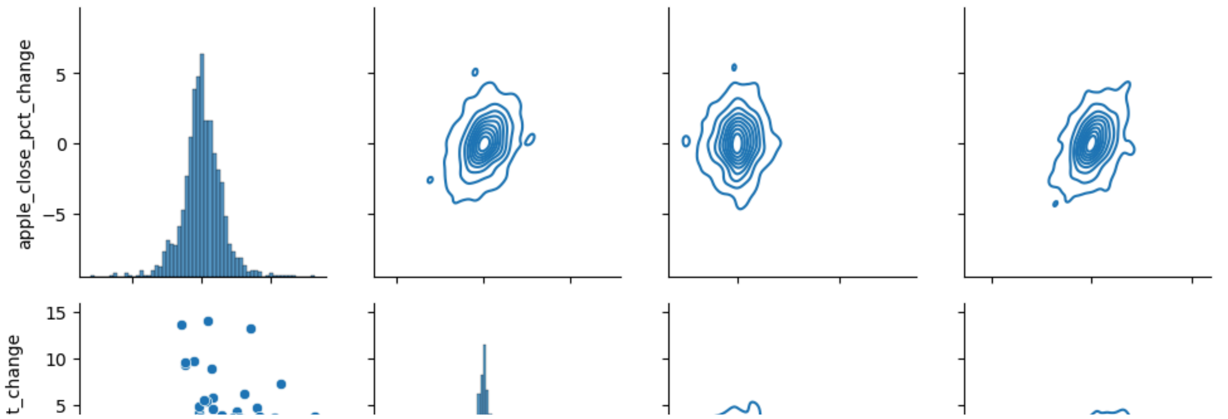
In [70]:
```
close_p.head()
```

Out[70]:

	apple_close_pct_change	amzn_close_pct_change	google_close_pct_change	msft_close_pct_change
0	NaN	NaN	NaN	NaN
1	1.042235	-1.809506	0.273968	1.125227
2	-2.506658	0.579293	-0.539295	0.071788
3	-0.190297	4.163123	1.829542	0.538020
4	-0.089934	-0.085353	-0.028211	0.035676

In [71]:
```
g = sns.PairGrid( data = close_p)
g.map_diag(sns.histplot)
g.map_lower(sns.scatterplot)
g.map_upper(sns.kdeplot)
```

Out[71]: <seaborn.axisgrid.PairGrid at 0x290f0721870>



- seaborn.PairGrid is a more flexible and customizable plotting version of seaborn.pairplot in the Seaborn library for creating grids of plots, but with more control over the individual plots in the grid. It allows for greater customization of the types of plots that appear in each part of the grid, and it is particularly useful when you need to create complex and highly customized visualizations. eg we can plot histogram , scatterplot and kdeplot as per our convinience below or above diagonal

In [72]:
```
close_p.corr()
```

Out[72]:

	apple_close_pct_change	amzn_close_pct_change	google_close_pct_change	msft_close_pct_change
apple_close_pct_change	1.000000	0.287659	0.036202	0.366598
amzn_close_pct_change	0.287659	1.000000	0.027698	0.402678
google_close_pct_change	0.036202	0.027698	1.000000	0.038939
msft_close_pct_change	0.366598	0.402678	0.038939	1.000000