Financial analysis 💰 💵 📈 of McDonald's 🍔 🍟 and STARBUCKS COFFEE ☕ 🥤

- McDonald's Corporation is an American multinational fast food chain which is the world's largest fast food chain.

- Starbucks Corporation is an American multinational chain of coffeehouses which is the world's largest coffeehouse chain.

Importing necessary libraries

In [1]:
```
from matplotlib import dates
import matplotlib.pyplot as plt
import yfinance as yf
import pandas as pd
import numpy as np
import datetime as dt
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

Step 1: Specifying date range for analysis

In [3]:
```
end = dt.datetime.now()
start = end - dt.timedelta(days = 5000)
start, end
```

Out[3]:
```
(datetime.datetime(2009, 10, 17, 23, 57, 32, 172155),
 datetime.datetime(2023, 6, 26, 23, 57, 32, 172155))
```

In [4]:
```
end = dt.datetime.now()
start = dt.datetime(2013,1, 1)
start, end
```

Out[4]:
```
(datetime.datetime(2013, 1, 1, 0, 0),
 datetime.datetime(2023, 6, 26, 23, 57, 32, 190156))
```

Step 2: Selecting the stocks/tickers for the analysis

For current ananlysis Mc. donald and Starbucks as selected as tickers

In [5]:
```
ticks = ['MCD', 'SBUX']
```

Step 3: Loading Data using yfinance library

yfinance library is used to load daily records of Mc. donald and Starbucks data startting from 2013 until today.

In [6]:
```
# downloading Historical Dataset
#data = yf.download('MCD SBUX' ,  start = '2013-01-01', end = '2023-06-21')

data = yf.download(ticks, start, end)
```

```
[**********100%**********]  2 of 2 completed
```

In [7]:
```
type(data)
```

Out[7]:
```
pandas.core.frame.DataFrame
```

In [8]:
```
data.tail(10).round(2)
```

Out[8]:

	Adj Close		Close		High		Low		Open		Volume	
	MCD	SBUX	MCD	SBUX	MCD	SBUX	MCD	SBUX	MCD	SBUX	MCD	SBUX
Date												
2023-06-12	288.57	98.46	288.57	98.46	288.75	98.51	286.50	97.52	288.16	97.96	1885000	6705800
2023-06-13	288.55	99.26	288.55	99.26	289.69	99.47	287.55	98.51	288.50	98.51	2027800	5137900
2023-06-14	288.44	100.66	288.44	100.66	289.56	101.36	287.31	99.19	288.10	99.56	1964500	9279500
2023-06-15	292.61	101.38	292.61	101.38	293.48	101.99	288.26	100.66	289.65	101.34	2619200	6448200
2023-06-16	293.70	101.87	293.70	101.87	296.57	102.51	293.14	101.48	294.51	102.01	4890400	11708300
2023-06-20	293.04	101.27	293.04	101.27	297.18	102.49	292.68	100.90	293.65	101.86	2729800	5506400
2023-06-21	294.52	101.87	294.52	101.87	295.16	102.46	292.81	100.36	293.32	100.60	2837100	5508800
2023-06-22	293.30	100.85	293.30	100.85	295.08	101.64	291.52	99.64	294.62	101.42	1799400	6125500
2023-06-23	289.91	98.34	289.91	98.34	292.45	99.73	289.64	97.52	291.35	99.65	3719700	18765000
2023-06-26	287.20	98.36	287.20	98.36	289.75	98.64	287.07	97.48	289.64	98.34	402173	1250732

In [9]:
```
data.T.round(2)
```

Out[9]:

	Date	2013-01-02	2013-01-03	2013-01-04	2013-01-07	2013-01-08	2013-01-09	2013-01-10	2013-01-11	2013-01-14
Adj Close	**MCD**	67.76	68.15	67.56	68.36	68.38	68.28	68.73	68.98	68.82
	SBUX	22.83	22.99	23.12	23.13	23.09	22.68	22.64	22.84	22.70
Close	**MCD**	90.12	90.63	89.85	90.91	90.94	90.81	91.40	91.73	91.53
	SBUX	27.50	27.68	27.84	27.86	27.81	27.32	27.27	27.50	27.33
High	**MCD**	90.35	90.70	90.79	91.05	90.96	91.24	91.64	91.98	92.12
	SBUX	27.50	27.81	28.00	27.90	27.86	27.95	27.58	27.62	27.75
Low	**MCD**	89.33	90.16	89.67	89.25	89.97	90.36	91.03	91.05	91.43
	SBUX	27.13	27.50	27.66	27.50	27.53	27.17	27.15	27.22	27.27
Open	**MCD**	89.40	90.31	90.62	89.77	90.54	90.80	91.35	91.94	91.76
	SBUX	27.30	27.53	27.76	27.70	27.79	27.94	27.38	27.56	27.55
Volume	**MCD**	7377200.00	5473500.00	5374100.00	5820900.00	6285200.00	4822100.00	4393300.00	3692600.00	3828100.00
	SBUX	13267600.00	14670400.00	10911400.00	8720000.00	9613400.00	16678400.00	14367600.00	13302600.00	11596400.00

12 rows × 2638 columns

Step 4: Understanding the dataset

In [10]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2638 entries, 2013-01-02 to 2023-06-26
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   (Adj Close, MCD)   2638 non-null   float64
 1   (Adj Close, SBUX)  2638 non-null   float64
 2   (Close, MCD)       2638 non-null   float64
 3   (Close, SBUX)      2638 non-null   float64
 4   (High, MCD)        2638 non-null   float64
 5   (High, SBUX)       2638 non-null   float64
 6   (Low, MCD)         2638 non-null   float64
 7   (Low, SBUX)        2638 non-null   float64
 8   (Open, MCD)        2638 non-null   float64
 9   (Open, SBUX)       2638 non-null   float64
 10  (Volume, MCD)      2638 non-null   int64  
 11  (Volume, SBUX)     2638 non-null   int64  
dtypes: float64(10), int64(2)
memory usage: 332.5 KB
```

In [11]:
```
data.describe().round(2)
```

Out[11]:

	Adj Close		Close		High		Low		Open		Volume	
	MCD	SBUX	MCD	SBUX	MCD	SBUX	MCD	SBUX	MCD	SBUX	MCD	SBUX
count	2638.00	2638.00	2638.00	2638.00	2638.00	2638.00	2638.00	2638.00	2638.00	2638.00	2638.00	2638.00
mean	151.08	62.45	166.96	67.75	168.14	68.36	165.72	67.10	166.92	67.73	4203658.33	8867874.16
std	65.68	25.78	60.56	24.77	61.06	25.01	60.05	24.51	60.57	24.76	2300234.52	4538380.17
min	67.56	22.17	88.46	26.60	89.82	26.82	87.50	26.26	88.07	26.53	402173.00	1250732.00
25%	80.64	46.37	101.64	52.64	102.13	53.10	101.15	52.03	101.58	52.53	2729800.00	6089125.00
50%	143.22	52.51	161.72	59.35	163.18	59.69	160.50	58.73	161.89	59.23	3614600.00	7811750.00
75%	201.02	82.95	215.16	87.15	217.09	88.22	213.62	86.19	215.42	87.27	4984975.00	10314775.00
max	296.50	120.92	298.07	126.06	298.86	126.32	296.30	124.81	298.63	126.08	25286600.00	62091100.00

Step 5: Index of stock data

Unlike, pandas library where the index is numeric value, the index in yfinance library is date. Each row represents a day.

In [12]:
```
data.index
```

Out[12]:
```
DatetimeIndex(['2013-01-02', '2013-01-03', '2013-01-04', '2013-01-07',
               '2013-01-08', '2013-01-09', '2013-01-10', '2013-01-11',
               '2013-01-14', '2013-01-15',
               ...
               '2023-06-12', '2023-06-13', '2023-06-14', '2023-06-15',
               '2023-06-16', '2023-06-20', '2023-06-21', '2023-06-22',
               '2023-06-23', '2023-06-26'],
              dtype='datetime64[ns]', name='Date', length=2638, freq=None)
```

Normally in pandas data frame the index are numerical, however the yfinance library is sticking in the date.

- So this is daily information.
- Each row represents a single day.

Step 6: Multi indexed columns of stock data

In [13]:
```
columns = data.columns
columns
```

Out[13]:
```
MultiIndex([('Adj Close',  'MCD'),
            ('Adj Close', 'SBUX'),
            (    'Close',  'MCD'),
            (    'Close', 'SBUX'),
            (     'High',  'MCD'),
            (     'High', 'SBUX'),
            (      'Low',  'MCD'),
            (      'Low', 'SBUX'),
            (     'Open',  'MCD'),
            (     'Open', 'SBUX'),
            (   'Volume',  'MCD'),
            (   'Volume', 'SBUX')],
           )
```

We have hirrarcheal or multi indexed columns here.

- Two sub columns are repeated under each outer column

Step 7: Acessing feature

In [14]:
```
# Closing stocks of McDonald and Starbucks
data['Close']
```

Out[14]:

Date	MCD	SBUX
2013-01-02	90.120003	27.500000
2013-01-03	90.629997	27.684999
2013-01-04	89.849998	27.844999
2013-01-07	90.910004	27.860001
2013-01-08	90.940002	27.809999
...	...	...
2023-06-20	293.040009	101.269997
2023-06-21	294.519989	101.870003
2023-06-22	293.299988	100.849998
2023-06-23	289.910004	98.339996
2023-06-26	287.200012	98.360001

2638 rows × 2 columns

In [15]:
```
# Closing stocks of McDonald
data['Close'][['MCD']]
```

Out[15]:

Date	MCD
2013-01-02	90.120003
2013-01-03	90.629997
2013-01-04	89.849998
2013-01-07	90.910004
2013-01-08	90.940002
...	...
2023-06-20	293.040009
2023-06-21	294.519989
2023-06-22	293.299988
2023-06-23	289.910004
2023-06-26	287.200012

2638 rows × 1 columns

In [16]:
```
# Closing stocks of Starbucks
data['Close'][['SBUX']]
```

Out[16]:

Date	SBUX
2013-01-02	27.500000
2013-01-03	27.684999
2013-01-04	27.844999
2013-01-07	27.860001
2013-01-08	27.809999
...	...
2023-06-20	101.269997
2023-06-21	101.870003
2023-06-22	100.849998
2023-06-23	98.339996
2023-06-26	98.360001

2638 rows × 1 columns

outer columns:

In [17]:
```
# to get outer column names as a list:
outer_columns = list()
for col in columns:
    outer_columns.append(col[0])
outer = list(set(outer_columns))
print(outer)
```

```
['Low', 'Close', 'Volume', 'Open', 'High', 'Adj Close']
```

inner columns:

In [18]:
```
# to get inner column names as a list:

inner_columns = list()
for col in columns:
    inner_columns.append(col[1])
inner = list(set(inner_columns))
print(inner)
```

```
['SBUX', 'MCD']
```

In [19]:
```
def column_outer(data):
    columns = data.columns
    outer_col = [col[0] for col in columns]
    data.columns = outer_col
    return data
```

- The function takes dataframe returns the dataframe.
- Inside the function, it pulls all the columns and taking the 1st input from the tupple that gives multi indexed columns as a list and set the columns

In [20]:
```
# Reading  each alternate column starting from column index 0 : for 'AAPL'
# outer_MCD = column_outer(data).iloc[: , ::2]
# outer_MCD
```

In [21]:
```
# Reading  each alternate column starting from column index 1 : for 'SPY'
# outer_SBUX = column_outer(data).iloc[: , 1::2]
# outer_SBUX
```

In [22]:
```
# using pipe method  syntax for pipe method : ( data.pipe(func))
#data.pipe?
```

pipe method

```
''' data.pipe(func) : pipe method creats a pipeline for data processing.
    It is a method apples chainable function '''
```

In [23]:
```
# MCD data
(data.iloc[: , ::2].pipe(column_outer))
```

Out[23]:

Date	Adj Close	Close	High	Low	Open	Volume
2013-01-02	67.764641	90.120003	90.349998	89.330002	89.400002	7377200
2013-01-03	68.148117	90.629997	90.699997	90.160004	90.309998	5473500
2013-01-04	67.561623	89.849998	90.790001	89.669998	90.620003	5374100
2013-01-07	68.358658	90.910004	91.050003	89.250000	89.769997	5820900
2013-01-08	68.381241	90.940002	90.959999	89.970001	90.540001	6285200
...	...	...	...	...	...	...
2023-06-20	293.040009	293.040009	297.179993	292.679993	293.649994	2729800
2023-06-21	294.519989	294.519989	295.160004	292.809998	293.320007	2837100
2023-06-22	293.299988	293.299988	295.079987	291.519989	294.619995	1799400
2023-06-23	289.910004	289.910004	292.450012	289.640015	291.350006	3719700
2023-06-26	287.200012	287.200012	289.750000	287.070007	289.640015	402173

2638 rows × 6 columns

In [24]:
```
#SUBX data
(data.iloc[: , 1:-1:2].pipe(column_outer))
```

Out[24]:

Date	Adj Close	Close	High	Low	Open
2013-01-02	22.833775	27.500000	27.500000	27.129999	27.295000
2013-01-03	22.987375	27.684999	27.805000	27.500000	27.535000
2013-01-04	23.120226	27.844999	28.000000	27.655001	27.764999
2013-01-07	23.132689	27.860001	27.895000	27.504999	27.700001
2013-01-08	23.091167	27.809999	27.860001	27.535000	27.790001
...	...	...	...	...	...
2023-06-20	101.269997	101.269997	102.489998	100.900002	101.860001
2023-06-21	101.870003	101.870003	102.459999	100.360001	100.599998
2023-06-22	100.849998	100.849998	101.639999	99.639999	101.419998
2023-06-23	98.339996	98.339996	99.730003	97.519997	99.650002
2023-06-26	98.360001	98.360001	98.639999	97.480003	98.339996

2638 rows × 5 columns

Step 8: Analysising Mc.Donald data

In [25]:
```
def tweak_data_mcd():
    data = yf.download(ticks, start, end )
    mcd = data.iloc[: , ::2]
    return (mcd.pipe(column_outer))
mc = tweak_data_mcd()
mc
```

```
[**********100%**********]  2 of 2 completed
```

Out[25]:

Date	Adj Close	Close	High	Low	Open	Volume
2013-01-02	67.764641	90.120003	90.349998	89.330002	89.400002	7377200
2013-01-03	68.148117	90.629997	90.699997	90.160004	90.309998	5473500
2013-01-04	67.561623	89.849998	90.790001	89.669998	90.620003	5374100
2013-01-07	68.358658	90.910004	91.050003	89.250000	89.769997	5820900
2013-01-08	68.381241	90.940002	90.959999	89.970001	90.540001	6285200
...	...	...	...	...	...	...
2023-06-20	293.040009	293.040009	297.179993	292.679993	293.649994	2729800
2023-06-21	294.519989	294.519989	295.160004	292.809998	293.320007	2837100
2023-06-22	293.299988	293.299988	295.079987	291.519989	294.619995	1799400
2023-06-23	289.910004	289.910004	292.450012	289.640015	291.350006	3719700
2023-06-26	287.200012	287.200012	289.750000	287.070007	289.640015	402173

2638 rows × 6 columns

In [26]:
```
(data.iloc[: , ::2]
#  .pipe(column_outer)
)
```

Out[26]:

	Adj Close	Close	High	Low	Open	Volume
	MCD	MCD	MCD	MCD	MCD	MCD
Date						
2013-01-02	67.764641	90.120003	90.349998	89.330002	89.400002	7377200
2013-01-03	68.148117	90.629997	90.699997	90.160004	90.309998	5473500
2013-01-04	67.561623	89.849998	90.790001	89.669998	90.620003	5374100
2013-01-07	68.358658	90.910004	91.050003	89.250000	89.769997	5820900
2013-01-08	68.381241	90.940002	90.959999	89.970001	90.540001	6285200
...	...	...	...	...	...	...
2023-06-20	293.040009	293.040009	297.179993	292.679993	293.649994	2729800
2023-06-21	294.519989	294.519989	295.160004	292.809998	293.320007	2837100
2023-06-22	293.299988	293.299988	295.079987	291.519989	294.619995	1799400
2023-06-23	289.910004	289.910004	292.450012	289.640015	291.350006	3719700
2023-06-26	287.200012	287.200012	289.750000	287.070007	289.640015	402173

2638 rows × 6 columns

In [27]:
```
(data.iloc[: , ::2]
  .pipe(column_outer)
)
```

Out[27]:

Date	Adj Close	Close	High	Low	Open	Volume
2013-01-02	67.764641	90.120003	90.349998	89.330002	89.400002	7377200
2013-01-03	68.148117	90.629997	90.699997	90.160004	90.309998	5473500
2013-01-04	67.561623	89.849998	90.790001	89.669998	90.620003	5374100
2013-01-07	68.358658	90.910004	91.050003	89.250000	89.769997	5820900
2013-01-08	68.381241	90.940002	90.959999	89.970001	90.540001	6285200
...	...	...	...	...	...	...
2023-06-20	293.040009	293.040009	297.179993	292.679993	293.649994	2729800
2023-06-21	294.519989	294.519989	295.160004	292.809998	293.320007	2837100
2023-06-22	293.299988	293.299988	295.079987	291.519989	294.619995	1799400
2023-06-23	289.910004	289.910004	292.450012	289.640015	291.350006	3719700
2023-06-26	287.200012	287.200012	289.750000	287.070007	289.640015	402173

2638 rows × 6 columns

In [28]:
```
mc.isna().sum()
```

Out[28]:
```
Adj Close    0
Close        0
High         0
Low          0
Open         0
Volume       0
dtype: int64
```

In [29]:
```
(mc[['Close']])
```

Out[29]:

Date	Close
2013-01-02	90.120003
2013-01-03	90.629997
2013-01-04	89.849998
2013-01-07	90.910004
2013-01-08	90.940002
...	...
2023-06-20	293.040009
2023-06-21	294.519989
2023-06-22	293.299988
2023-06-23	289.910004
2023-06-26	287.200012

2638 rows × 1 columns

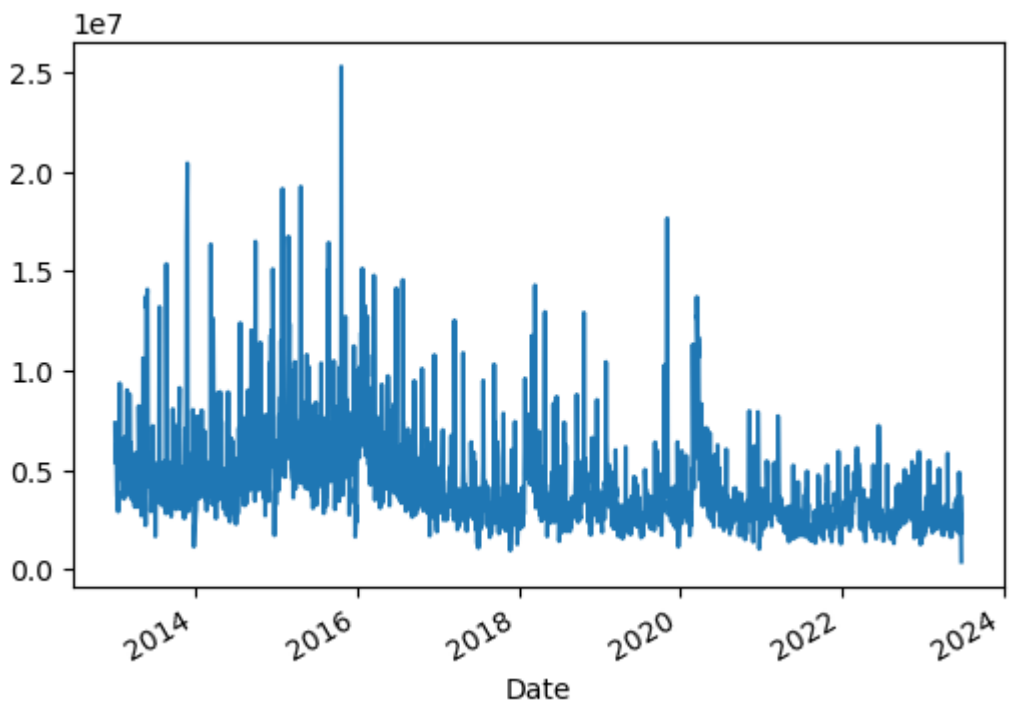
- Line plot of Closing stocks of Mc.Donald's using panda.

In [30]:
```
(mc['Close']
 .plot(figsize=(6,4)))
plt.title('Closing stock of McDonald\'s')
plt.show()
```


Giving 'Close' value of McDonald on daily basis from 2013 to current date.

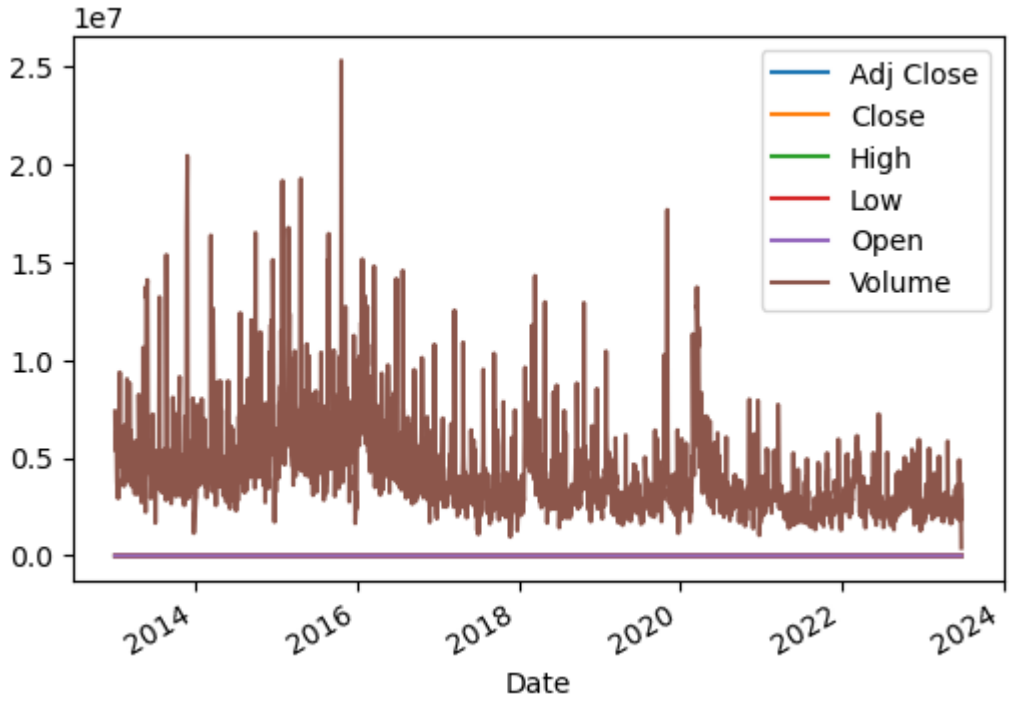
- Line plot of Volume stocks of Mc.Donald's using panda.

In [31]:
```
(mc
 ['Volume']
 .plot(figsize=(6,4)))
plt.show()
```



Line plot of all the stock value of Mc. donalds using panda.

In [32]:
```
(mc.plot(figsize=(6,4)))
plt.show()
```



Step 9: Resampling

1. resampling on basis of Month : resample('M')

In [33]:
```
(mc
 .resample('M')
#  ['Open'].mean()
)
```

Out[33]: <pandas.core.resample.DatetimeIndexResampler object at 0x0000020D6F1ECDC0>

In [34]:
```
(mc
 .resample('M')
 [['Open']]
# .mean()
)
```

Out[34]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000020D6F285330>

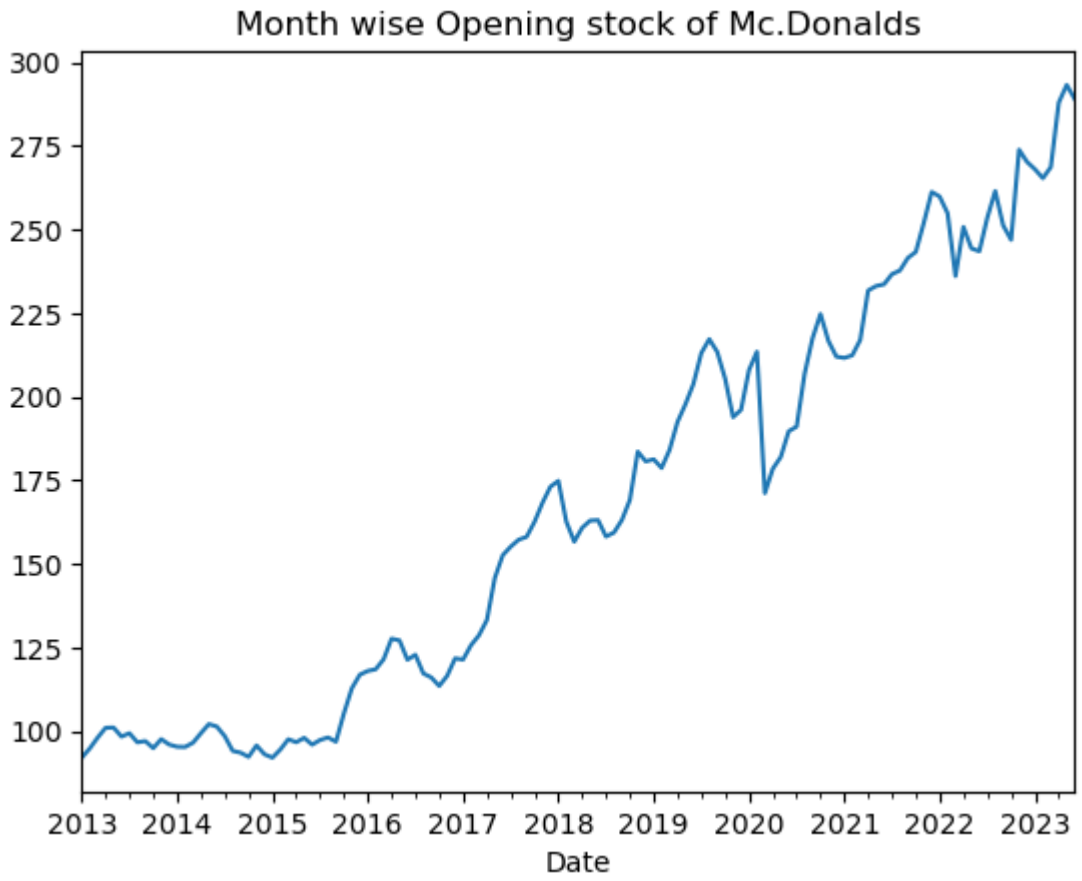
In [35]:
```
(mc
 .resample('M')
 [['Open']]
 .mean()
)
```

Out[35]:

Date	Open
2013-01-31	92.014762
2013-02-28	94.745789
2013-03-31	98.007499
2013-04-30	100.939545
2013-05-31	101.026363
...	...
2023-02-28	265.340001
2023-03-31	268.677826
2023-04-30	287.956315
2023-05-31	293.237726
2023-06-30	289.054120

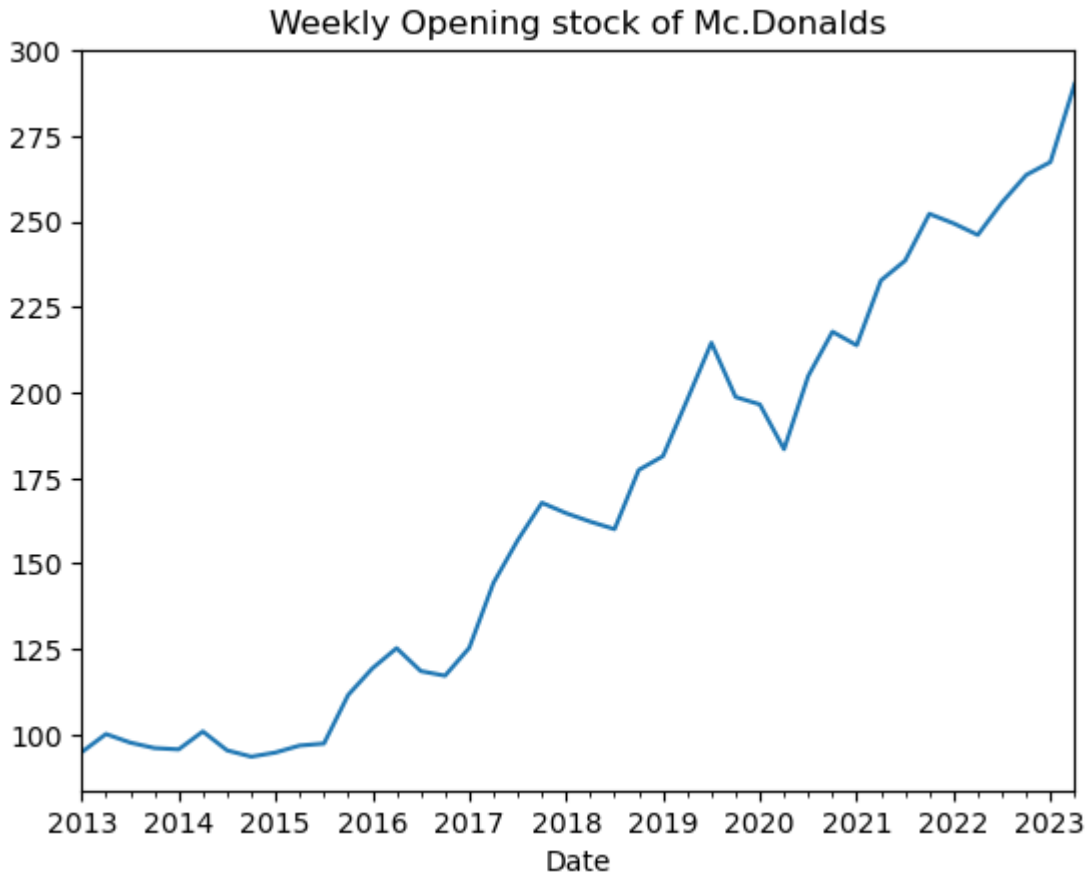
126 rows × 1 columns

In [36]:
```
(mc
 .resample('M')
 ['Open']
 .mean()
 .plot()
)
plt.title('Month wise Opening stock of Mc.Donalds')
plt.show()
```



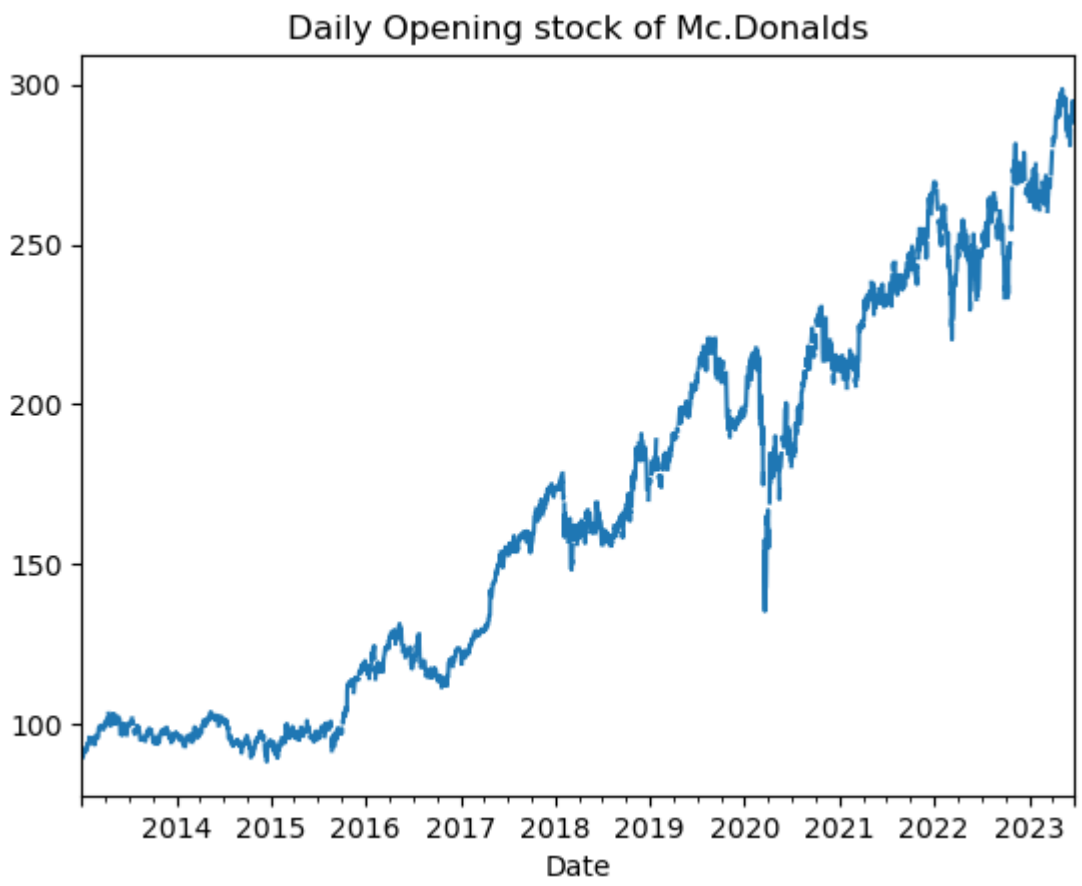
2. resampling on basis of Quaterly : resample('Q')

In [37]:
```
(mc
 .resample('Q')
 ['Open']
 .mean()
 .plot()
)
plt.title('Weekly Opening stock of Mc.Donalds')
plt.show()
```



3. resampling on basis of daily : resample('d')

In [38]:
```
(mc
 .resample('d')
 ['Open']
 .mean()
 .plot()
)
plt.title('Daily Opening stock of Mc.Donalds')
plt.show()
```

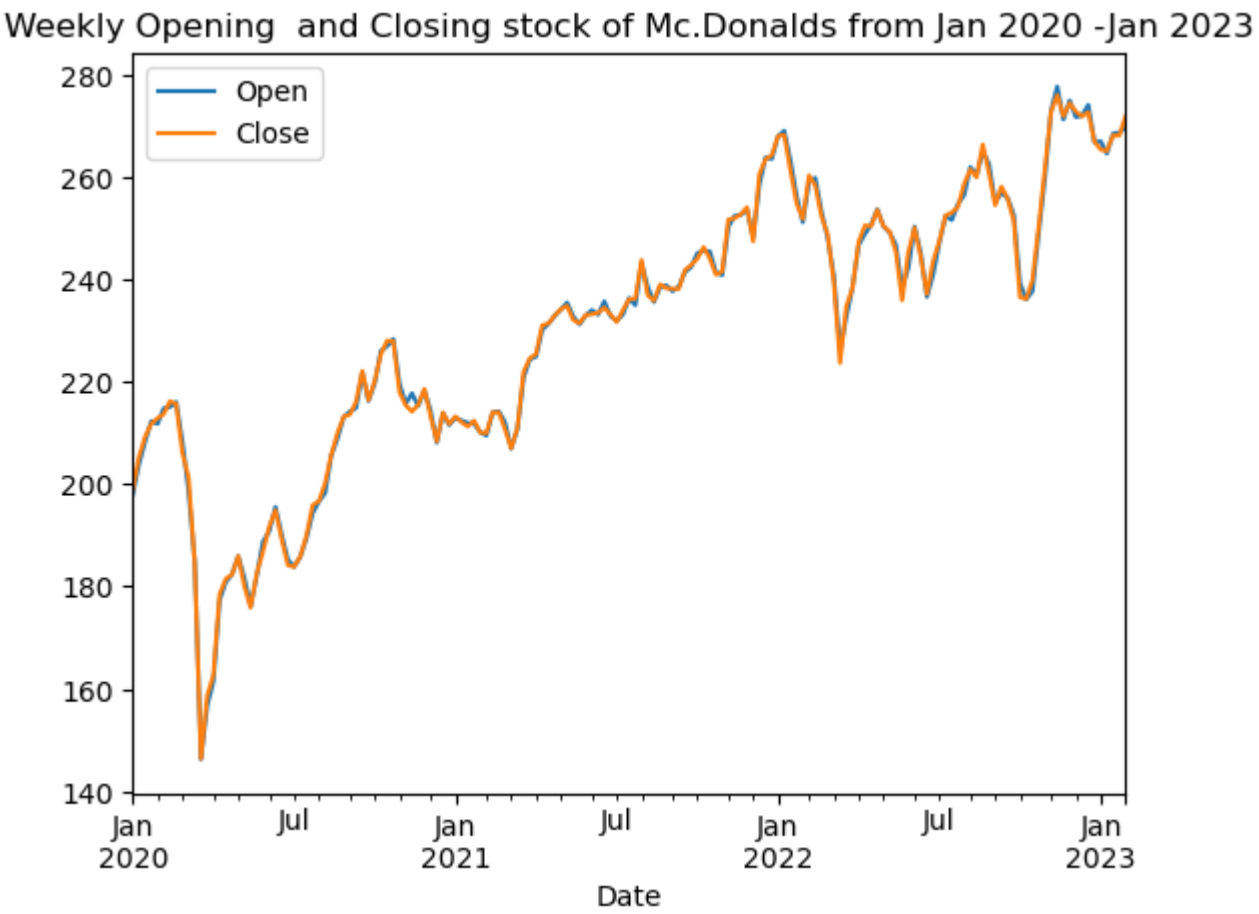


4. resampling on basis of weekly : resample('w')

In [39]:
```
(mc
 .resample('w')
 ['Close']
 .mean()
 #.plot()
)
```

Out[39]:
```
Date
2013-01-06     90.199999
2013-01-13     91.158002
2013-01-20     91.632001
2013-01-27     93.365000
2013-02-03     95.037999
                  ...
2023-06-04    286.952499
2023-06-11    285.488000
2023-06-18    290.373999
2023-06-25    292.692497
2023-07-02    287.200012
Freq: W-SUN, Name: Close, Length: 548, dtype: float64
```

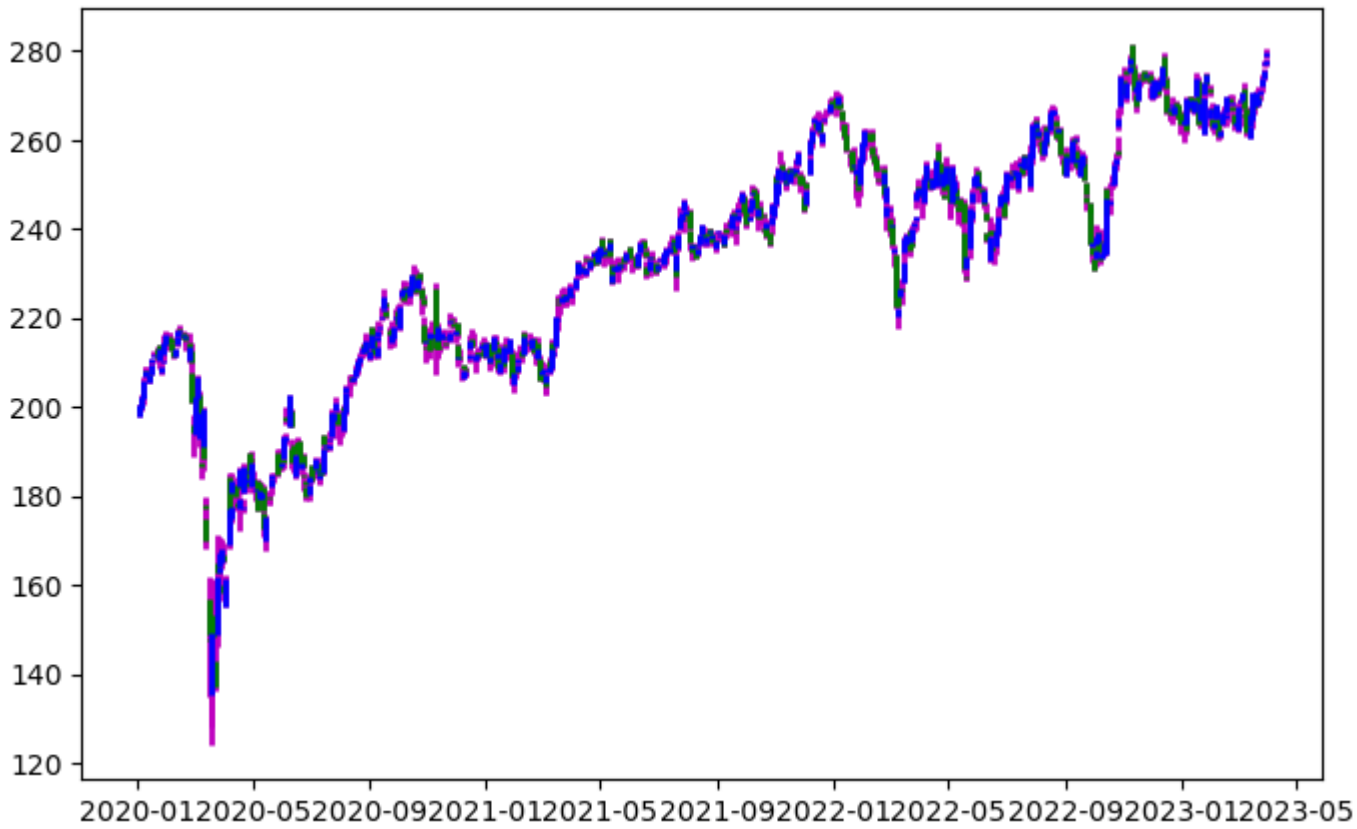
In [40]:
```
(mc
  .resample('w')
 ['Open','Close']
 .mean()
 .loc['jan 2020':'jan 2023']
 .plot()
)
plt.title('Weekly Opening  and Closing stock of Mc.Donalds from Jan 2020 -Jan 2023')
plt.show()
```



Candlestick Plot

In [41]:
```
fig, ax = plt.subplots (figsize = (8,5))
def candle_plot(data, ax):
    # wick
    ax.vlines(x = data.index, ymin = data.Low, ymax = data.High, colors='m', linewidth=2 )
    # green : decrease
    green = data.query('Open > Close')
    ax.vlines(x = green.index, ymin = green.Close, ymax = green.Open, colors='g', linewidth=2 )
    # blue : increase
    blue = data.query('Open <= Close')
    ax.vlines(x = blue.index, ymin = blue.Close, ymax = blue.Open, colors='b', linewidth=2 )
    return data


(mc
 .resample('d')
 .agg({'Open' : "first", 'High' : "max", 'Low':"min", 'Close' : "last"})
 .loc['jan 2020' : 'march 2023']
 .pipe(candle_plot, ax)
)
plt.title('Candlestick plot for stocks of Mc.Donalds')
```

Out[41]: Text(0.5, 1.0, 'Candlestick plot for stocks of Mc.Donalds')

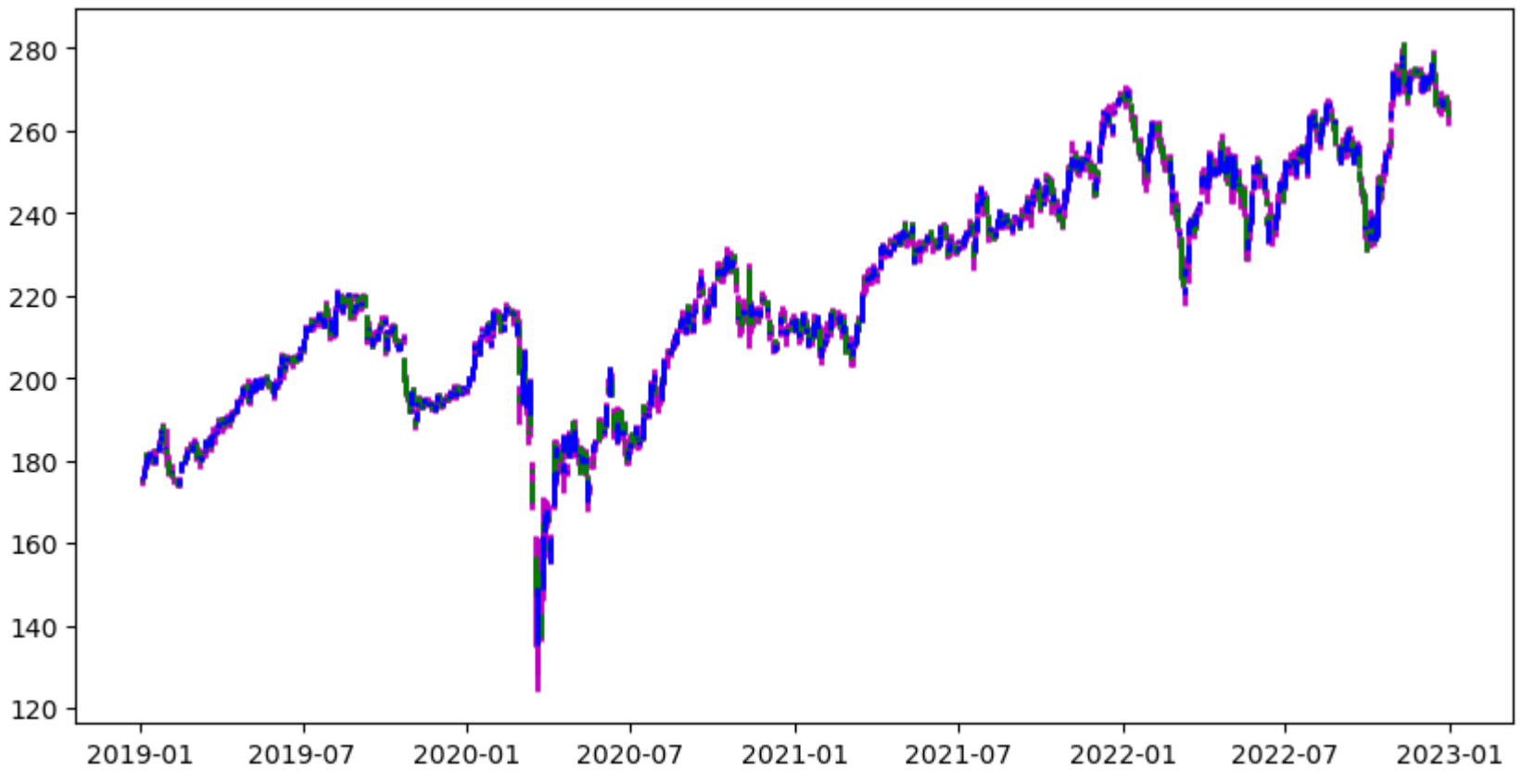


In [42]:
```
fig, ax = plt.subplots (figsize = (10,5))
(mc
 .resample('d')
 .agg({'Open' : "first", 'High' : "max", 'Low':"min", 'Close' : "last"})
 .loc['jan 2019' : 'dec 2022']
 .pipe(candle_plot, ax)
)
```

Out[42]:

Date	Open	High	Low	Close
2019-01-01	NaN	NaN	NaN	NaN
2019-01-02	175.410004	176.300003	174.169998	176.059998
2019-01-03	175.449997	176.449997	174.410004	174.899994
2019-01-04	176.029999	179.199997	175.690002	178.279999
2019-01-05	NaN	NaN	NaN	NaN
...	...	...	...	...
2022-12-27	268.660004	268.869995	266.600006	266.839996
2022-12-28	268.000000	268.140015	265.070007	265.109985
2022-12-29	265.940002	267.809998	264.880005	265.929993
2022-12-30	265.200012	265.380005	261.399994	263.529999
2022-12-31	NaN	NaN	NaN	NaN

1461 rows × 4 columns



Step 10: Returns

In [43]:
```
# Panda has a nice method to calculate returns. data.pct_change()
```

In [44]:
```
# mc.pct_change?
```

In [45]:
```
mc.pct_change()
```

Out[45]:

Date	Adj Close	Close	High	Low	Open	Volume
2013-01-02	NaN	NaN	NaN	NaN	NaN	NaN
2013-01-03	0.005659	0.005659	0.003874	0.009291	0.010179	-0.258052
2013-01-04	-0.008606	-0.008606	0.000992	-0.005435	0.003433	-0.018160
2013-01-07	0.011797	0.011797	0.002864	-0.004684	-0.009380	0.083140
2013-01-08	0.000330	0.000330	-0.000989	0.008067	0.008578	0.079764
...	...	...	...	...	...	...
2023-06-20	-0.002247	-0.002247	0.002057	-0.001569	-0.002920	-0.441804
2023-06-21	0.005050	0.005050	-0.006797	0.000444	-0.001124	0.039307
2023-06-22	-0.004142	-0.004142	-0.000271	-0.004406	0.004432	-0.365761
2023-06-23	-0.011558	-0.011558	-0.008913	-0.006449	-0.011099	1.067189
2023-06-26	-0.009348	-0.009348	-0.009232	-0.008873	-0.005869	-0.891880

2638 rows × 6 columns

In [46]:
```
mc.pct_change(periods = 2)
```

Out[46]:

Date	Adj Close	Close	High	Low	Open	Volume
2013-01-02	NaN	NaN	NaN	NaN	NaN	NaN
2013-01-03	NaN	NaN	NaN	NaN	NaN	NaN
2013-01-04	-0.002996	-0.002996	0.004870	0.003806	0.013647	-0.271526
2013-01-07	0.003089	0.003090	0.003859	-0.010093	-0.005979	0.063469
2013-01-08	0.012131	0.012131	0.001872	0.003346	-0.000883	0.169535
...	...	...	...	...	...	...
2023-06-20	0.001470	0.001470	0.012607	0.015333	0.013810	0.042227
2023-06-21	0.002792	0.002792	-0.004754	-0.001126	-0.004041	-0.419863
2023-06-22	0.000887	0.000887	-0.007066	-0.003963	0.003303	-0.340831
2023-06-23	-0.015653	-0.015653	-0.009181	-0.010826	-0.006716	0.311092
2023-06-26	-0.020798	-0.020798	-0.018063	-0.015265	-0.016903	-0.776496

2638 rows × 6 columns

In [47]:
```
mc.pct_change(periods = 3)
```

Out[47]:

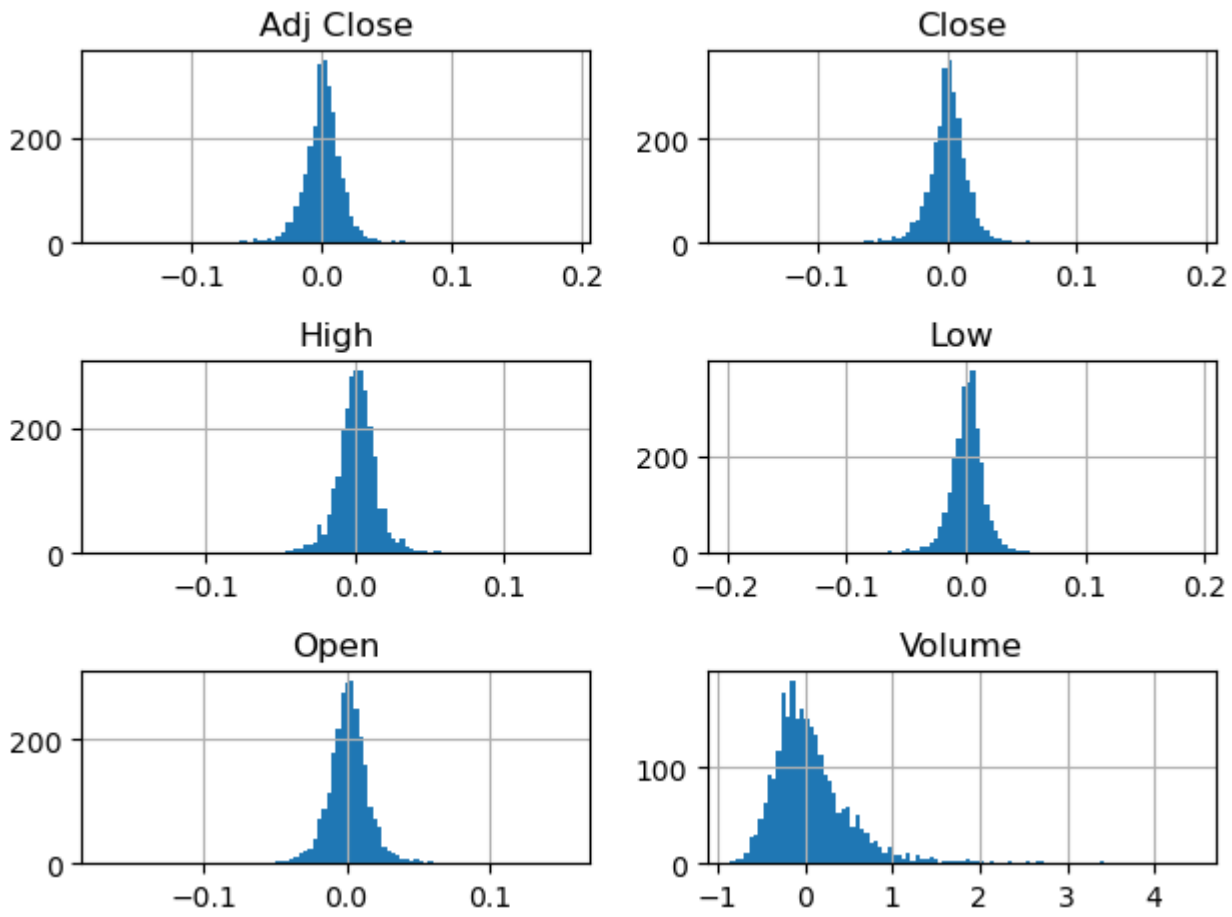
Date	Adj Close	Close	High	Low	Open	Volume
2013-01-02	NaN	NaN	NaN	NaN	NaN	NaN
2013-01-03	NaN	NaN	NaN	NaN	NaN	NaN
2013-01-04	NaN	NaN	NaN	NaN	NaN	NaN
2013-01-07	0.008766	0.008766	0.007748	-0.000896	0.004139	-0.210961
2013-01-08	0.003421	0.003421	0.002867	-0.002107	0.002547	0.148296
...	...	...	...	...	...	...
2023-06-20	0.015948	0.015948	0.026316	0.018691	0.019264	0.389565
2023-06-21	0.006527	0.006527	0.005724	0.015784	0.012671	0.083193
2023-06-22	-0.001362	-0.001362	-0.005024	-0.005526	0.000373	-0.632055
2023-06-23	-0.010681	-0.010681	-0.015916	-0.010387	-0.007832	0.362627
2023-06-26	-0.024854	-0.024854	-0.018329	-0.019603	-0.012546	-0.858245

2638 rows × 6 columns

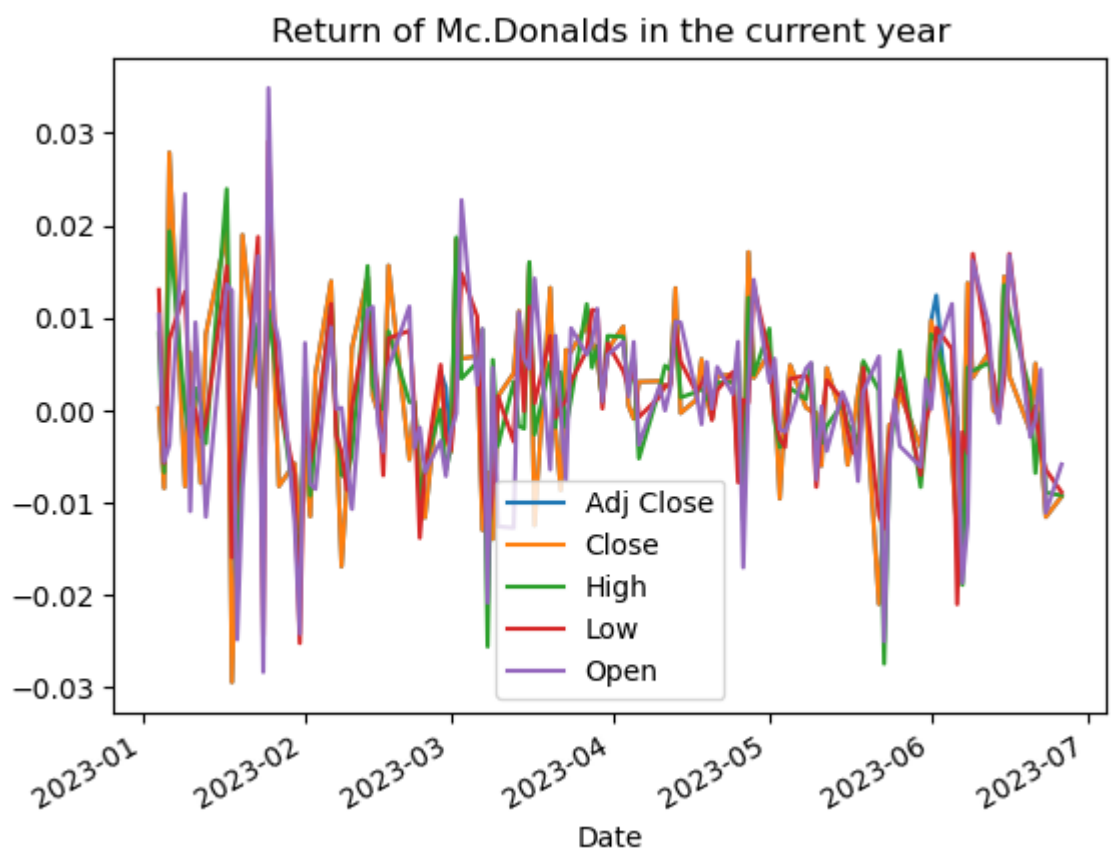
Plotting return (Histogram using panda)

- Are the returns positive?
- Are the returns negative?
- A histogram allows us to see that.
- Creating a histogram with pandas. (deafult bin value is 10)

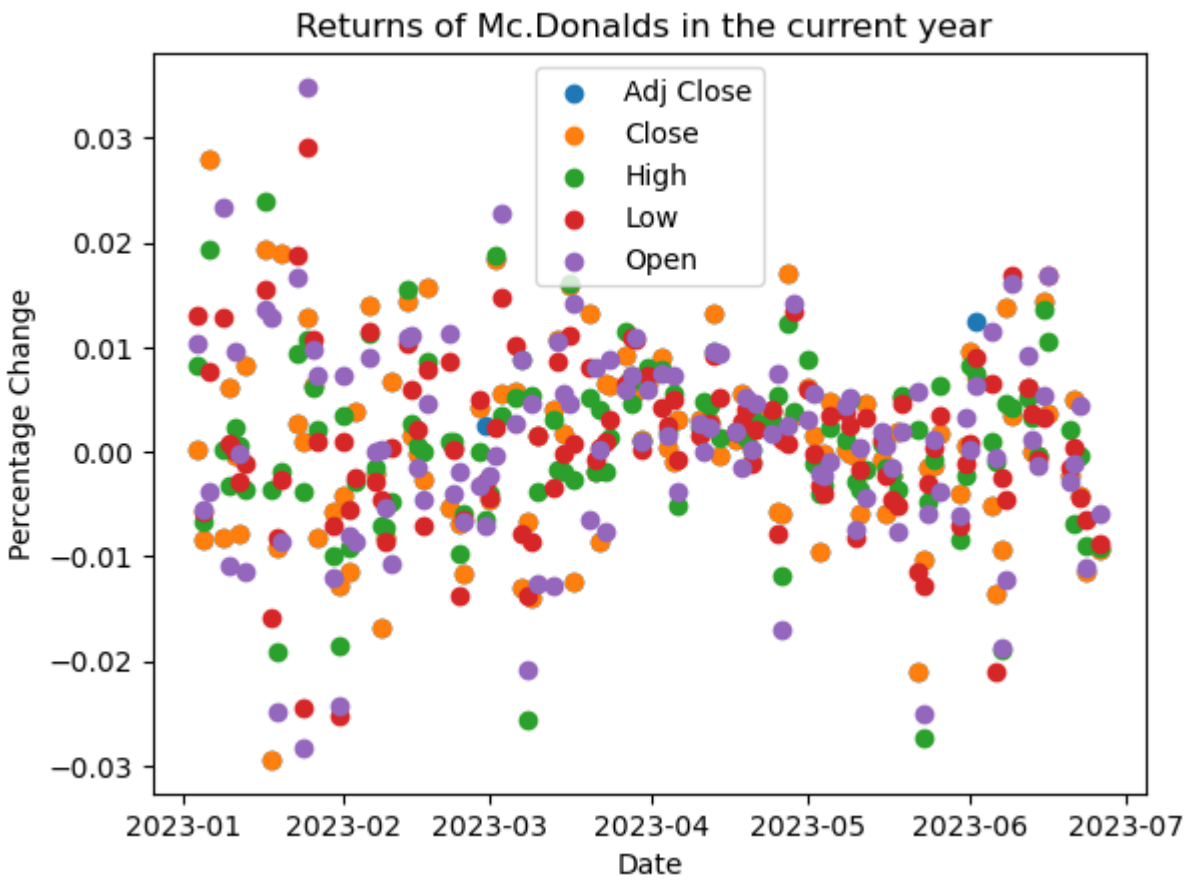
In [48]:
```
(mc
.pct_change(periods = 2)
.hist(bins=100)
)
plt.tight_layout()
plt.show()
```



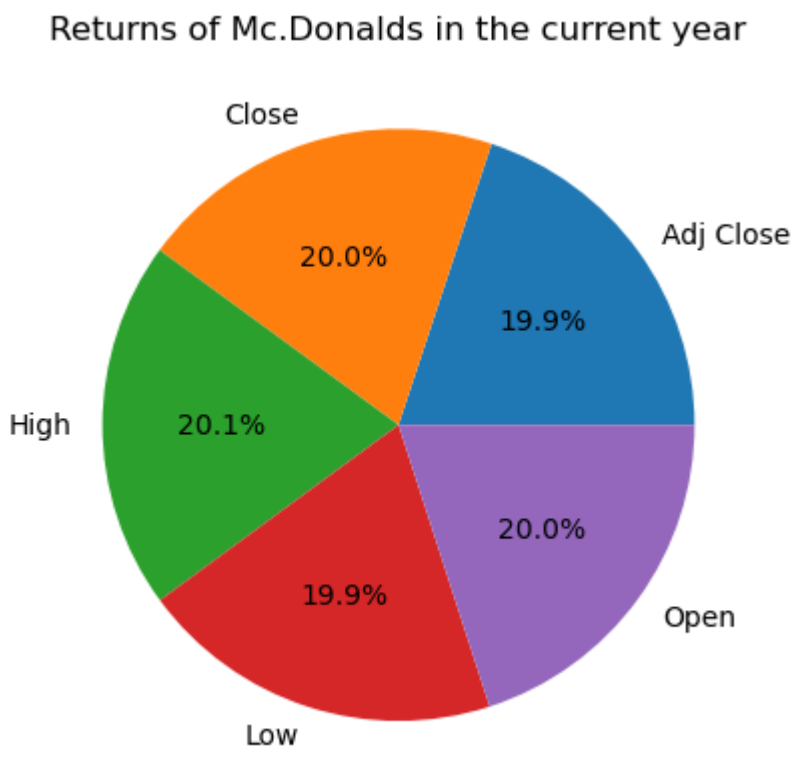
In [49]:
```
(mc.iloc[:,:-1].loc['Jan 2023':'Jun 2023']
.pct_change()
.plot()
)

plt.title('Return of Mc.Donalds in the current year')
plt.show()
```



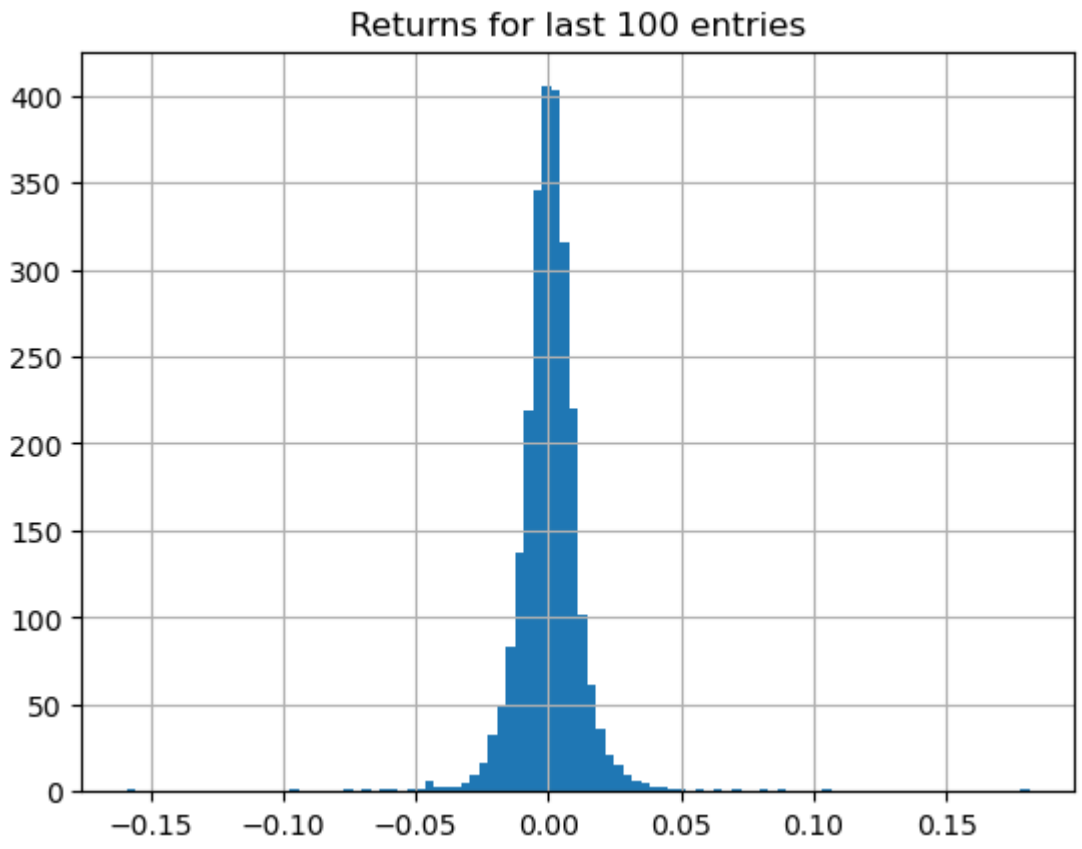
In [50]:
```
data = mc.iloc[:, :-1].loc['Jan 2023':'Jun 2023'].pct_change()

for column in data.columns:
    plt.scatter(data.index, data[column], label=column)


plt.xlabel('Date')
plt.ylabel('Percentage Change')
plt.title('Returns of Mc.Donalds in the current year ')
plt.legend()

# Display the plot
plt.show()
```



In [51]:
```
data = mc.iloc[:, :-1].loc['Jan 2023':'Jun 2023'].sum()
plt.pie(data, labels=data.index, autopct='%1.1f%%')
plt.title('Returns of Mc.Donalds in the current year')

# Display the chart
plt.show()
```

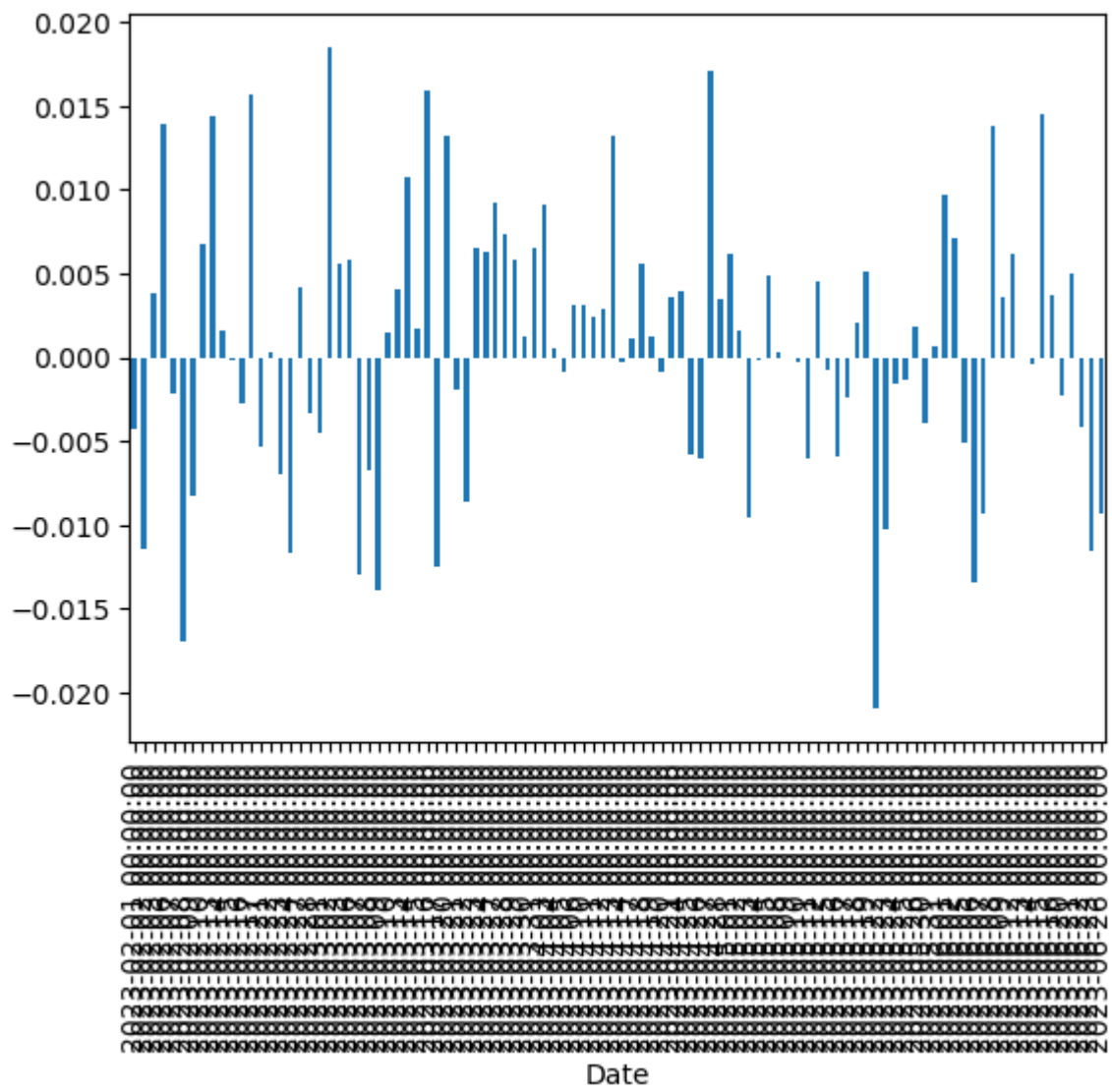


In [52]:
```
# For last 100 entries of  Closing stock:
(mc
.pct_change()
.Close
.iloc[100:]
.hist(bins=100)
)
plt.title('Returns for last 100 entries')
plt.show()
```



In [53]:
```
(mc
.pct_change()
.Close
.iloc[-100:]
.plot.bar()
)
```

Out[53]: <Axes: xlabel='Date'>

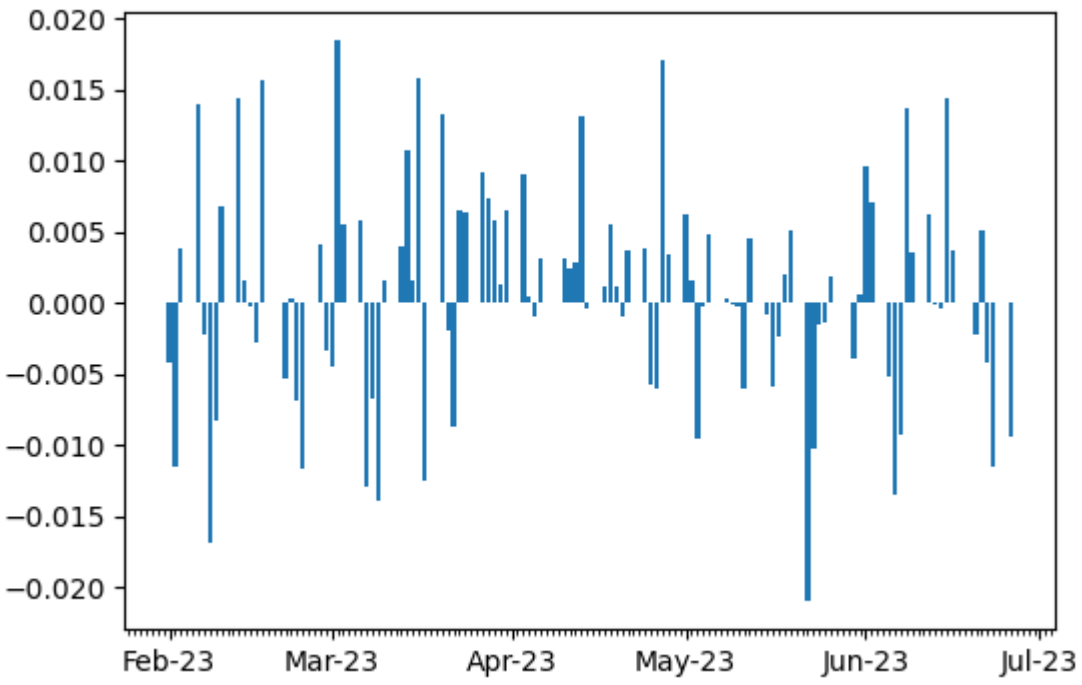


- There is an issue of overlapping in x-axis.
- Bar plot using panda converts whatever's in the X-axis to categoricals. (taking allthose dates and plotting each and every date there)
- To fix this, matplotlib is used to plot a bar plot rather than pandas.

Fix the barplot

In [54]:
```
# Function for taking care of index using matplotlib
def bar_index(col,ax):
    ax.bar(col.index, col)
    ax.xaxis.set_major_locator(dates.MonthLocator())
    ax.xaxis.set_major_formatter(dates.DateFormatter('%b-%y'))
    ax.xaxis.set_minor_locator(dates.DayLocator())
    return col
```

In [55]:
```
fig, ax = plt.subplots (figsize = (6,4))
_=(mc
.pct_change()
.Close
.iloc[-100:]
.pipe(bar_index, ax)
)
```



Step 11: Cumulative Return

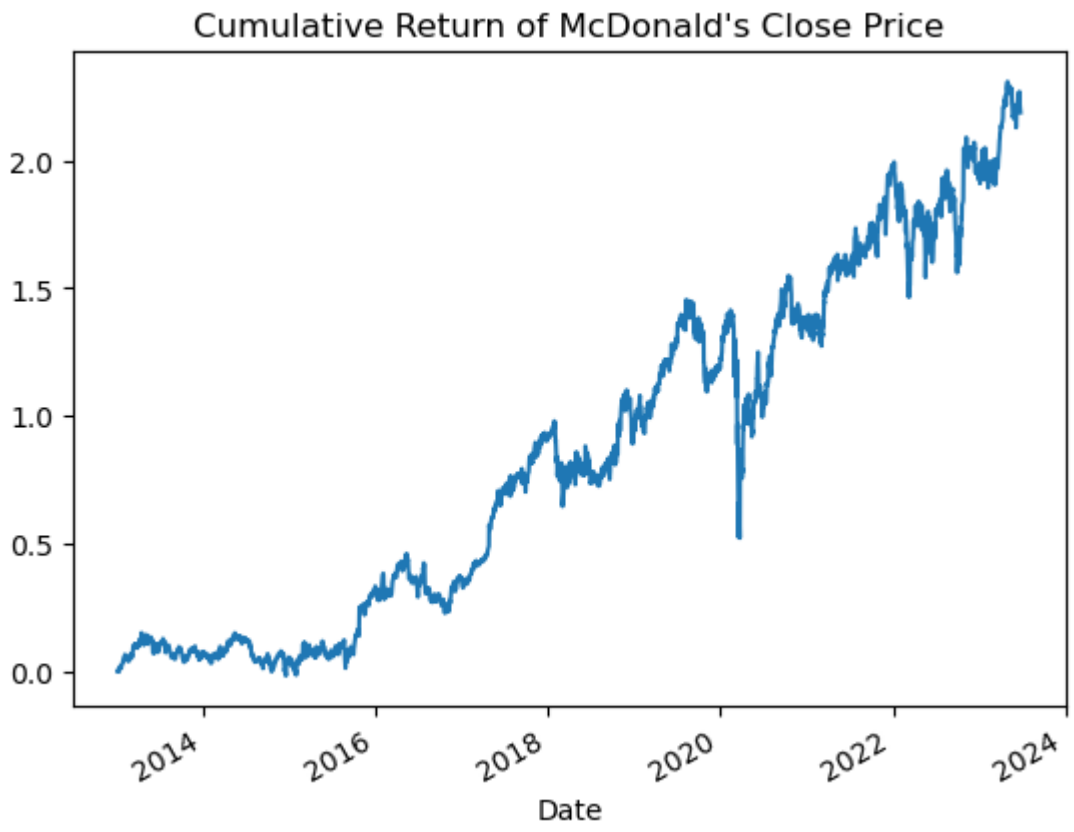
Goal:

- More complicated pandas
- Refactoring into function
- Explore source
- Creating new columns with .assign
- illistrate lambda

cumulative Return is the amount that investment has gained or lost over time:

$$x = \frac{\text{Current price} - \text{Original price}}{\text{Original price}}$$

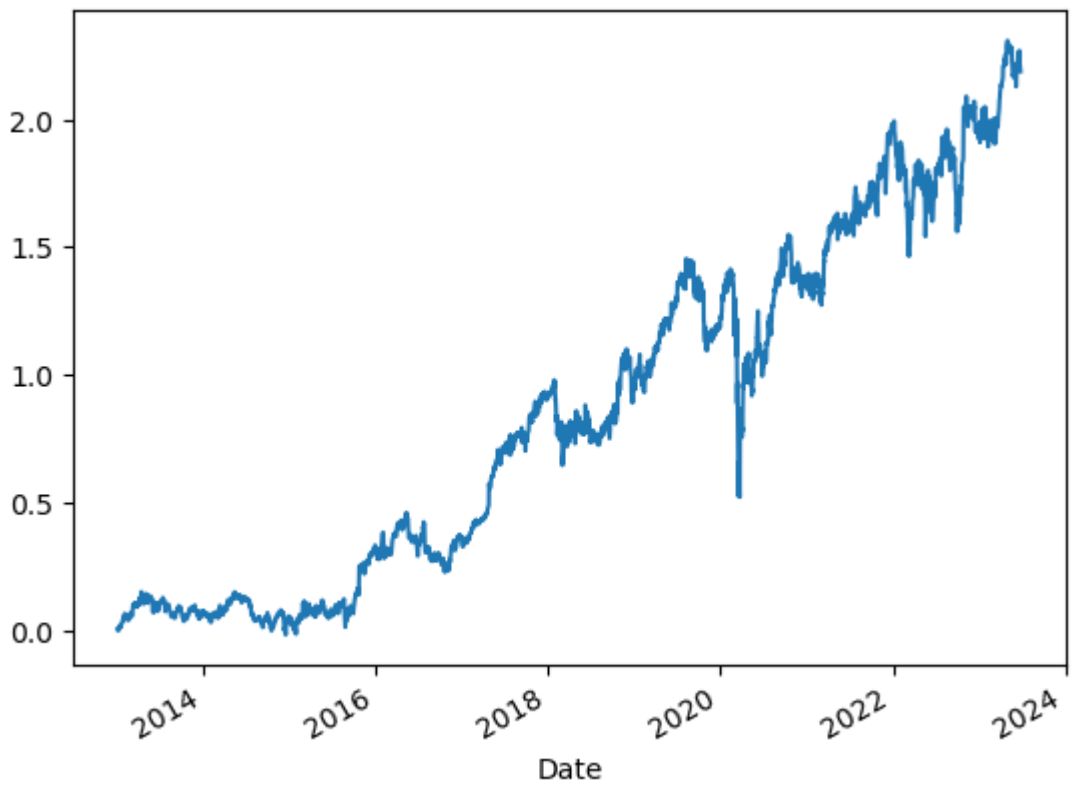
In [56]:
```
# 1st value in the column is the original price.
(mc
 .Close
 .sub(mc.Close[0])   # Substracting 1st value from the Close column
 .div(mc.Close[0])   # Diving 1st value from the Close column
 .plot())
plt.title('Cumulative Return of McDonald\'s Close Price')
plt.show()
```



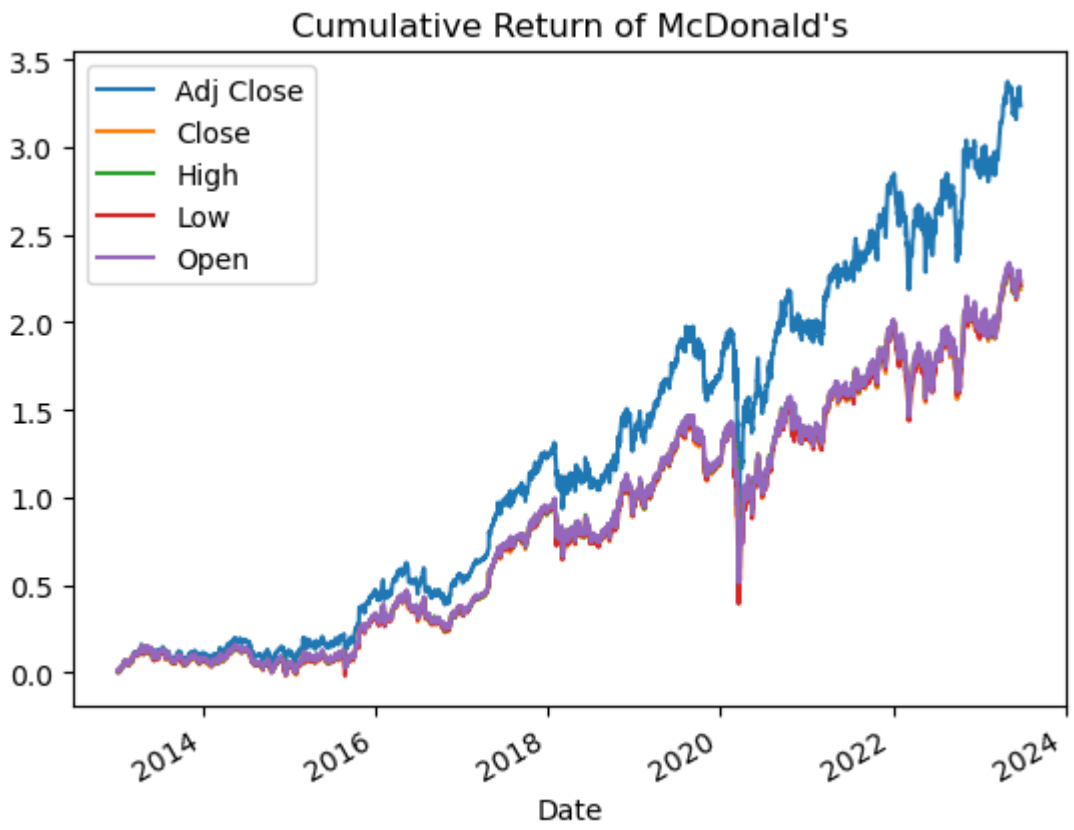
- An alternate method:

In [57]:
```
(mc.Close
 .pct_change()
 .add(1)
 .cumprod()
 .sub(1)
 .plot())
```

Out[57]: <Axes: xlabel='Date'>



In [58]:
```
(mc.iloc[:,:-1]
 .pct_change()
 .add(1)
 .cumprod()
 .sub(1)
 .plot())
plt.title('Cumulative Return of McDonald\'s')
plt.show()
```



In [59]:
```
# mc.cumprod??
```

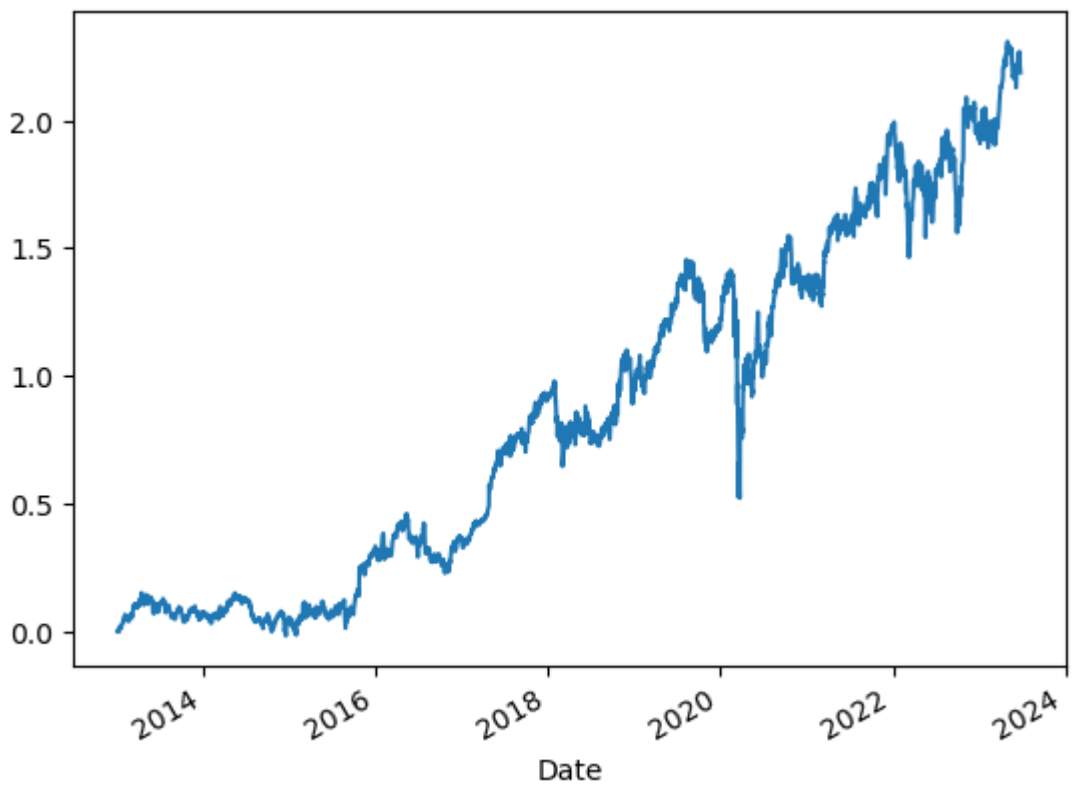
In [60]:
```
# pd.core.generic.NDFrame.cumprod??
```

In [61]:
```
# np.cumprod??
```

Create a function to calculate cumulative return

In [62]:
```
def cum_returns( data, col):
    series = data[col]

    ''' series[0] is original price which is substracted
    from the other values in the series and then the
    result get divided by the original data (series[0])'''

    return (series
            .sub(series[0])
            .div(series[0])
           )
(mc
    .pipe(cum_returns, 'Close')
    .plot()
)
```

Out[62]: <Axes: xlabel='Date'>



In [63]:
```
# Using anonymous function lambda
def get_returns(data,col):
    return cum_returns(data, col)

(lambda data : get_returns(data, 'Close'))(mc)
```

Out[63]:
```
Date
2013-01-02    0.000000
2013-01-03    0.005659
2013-01-04   -0.002996
2013-01-07    0.008766
2013-01-08    0.009099
                ...
2023-06-20    2.251664
2023-06-21    2.268087
2023-06-22    2.254549
2023-06-23    2.216933
2023-06-26    2.186862
Name: Close, Length: 2638, dtype: float64
```

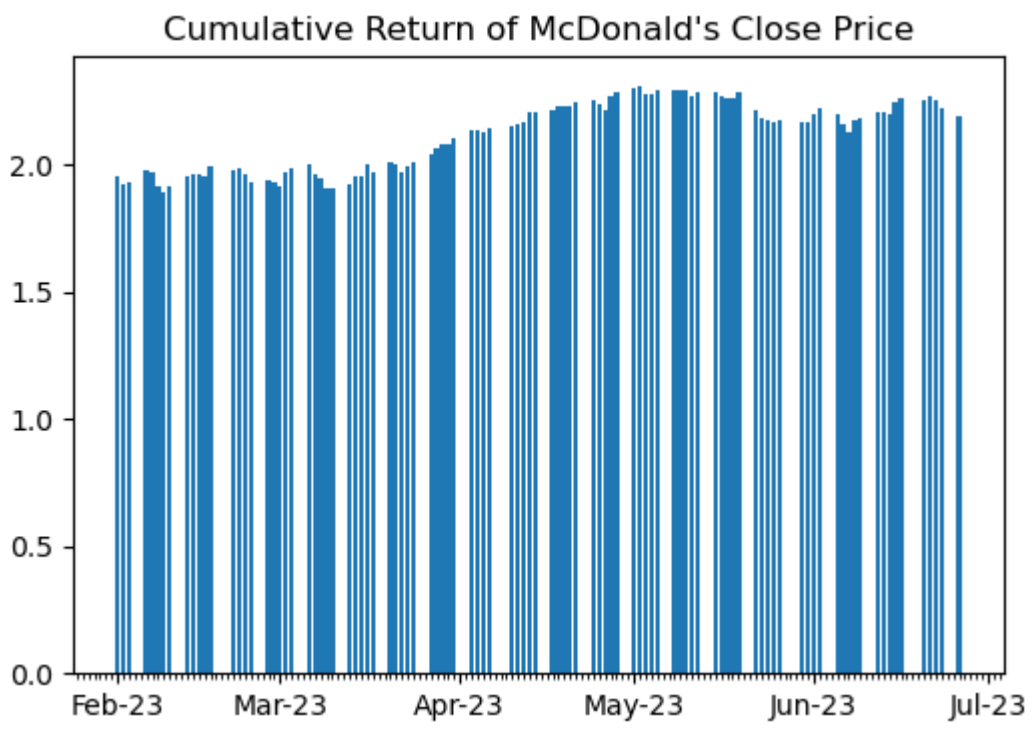
In [64]:
```
# creating a new column cum_return
(mc
.assign(cum_returns = lambda data: cum_returns(data, 'Close'))
)
```

Out[64]:

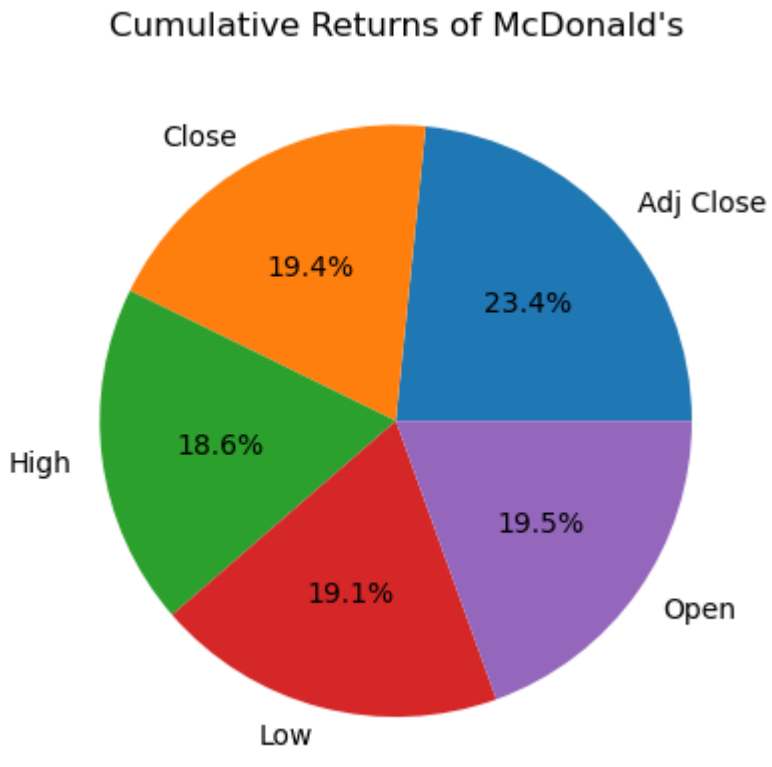
Date	Adj Close	Close	High	Low	Open	Volume	cum_returns
2013-01-02	67.764641	90.120003	90.349998	89.330002	89.400002	7377200	0.000000
2013-01-03	68.148117	90.629997	90.699997	90.160004	90.309998	5473500	0.005659
2013-01-04	67.561623	89.849998	90.790001	89.669998	90.620003	5374100	-0.002996
2013-01-07	68.358658	90.910004	91.050003	89.250000	89.769997	5820900	0.008766
2013-01-08	68.381241	90.940002	90.959999	89.970001	90.540001	6285200	0.009099
...	...	...	...	...	...	...	...
2023-06-20	293.040009	293.040009	297.179993	292.679993	293.649994	2729800	2.251664
2023-06-21	294.519989	294.519989	295.160004	292.809998	293.320007	2837100	2.268087
2023-06-22	293.299988	293.299988	295.079987	291.519989	294.619995	1799400	2.254549
2023-06-23	289.910004	289.910004	292.450012	289.640015	291.350006	3719700	2.216933
2023-06-26	287.200012	287.200012	289.750000	287.070007	289.640015	402173	2.186862

2638 rows × 7 columns

In [65]:
```
# Returns using matplotlib
fig, ax = plt.subplots (figsize = (6,4))

_=(mc
.pipe(cum_returns, 'Close')
.iloc[-100:]
.pipe(bar_index, ax)
)
plt.title('Cumulative Return of McDonald\'s Close Price')
plt.show()
```



In [66]:
```
percentage_change = mc.iloc[:, :-1].pct_change().mean()
plt.pie(percentage_change, labels=percentage_change.index, autopct='%1.1f%%')
plt.title('Cumulative Returns of McDonald\'s')
plt.show()
```



Step 11: Rolling Windows

In [67]:
```
(mc
.Close
.mean()
)
```

Out[67]: 166.9577217998606

In [68]:
```
(mc
.Close
.std()
)
```

Out[68]: 60.562615779627194

In [69]:
```
(mc
.assign(pct_change_close = mc.Close.pct_change())
# .pct_change_close
# .std()
)
```

Out[69]:

Date	Adj Close	Close	High	Low	Open	Volume	pct_change_close
2013-01-02	67.764641	90.120003	90.349998	89.330002	89.400002	7377200	NaN
2013-01-03	68.148117	90.629997	90.699997	90.160004	90.309998	5473500	0.005659
2013-01-04	67.561623	89.849998	90.790001	89.669998	90.620003	5374100	-0.008606
2013-01-07	68.358658	90.910004	91.050003	89.250000	89.769997	5820900	0.011797
2013-01-08	68.381241	90.940002	90.959999	89.970001	90.540001	6285200	0.000330
...	...	...	...	...	...	...	...
2023-06-20	293.040009	293.040009	297.179993	292.679993	293.649994	2729800	-0.002247
2023-06-21	294.519989	294.519989	295.160004	292.809998	293.320007	2837100	0.005050
2023-06-22	293.299988	293.299988	295.079987	291.519989	294.619995	1799400	-0.004142
2023-06-23	289.910004	289.910004	292.450012	289.640015	291.350006	3719700	-0.011558
2023-06-26	287.200012	287.200012	289.750000	287.070007	289.640015	402173	-0.009348

2638 rows × 7 columns

In [70]:
```
(mc
.assign(pct_change_close = mc.Close.pct_change())
.pct_change_close
.std()
)
```

Out[70]: 0.0124202461156308

In [71]:
```
(mc
.assign(close_vol = mc.rolling(30).Close.std(),
 per_vol = mc.Close.pct_change().rolling(30).std())
#  .iloc[:, -2:]
#  .plot(subplots= True)
)
```

Out[71]:

Date	Adj Close	Close	High	Low	Open	Volume	close_vol	per_vol
2013-01-02	67.764641	90.120003	90.349998	89.330002	89.400002	7377200	NaN	NaN
2013-01-03	68.148117	90.629997	90.699997	90.160004	90.309998	5473500	NaN	NaN
2013-01-04	67.561623	89.849998	90.790001	89.669998	90.620003	5374100	NaN	NaN
2013-01-07	68.358658	90.910004	91.050003	89.250000	89.769997	5820900	NaN	NaN
2013-01-08	68.381241	90.940002	90.959999	89.970001	90.540001	6285200	NaN	NaN
...	...	...	...	...	...	...	...	...
2023-06-20	293.040009	293.040009	297.179993	292.679993	293.649994	2729800	4.491096	0.007429
2023-06-21	294.519989	294.519989	295.160004	292.809998	293.320007	2837100	4.400253	0.007494
2023-06-22	293.299988	293.299988	295.079987	291.519989	294.619995	1799400	4.270250	0.007528
2023-06-23	289.910004	289.910004	292.450012	289.640015	291.350006	3719700	4.089412	0.007801
2023-06-26	287.200012	287.200012	289.750000	287.070007	289.640015	402173	4.006403	0.007902

2638 rows × 8 columns

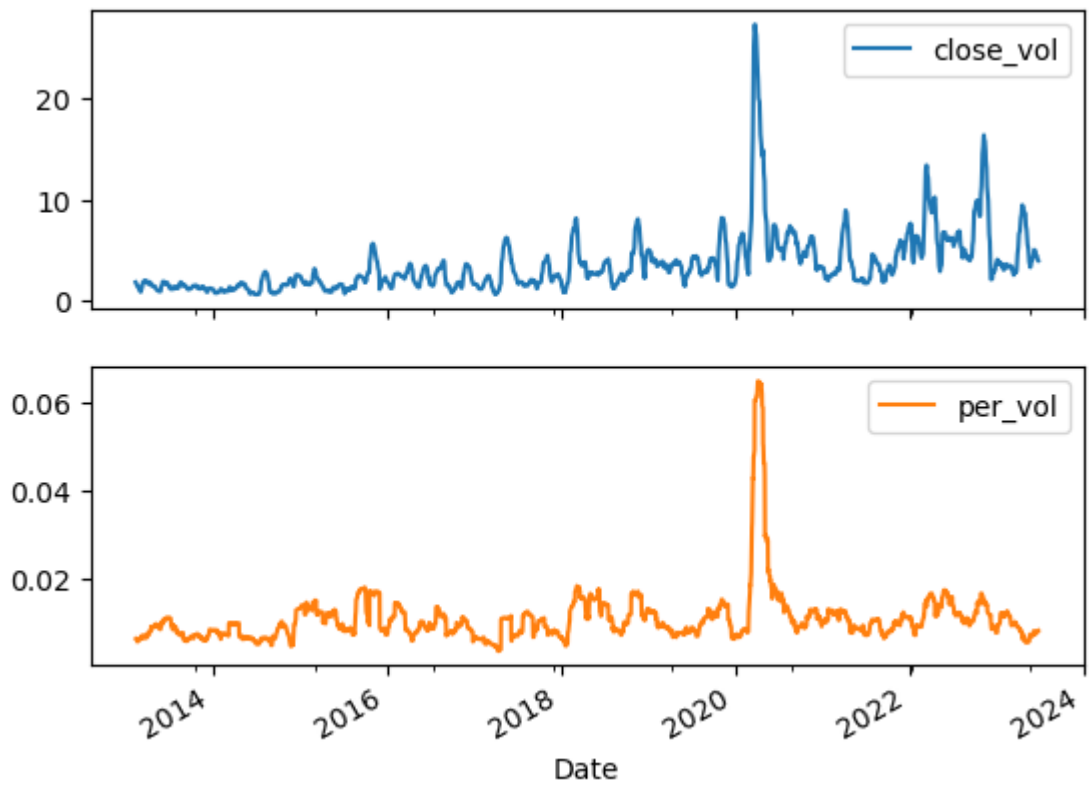
In [72]:
```
(mc
.assign(close_vol = mc.rolling(30).Close.std(),
 per_vol = mc.Close.pct_change().rolling(30).std())
 .iloc[:, -2:]
#  .plot(subplots= True)
)
```

Out[72]:

Date	close_vol	per_vol
2013-01-02	NaN	NaN
2013-01-03	NaN	NaN
2013-01-04	NaN	NaN
2013-01-07	NaN	NaN
2013-01-08	NaN	NaN
...	...	...
2023-06-20	4.491096	0.007429
2023-06-21	4.400253	0.007494
2023-06-22	4.270250	0.007528
2023-06-23	4.089412	0.007801
2023-06-26	4.006403	0.007902

2638 rows × 2 columns

In [73]:
```
(mc
.assign(close_vol = mc.rolling(30).Close.std(),
 per_vol = mc.Close.pct_change().rolling(30).std())
 .iloc[:, -2:]
 .plot(subplots= True)
)

plt.show()
```



In [74]:
```
#15 Days volatility
(mc
.assign(pct_change_close = mc.Close.pct_change())
 .resample('15D')
.std()
)
```

Out[74]:

Date	Adj Close	Close	High	Low	Open	Volume	pct_change_close
2013-01-02	0.445189	0.592066	0.551310	0.747213	0.780337	1.311617e+06	0.005921
2013-01-17	0.867751	1.154015	1.118608	1.151020	1.137432	1.531494e+06	0.003671
2013-02-01	0.509848	0.678046	0.676375	0.706809	0.561321	8.191800e+05	0.006692
2013-02-16	0.896198	0.970693	0.922171	0.962871	0.990779	1.703273e+06	0.005342
2013-03-03	1.234800	1.629022	1.507083	1.543329	1.518905	1.548244e+06	0.006548
...	...	...	...	...	...	...	...
2023-04-25	2.515142	2.528495	2.139819	2.574531	2.670171	1.169928e+06	0.007196
2023-05-10	3.861675	3.882167	3.562778	3.542131	3.414074	4.440435e+05	0.007454
2023-05-25	2.430560	2.257589	2.097191	2.401790	2.711986	5.059806e+05	0.008488
2023-06-09	2.770172	2.770172	3.512023	2.984027	3.194511	9.902811e+05	0.006945
2023-06-24	NaN	NaN	NaN	NaN	NaN	NaN	NaN

256 rows × 7 columns

In [75]:
```
#15 Days  rolling volatility
(mc
.assign(pct_change_close_15 = mc.Close.pct_change())
 .rolling (window = 15, min_periods = 15)
.std()
)
```

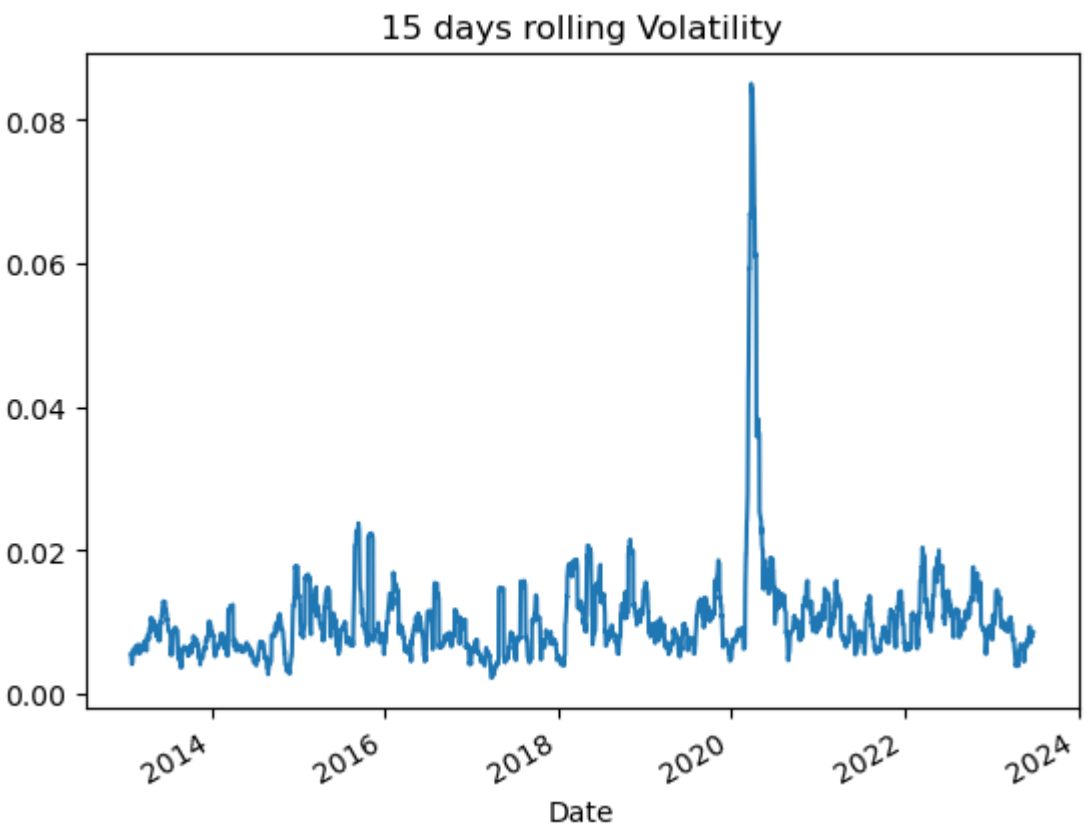
Out[75]:

Date	Adj Close	Close	High	Low	Open	Volume	pct_change_close_15
2013-01-02	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2013-01-03	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2013-01-04	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2013-01-07	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2013-01-08	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...
2023-06-20	3.597203	3.351179	3.714098	3.807213	3.663108	8.054466e+05	0.007936
2023-06-21	3.788148	3.624330	3.853685	4.144451	3.793592	7.874895e+05	0.007825
2023-06-22	3.729211	3.669457	3.830666	4.237650	4.009822	8.144862e+05	0.007991
2023-06-23	3.654549	3.654549	3.769603	4.174122	3.900513	8.605650e+05	0.008389
2023-06-26	3.690775	3.690775	3.779300	4.160306	3.834340	1.044950e+06	0.008540

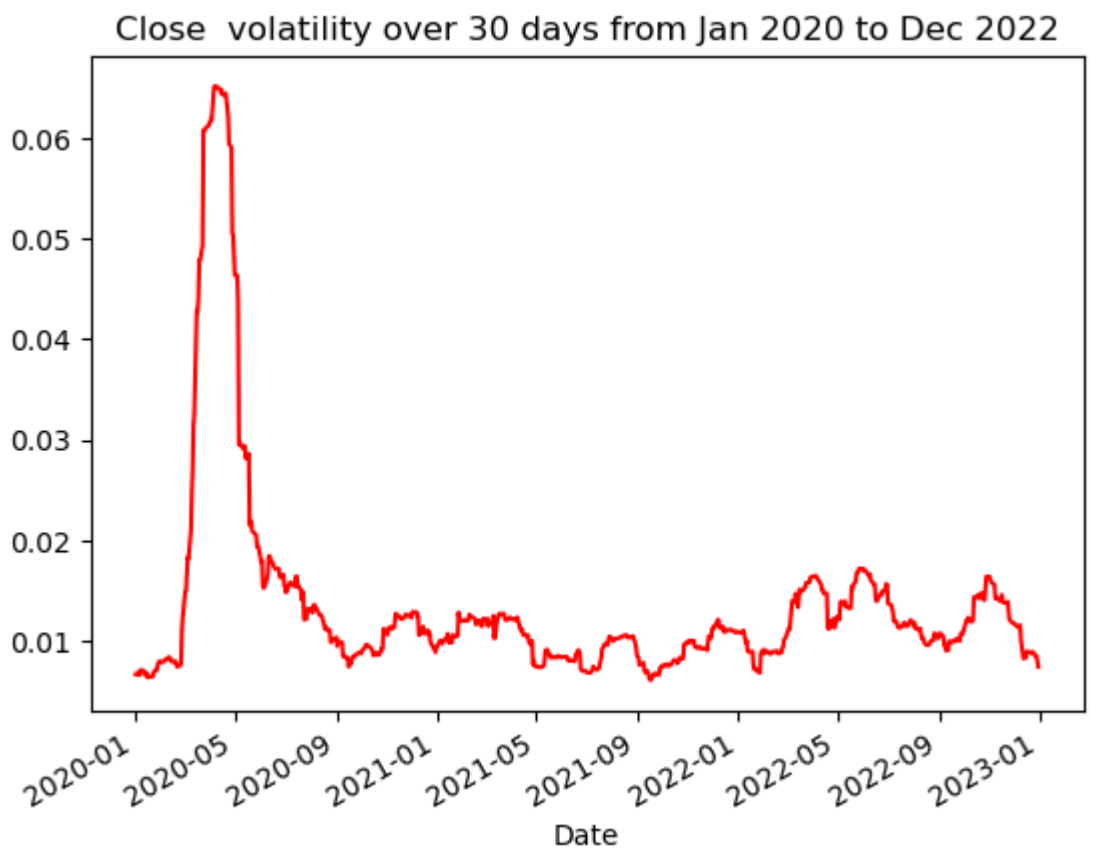
2638 rows × 7 columns

In [76]:
```
# mc.rolling?
```

In [77]:
```
#15 Days  rolling volatility
(mc
.assign(pct_change = mc.Close.pct_change())
.rolling (window = 15, min_periods = 15)
.std()
['pct_change']
.plot()
)
plt.title('15 days rolling Volatility')
plt.show()
```



In [78]:
```
#30 Days volatility for 2020- 2022
(mc
 .assign(pct_change_30 = mc.Close.pct_change())
 .rolling (window = 30, min_periods = 30)
 ['pct_change_30']
 .std()
 .loc['jan 2020' : 'dec 2022']
 .plot(color = 'r')
)
plt.title('Close  volatility over 30 days from Jan 2020 to Dec 2022')
plt.show()
```



Step 12: Simple Moving Average

Shifting method and built - in method

In [79]:
```
(mc
 .assign(s1 = mc.Close.shift(1),
         s2 = mc.Close.shift(2),
        ))
```

Out[79]:

Date	Adj Close	Close	High	Low	Open	Volume	s1	s2
2013-01-02	67.764641	90.120003	90.349998	89.330002	89.400002	7377200	NaN	NaN
2013-01-03	68.148117	90.629997	90.699997	90.160004	90.309998	5473500	90.120003	NaN
2013-01-04	67.561623	89.849998	90.790001	89.669998	90.620003	5374100	90.629997	90.120003
2013-01-07	68.358658	90.910004	91.050003	89.250000	89.769997	5820900	89.849998	90.629997
2013-01-08	68.381241	90.940002	90.959999	89.970001	90.540001	6285200	90.910004	89.849998
...	...	...	...	...	...	...	...	...
2023-06-20	293.040009	293.040009	297.179993	292.679993	293.649994	2729800	293.700012	292.609985
2023-06-21	294.519989	294.519989	295.160004	292.809998	293.320007	2837100	293.040009	293.700012
2023-06-22	293.299988	293.299988	295.079987	291.519989	294.619995	1799400	294.519989	293.040009
2023-06-23	289.910004	289.910004	292.450012	289.640015	291.350006	3719700	293.299988	294.519989
2023-06-26	287.200012	287.200012	289.750000	287.070007	289.640015	402173	289.910004	293.299988

2638 rows × 8 columns

In [80]:
```
(mc
 .assign(shift1 = mc['Close'].shift(1),    # shifting col with
         shift2 = mc['Close'].shift(2),
         ma3 = lambda data_ : data_.loc[:,['Close', 'shift1','shift2']].mean(axis ='columns'),  #manually
         ma3_builtin = mc['Close'].rolling(3).mean()    #builtin method
        )
)
```

Out[80]:

Date	Adj Close	Close	High	Low	Open	Volume	shift1	shift2	ma3	ma3_builtin
2013-01-02	67.764641	90.120003	90.349998	89.330002	89.400002	7377200	NaN	NaN	90.120003	NaN
2013-01-03	68.148117	90.629997	90.699997	90.160004	90.309998	5473500	90.120003	NaN	90.375000	NaN
2013-01-04	67.561623	89.849998	90.790001	89.669998	90.620003	5374100	90.629997	90.120003	90.199999	90.199999
2013-01-07	68.358658	90.910004	91.050003	89.250000	89.769997	5820900	89.849998	90.629997	90.463333	90.463333
2013-01-08	68.381241	90.940002	90.959999	89.970001	90.540001	6285200	90.910004	89.849998	90.566668	90.566668
...	...	...	...	...	...	...	...	...	...	...
2023-06-20	293.040009	293.040009	297.179993	292.679993	293.649994	2729800	293.700012	292.609985	293.116669	293.116669
2023-06-21	294.519989	294.519989	295.160004	292.809998	293.320007	2837100	293.040009	293.700012	293.753337	293.753337
2023-06-22	293.299988	293.299988	295.079987	291.519989	294.619995	1799400	294.519989	293.040009	293.619995	293.619995
2023-06-23	289.910004	289.910004	292.450012	289.640015	291.350006	3719700	293.299988	294.519989	292.576660	292.576660
2023-06-26	287.200012	287.200012	289.750000	287.070007	289.640015	402173	289.910004	293.299988	290.136668	290.136668

2638 rows × 10 columns

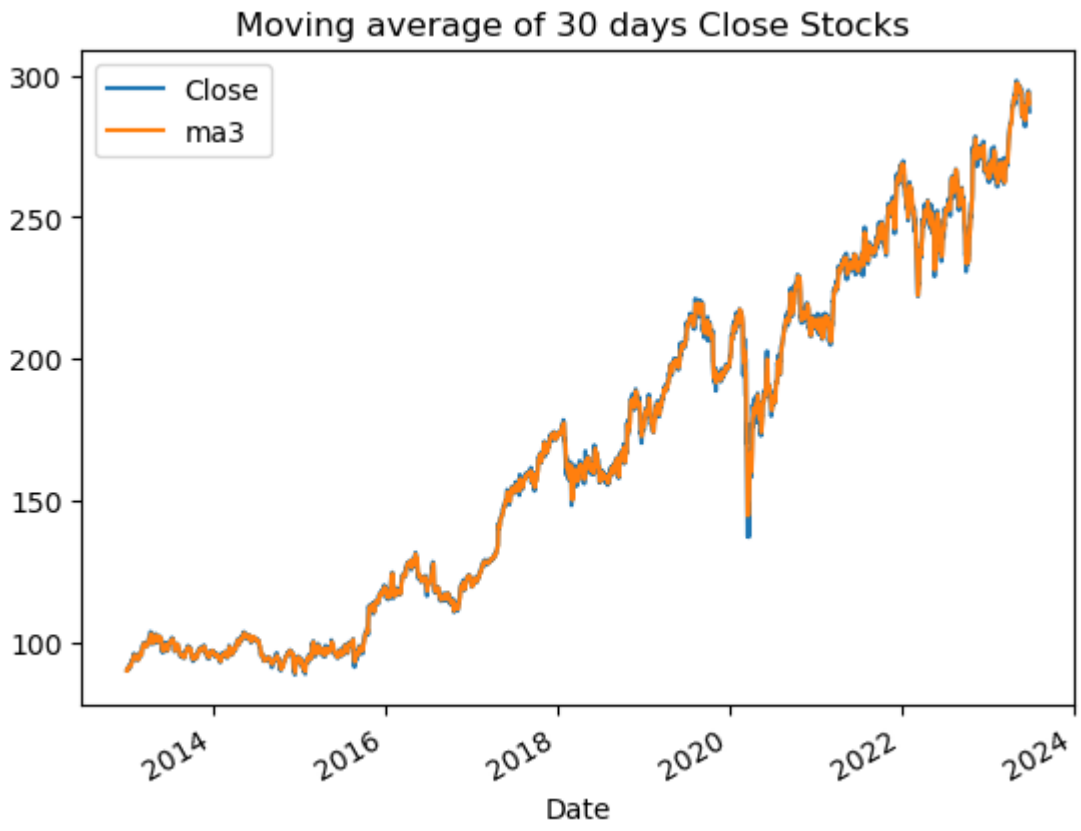
In [81]:
```
(mc
 .assign(shift1 = mc['Close'].shift(1),    # shifting col with
         shift2 = mc['Close'].shift(2),
         ma3 = lambda data_ : data_.loc[:,['Close', 'shift1','shift2']].mean(axis ='columns'),  #manually
         ma3_builtin = mc['Close'].rolling(3).mean()    #builtin method

        )
  [['Close', 'ma3']]
)
```

Out[81]:

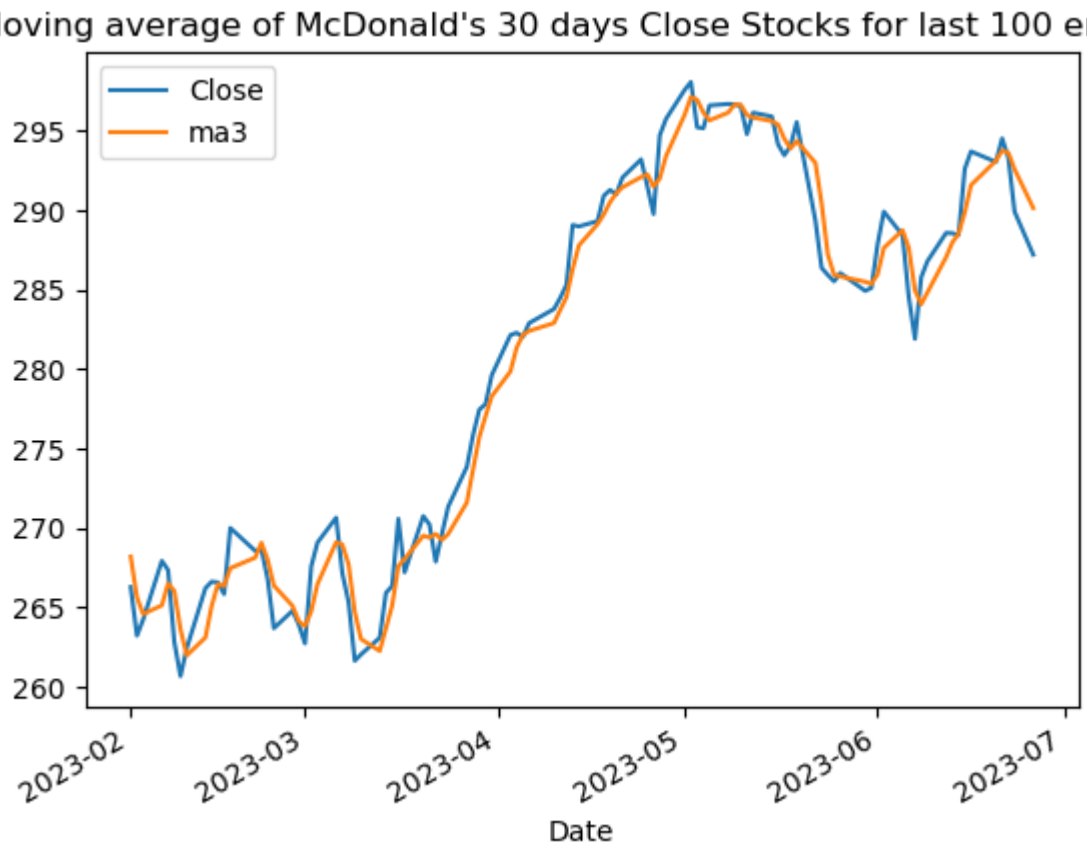
Date	Close	ma3
2013-01-02	90.120003	90.120003
2013-01-03	90.629997	90.375000
2013-01-04	89.849998	90.199999
2013-01-07	90.910004	90.463333
2013-01-08	90.940002	90.566668
...	...	...
2023-06-20	293.040009	293.116669
2023-06-21	294.519989	293.753337
2023-06-22	293.299988	293.619995
2023-06-23	289.910004	292.576660
2023-06-26	287.200012	290.136668

2638 rows × 2 columns

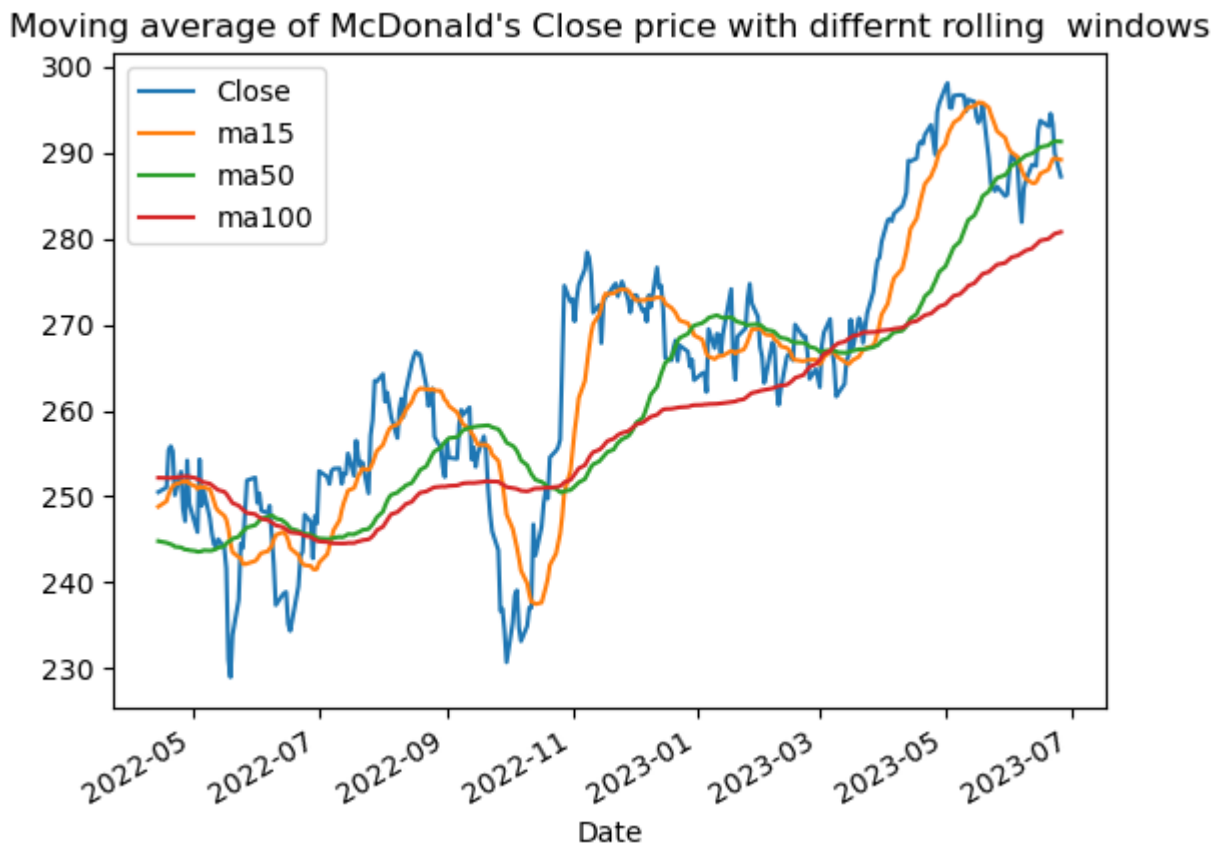
In [82]:
```
(mc
 .assign(shift1 = mc['Close'].shift(1),    # shifting col with
         shift2 = mc['Close'].shift(2),
         ma3 = lambda data_ : data_.loc[:,['Close', 'shift1','shift2']].mean(axis ='columns'),  #manually
         ma3_builtin = mc['Close'].rolling(3).mean()    #builtin method

        )
  [['Close', 'ma3']]
 .plot()
)
plt.title('Moving average of 30 days Close Stocks ')
plt.show()
```



In [83]:
```
# For last 100 entries
(mc
 .assign(shift1 = mc['Close'].shift(1),    # shifting col with
         shift2 = mc['Close'].shift(2),
         # manually calculating mean of 3 columns and storing in ma3
         ma3 = lambda data_ : data_.loc[:,['Close', 'shift1','shift2']].mean(axis ='columns'),
         #builtin method
         ma3_builtin = mc['Close'].rolling(3).mean()

        )
  [['Close', 'ma3']]
 .iloc[-100:]
 .plot()
)
plt.title('Moving average of McDonald\'s 30 days Close Stocks for last 100 entries ')
plt.show()
```



In [84]:
```
(mc
 .assign(
         ma15 = mc['Close'].rolling(15).mean(),
         ma50 = mc['Close'].rolling(50).mean(),
         ma100 = mc['Close'].rolling(100).mean(),

        )
[['Close', 'ma15','ma50', 'ma100']]
 .iloc[-300:]
 .plot()
)
plt.title('Moving average of McDonald\'s Close price with differnt rolling  windows')
plt.show()
```

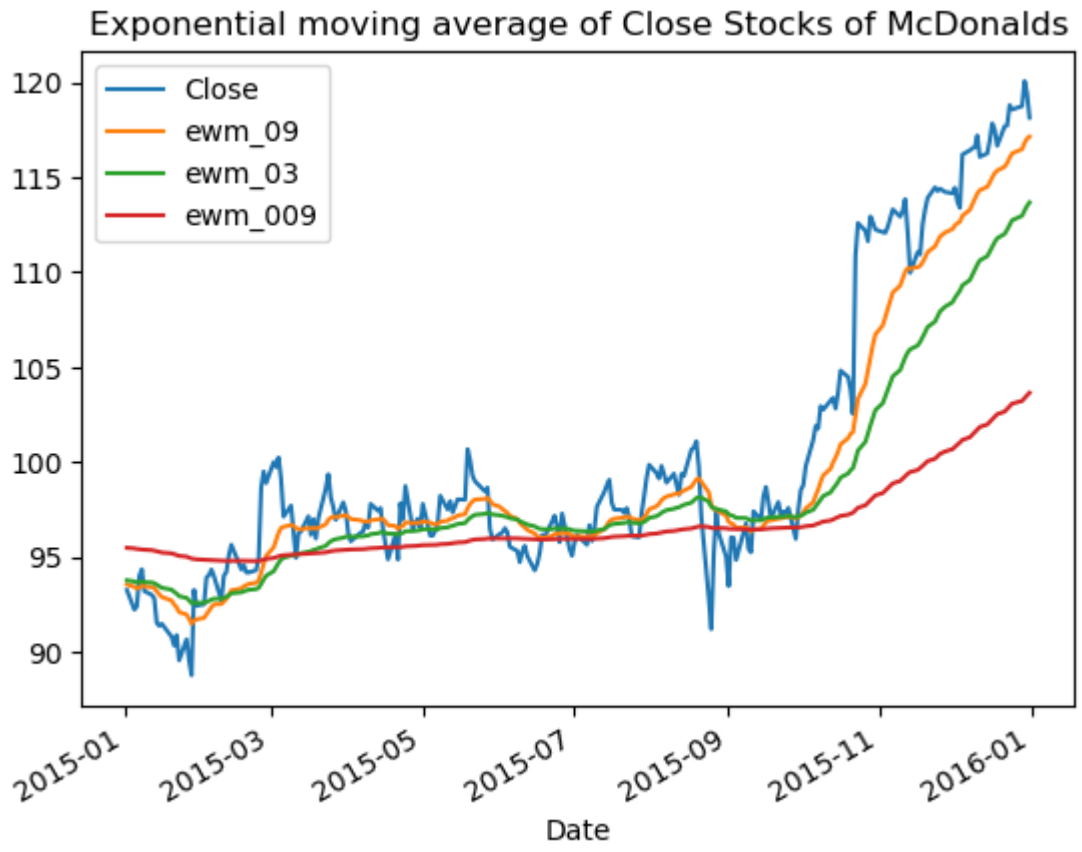


15 days rolling average better fits compared to 50days and 100 days. By doing a more number of days rolling average it smooths out more.

Step 13: Exponetial Moving Average (data.ewm() create rolling aggregator)

In [85]:
```
#mc.ewm?
```

In [86]:
```
(mc
 .assign(
         ewm_09 = mc['Close'].ewm(alpha = .09).mean(),
         ewm_03 = mc['Close'].ewm(alpha = .0392).mean(),
         ewm_009 = mc['Close'].ewm(alpha =.009).mean(),


        )
[['Close','ewm_09', 'ewm_03','ewm_009']]
 .loc['jan 2015':'dec 2015']
 .plot()
)
plt.title('Exponential moving average of Close Stocks of McDonalds')
plt.show()
```



Step 14: Technical

On-balance Volume (OBV)

$$OBV = OBV_{\text{Previous}} + \begin{cases} \text{volume} & \text{if close} > \text{close}_{\text{previous}} \\ 0 & \text{if close} = \text{close}_{\text{previous}} \\ -\text{volume} & \text{if close} < \text{close}_{\text{previous}} \end{cases}$$

Where

- OBV: On-balance Volume level
- OBV_{Previous}: Previous On-balance Volume level
- volume: Last trending volume amount

In [87]:
```
# To get previous value..we can do shift
mc['Close']
```

Out[87]:
```
Date
2013-01-02     90.120003
2013-01-03     90.629997
2013-01-04     89.849998
2013-01-07     90.910004
2013-01-08     90.940002
                   ...
2023-06-20    293.040009
2023-06-21    294.519989
2023-06-22    293.299988
2023-06-23    289.910004
2023-06-26    287.200012
Name: Close, Length: 2638, dtype: float64
```

In [88]:
```
# To get previous value..we can do shift
(mc['Close'].shift(1))
```

Out[88]:
```
Date
2013-01-02           NaN
2013-01-03     90.120003
2013-01-04     90.629997
2013-01-07     89.849998
2013-01-08     90.910004
                  ...
2023-06-20    293.700012
2023-06-21    293.040009
2023-06-22    294.519989
2023-06-23    293.299988
2023-06-26    289.910004
Name: Close, Length: 2638, dtype: float64
```

In [89]:
```
# naive method
def OBV(data):
    data=data.copy()
    data['OBV'] = 0.0

    # loop through data and calculate OBV
    for i in range (1,len(data)):
        if data['Close'][i] > data['Close'][i-1]:
            data['OBV'][i] = data['OBV'][i-1] + data['Volume'][i]

        elif data['Close'][i] < data['Close'][i-1]:
            data['OBV'][i] = data['OBV'][i-1] - data['Volume'][i]

        else:
            data['OBV'][i] =  data['OBV'][i-1]
    return data
```

In [90]:
```
OBV(mc)
```

Out[90]:

Date	Adj Close	Close	High	Low	Open	Volume	OBV
2013-01-02	67.764641	90.120003	90.349998	89.330002	89.400002	7377200	0.0
2013-01-03	68.148117	90.629997	90.699997	90.160004	90.309998	5473500	5473500.0
2013-01-04	67.561623	89.849998	90.790001	89.669998	90.620003	5374100	99400.0
2013-01-07	68.358658	90.910004	91.050003	89.250000	89.769997	5820900	5920300.0
2013-01-08	68.381241	90.940002	90.959999	89.970001	90.540001	6285200	12205500.0
...	...	...	...	...	...	...	...
2023-06-20	293.040009	293.040009	297.179993	292.679993	293.649994	2729800	522173900.0
2023-06-21	294.519989	294.519989	295.160004	292.809998	293.320007	2837100	525011000.0
2023-06-22	293.299988	293.299988	295.079987	291.519989	294.619995	1799400	523211600.0
2023-06-23	289.910004	289.910004	292.450012	289.640015	291.350006	3719700	519491900.0
2023-06-26	287.200012	287.200012	289.750000	287.070007	289.640015	402173	519089727.0

2638 rows × 7 columns

In [91]:
```
# naive method runs slow
```

In [92]:
```
%%timeit
OBV(mc)
```

```
1.13 s ± 87.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [93]:
```
# mc.Volume.where?
```

In [94]:
```
%%timeit
(mc.assign(close_prev = mc.Close.shift(1),
             vol = 0,
             obv = lambda dt: dt.vol.where(cond = dt.Close == dt.close_prev,
                                           other= dt.Volume.where(cond =dt.Close > dt.close_prev,
                                                    other=-dt.Volume.where(cond =dt.Close < dt.close_prev, ot
                                                                ))).cumsum()
           )
)
```

```
3.04 ms ± 98.8 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

In [95]:
```
pd.Series(np.select(condlist = [mc.Close < 90.1 ,mc.Close >284],
                    choicelist= [90, 284], default = 100))
```

Out[95]:
```
0       100
1       100
2        90
3       100
4       100
       ...
2633    284
2634    284
2635    284
2636    284
2637    284
Length: 2638, dtype: int32
```

In [96]:
```
(mc
.assign(vol =np.select([mc.Close > mc.Close.shift(1),
                        mc.Close == mc.Close.shift(1),
                        mc.Close < mc.Close.shift(1)],
                       [mc.Volume, 0, -mc.Volume]
                      ),
        obv = lambda data_: data_.vol.cumsum(),
       )
)
```

Out[96]:

Date	Adj Close	Close	High	Low	Open	Volume	vol	obv
2013-01-02	67.764641	90.120003	90.349998	89.330002	89.400002	7377200	0	0
2013-01-03	68.148117	90.629997	90.699997	90.160004	90.309998	5473500	5473500	5473500
2013-01-04	67.561623	89.849998	90.790001	89.669998	90.620003	5374100	-5374100	99400
2013-01-07	68.358658	90.910004	91.050003	89.250000	89.769997	5820900	5820900	5920300
2013-01-08	68.381241	90.940002	90.959999	89.970001	90.540001	6285200	6285200	12205500
...	...	...	...	...	...	...	...	...
2023-06-20	293.040009	293.040009	297.179993	292.679993	293.649994	2729800	-2729800	522173900
2023-06-21	294.519989	294.519989	295.160004	292.809998	293.320007	2837100	2837100	525011000
2023-06-22	293.299988	293.299988	295.079987	291.519989	294.619995	1799400	-1799400	523211600
2023-06-23	289.910004	289.910004	292.450012	289.640015	291.350006	3719700	-3719700	519491900
2023-06-26	287.200012	287.200012	289.750000	287.070007	289.640015	402173	-402173	519089727

2638 rows × 8 columns

In [97]:
```
def cal_obv(data, close_col= 'Close', vol_col= 'Volume'):
    close = data[close_col]
    vol = data[vol_col]
    close_shift = close.shift(1)
    return (data
            .assign(vol =np.select([close > close.shift(1),
                                    close == close.shift(1),
                                    close < close.shift(1)],
                                   [vol, 0, -vol]),
        obv = lambda data_: data_.vol.fillna(0).cumsum(),
       )
            ['obv']
           )

(mc
.assign(obv=cal_obv)
)
```

Out[97]:

Date	Adj Close	Close	High	Low	Open	Volume	obv
2013-01-02	67.764641	90.120003	90.349998	89.330002	89.400002	7377200	0
2013-01-03	68.148117	90.629997	90.699997	90.160004	90.309998	5473500	5473500
2013-01-04	67.561623	89.849998	90.790001	89.669998	90.620003	5374100	99400
2013-01-07	68.358658	90.910004	91.050003	89.250000	89.769997	5820900	5920300
2013-01-08	68.381241	90.940002	90.959999	89.970001	90.540001	6285200	12205500
...	...	...	...	...	...	...	...
2023-06-20	293.040009	293.040009	297.179993	292.679993	293.649994	2729800	522173900
2023-06-21	294.519989	294.519989	295.160004	292.809998	293.320007	2837100	525011000
2023-06-22	293.299988	293.299988	295.079987	291.519989	294.619995	1799400	523211600
2023-06-23	289.910004	289.910004	292.450012	289.640015	291.350006	3719700	519491900
2023-06-26	287.200012	287.200012	289.750000	287.070007	289.640015	402173	519089727

2638 rows × 7 columns

Acumulation/ Distribution Indicator (A/D)

Goals

$$MFM = \frac{(Close - Low) + (Close - High)}{High - Low}$$

where:

- MFM : Money Flow Multiplier
- Close : closing price
- Low : Low price of the period
- High : High price of the period

- MFV = MFM X Period volume
- A/D = previous A/D + CMFV

where:

- MFV : Money Flow Volume
- CMFV : Current Money Flow Volume

In [98]:
```
(mc
.assign(mfm =((mc.Close-mc.Low)+(mc.Close-mc.High))/(mc.High -mc.Low),
        mfv = lambda data_ : data_.mfm *data_.Volume,
       cmfv = lambda data_ : data_.mfv.cumsum())
)
```

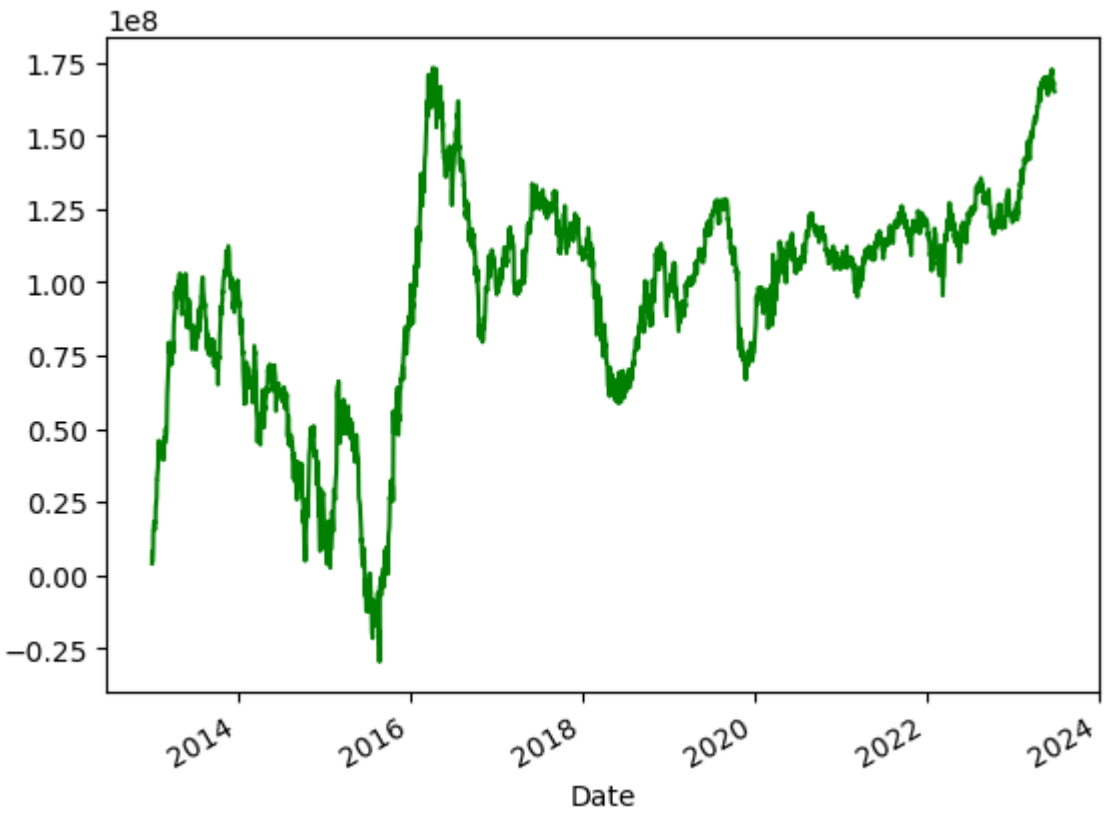
Out[98]:

Date	Adj Close	Close	High	Low	Open	Volume	mfm	mfv	cmfv
2013-01-02	67.764641	90.120003	90.349998	89.330002	89.400002	7377200	0.549027	4.050278e+06	4.050278e+06
2013-01-03	68.148117	90.629997	90.699997	90.160004	90.309998	5473500	0.740739	4.054433e+06	8.104711e+06
2013-01-04	67.561623	89.849998	90.790001	89.669998	90.620003	5374100	-0.678572	-3.646712e+06	4.457999e+06
2013-01-07	68.358658	90.910004	91.050003	89.250000	89.769997	5820900	0.844445	4.915432e+06	9.373431e+06
2013-01-08	68.381241	90.940002	90.959999	89.970001	90.540001	6285200	0.959603	6.031295e+06	1.540473e+07
...	...	...	...	...	...	...	...	...	...
2023-06-20	293.040009	293.040009	297.179993	292.679993	293.649994	2729800	-0.839993	-2.293013e+06	1.672119e+08
2023-06-21	294.519989	294.519989	295.160004	292.809998	293.320007	2837100	0.455308	1.291755e+06	1.685036e+08
2023-06-22	293.299988	293.299988	295.079987	291.519989	294.619995	1799400	0.000000	0.000000e+00	1.685036e+08
2023-06-23	289.910004	289.910004	292.450012	289.640015	291.350006	3719700	-0.807837	-3.004911e+06	1.654987e+08
2023-06-26	287.200012	287.200012	289.750000	287.070007	289.640015	402173	-0.902981	-3.631546e+05	1.651356e+08

2638 rows × 9 columns

In [99]:
```
# refactored function
def cal_ad(data, close_col= 'Close', vol_col= 'Volume', low_col= 'Low', high_col= 'High' ):
    close = data[close_col]
    low   = data[low_col]
    high  = data[high_col]
    return (data
            .assign(mfm  = ((close- low)+(close-high))/(high -low),
                    mfv  = lambda data_ : data_.mfm *data_[vol_col],
                    cmfv = lambda data_ : data_.mfv.cumsum())
            .cmfv

       )

(mc
.assign(ad=cal_ad)
.ad
.plot(color='g')
)
```

Out[99]: <Axes: xlabel='Date'>



Analysising Starbucks data

In [100]:
```
def tweak_data_sb():
    data = yf.download(ticks, start, end )
    sbx = data.iloc[: , 1::2]
    return (sbx.pipe(column_outer))
sb = tweak_data_sb()
sb
```

```
[**********************100%**********************]  2 of 2 completed
```

Out[100]:

Date	Adj Close	Close	High	Low	Open	Volume
2013-01-02	22.833767	27.500000	27.500000	27.129999	27.295000	13267600
2013-01-03	22.987379	27.684999	27.805000	27.500000	27.535000	14670400
2013-01-04	23.120226	27.844999	28.000000	27.655001	27.764999	10911400
2013-01-07	23.132690	27.860001	27.895000	27.504999	27.700001	8720000
2013-01-08	23.091167	27.809999	27.860001	27.535000	27.790001	9613400
...	...	...	...	...	...	...
2023-06-20	101.269997	101.269997	102.489998	100.900002	101.860001	5506400
2023-06-21	101.870003	101.870003	102.459999	100.360001	100.599998	5508800
2023-06-22	100.849998	100.849998	101.639999	99.639999	101.419998	6125500
2023-06-23	98.339996	98.339996	99.730003	97.519997	99.650002	18765000
2023-06-26	98.400002	98.400002	98.639999	97.480003	98.339996	1256095

2638 rows × 6 columns

In [101]:
```
sb.isna().sum()
```

Out[101]:
```
Adj Close    0
Close        0
High         0
Low          0
Open         0
Volume       0
dtype: int64
```

In [102]:
```
(sb[['Adj Close']])
```

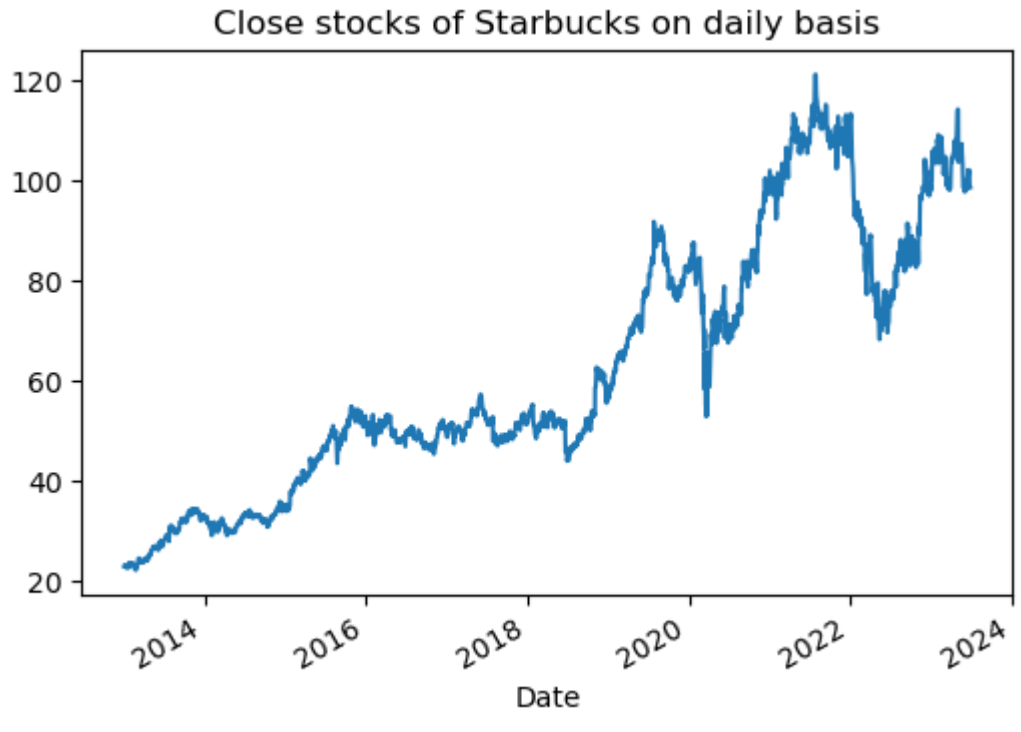
Out[102]:

Date	Adj Close
2013-01-02	22.833767
2013-01-03	22.987379
2013-01-04	23.120226
2013-01-07	23.132690
2013-01-08	23.091167
...	...
2023-06-20	101.269997
2023-06-21	101.870003
2023-06-22	100.849998
2023-06-23	98.339996
2023-06-26	98.400002

2638 rows × 1 columns

In [ ]:
```

```

In [103]:
```
(sb['Adj Close']
 .plot(figsize=(6,4)))
plt.title('Close stocks of Starbucks on daily basis')
plt.show()
```

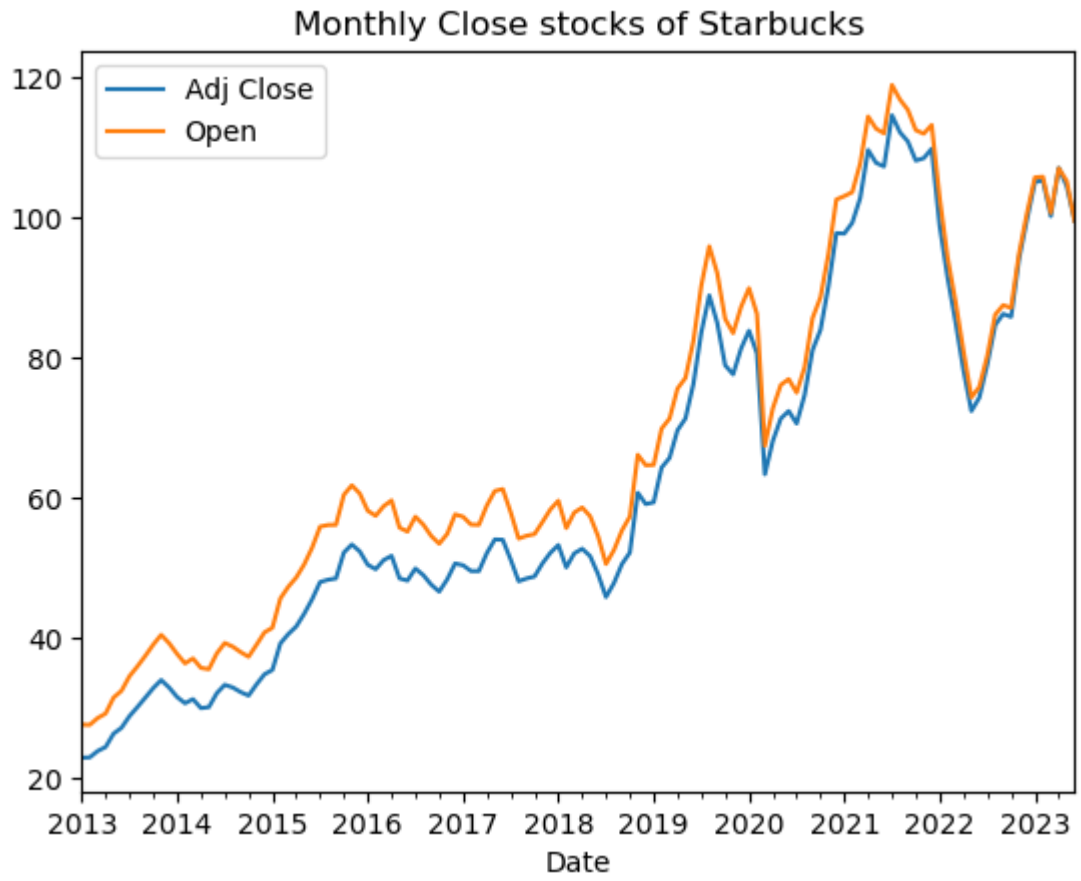


Giving 'Close' value of STARBUCKS on daily basis from 2013 to current date

Resampling

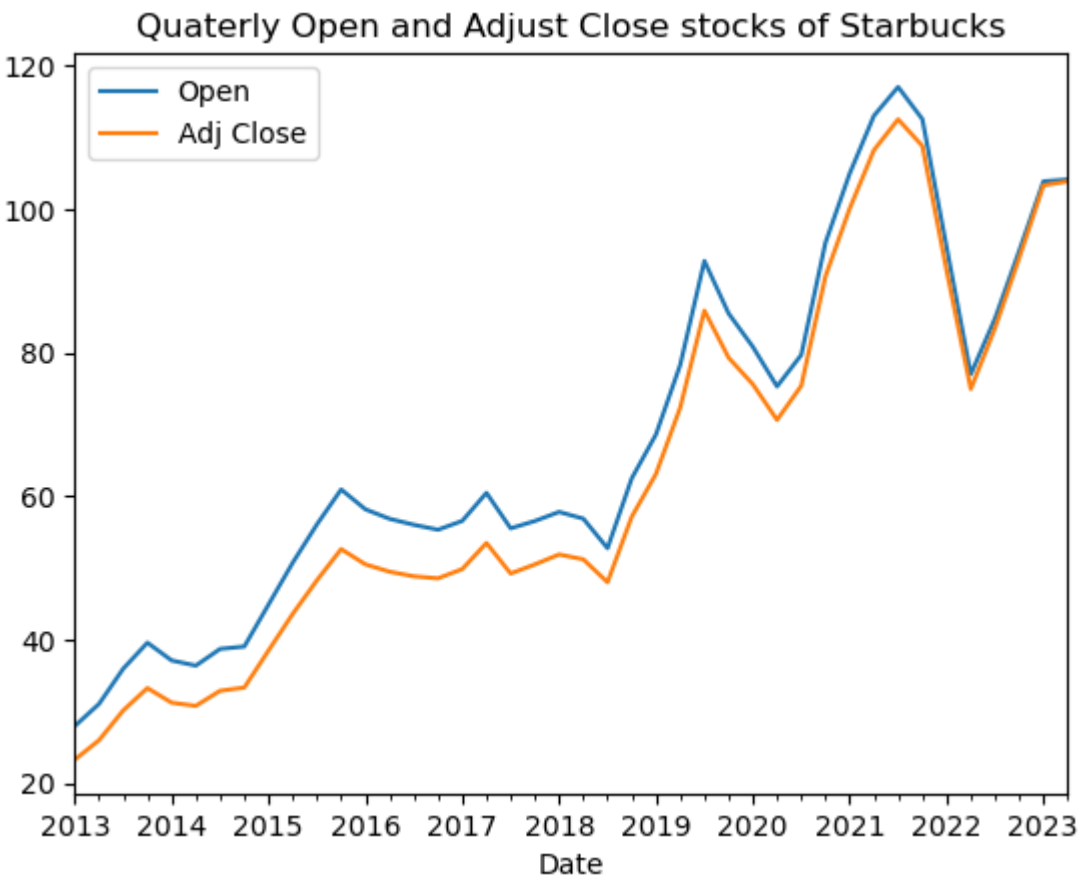
1. resampling on basis of Month : resample('M')

In [104]:
```
(sb
 .resample('M')
 ['Adj Close','Open']
 .mean()
 .plot()
)
plt.title('Monthly Close stocks of Starbucks')
plt.show()
```



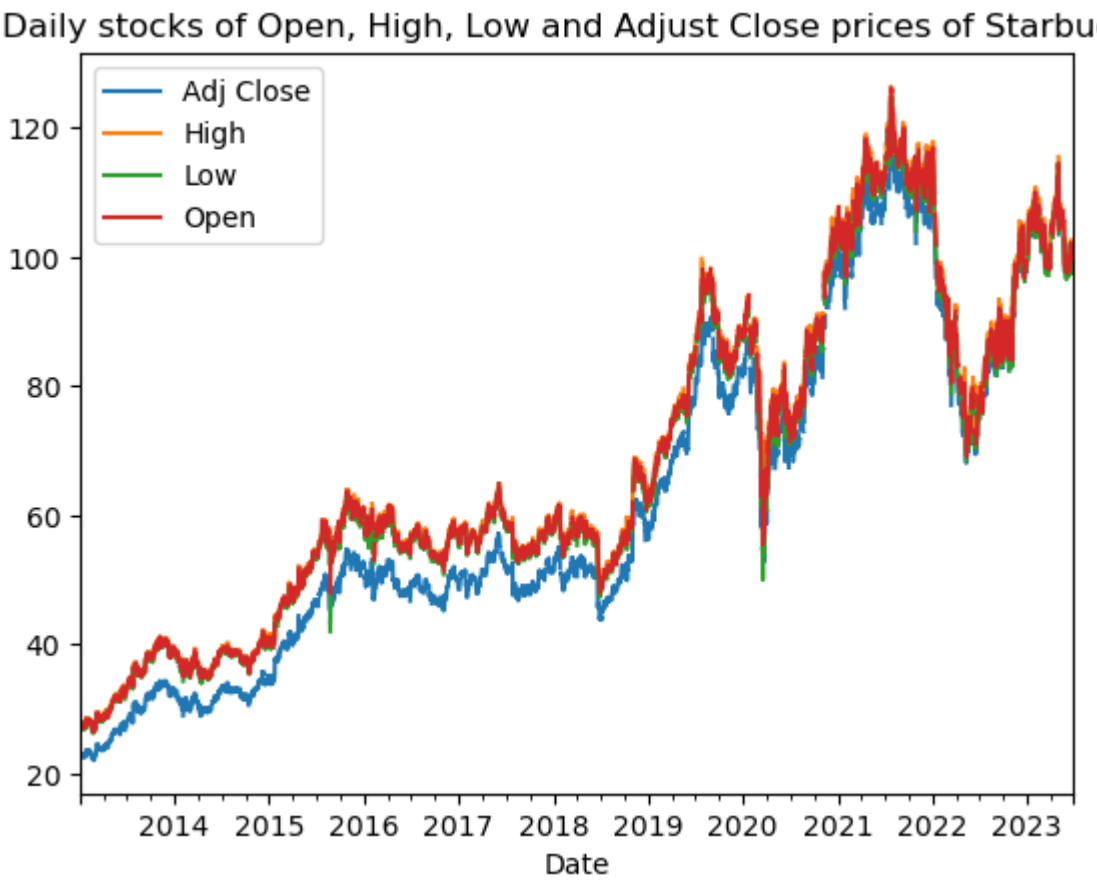
2. resampling on basis of Quaterly : resample('Q')

In [105]:
```
(sb
 .resample('Q')
 ['Open','Adj Close']
 .mean()
 .plot()
)
plt.title('Quaterly Open and Adjust Close stocks of Starbucks')
plt.show()
```



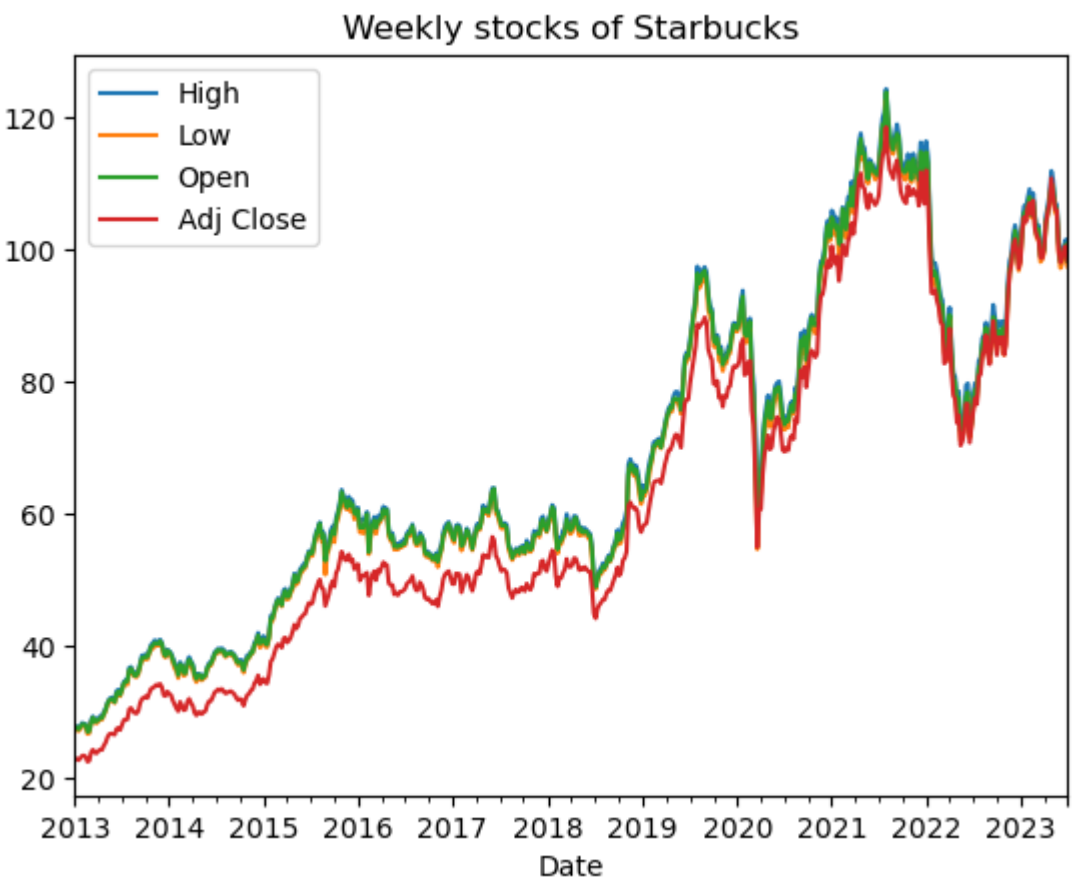
3. resampling on basis of daily : resample('d')

In [106]:
```
(sb
 .resample('d')
 ['Adj Close','High','Low','Open']
 .mean()
 .plot()
)
plt.title('Daily stocks of Open, High, Low and Adjust Close prices of Starbucks')
plt.show()
```

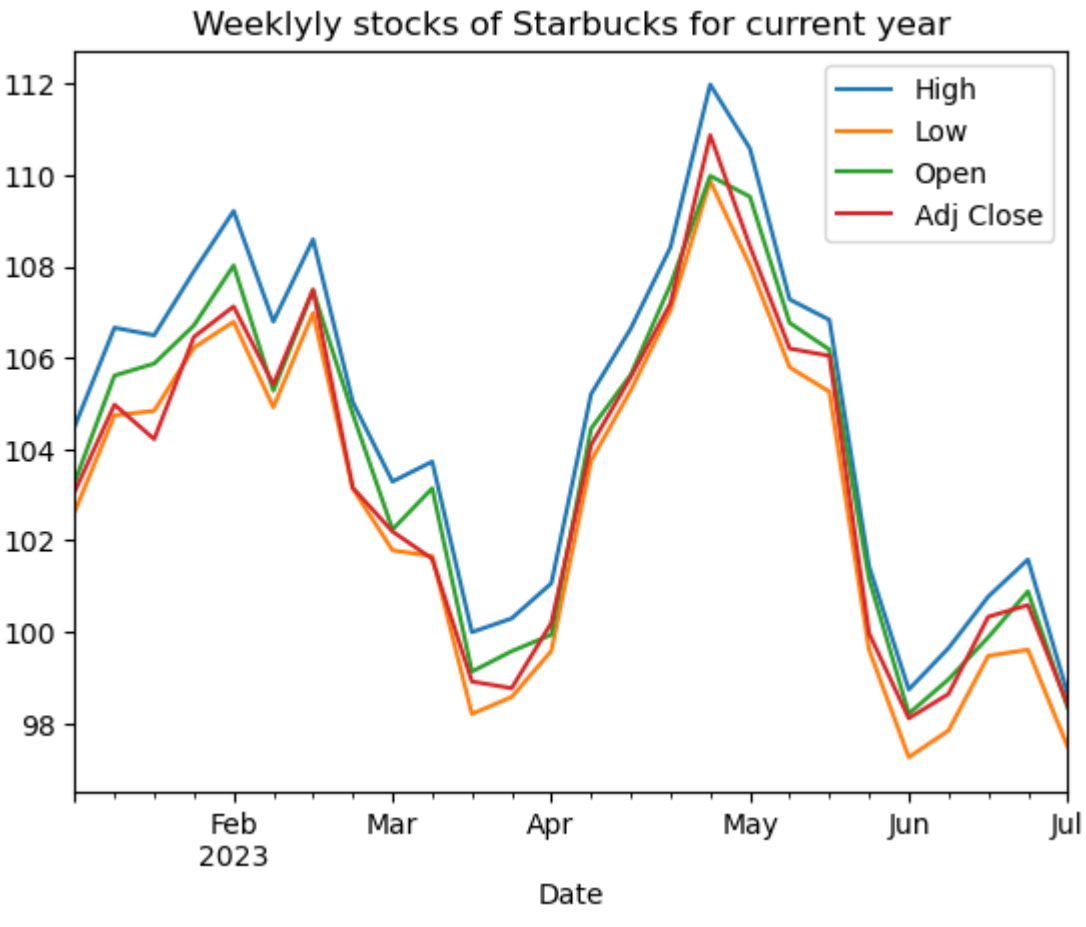


4. resampling on basis of weekly : resample('w')

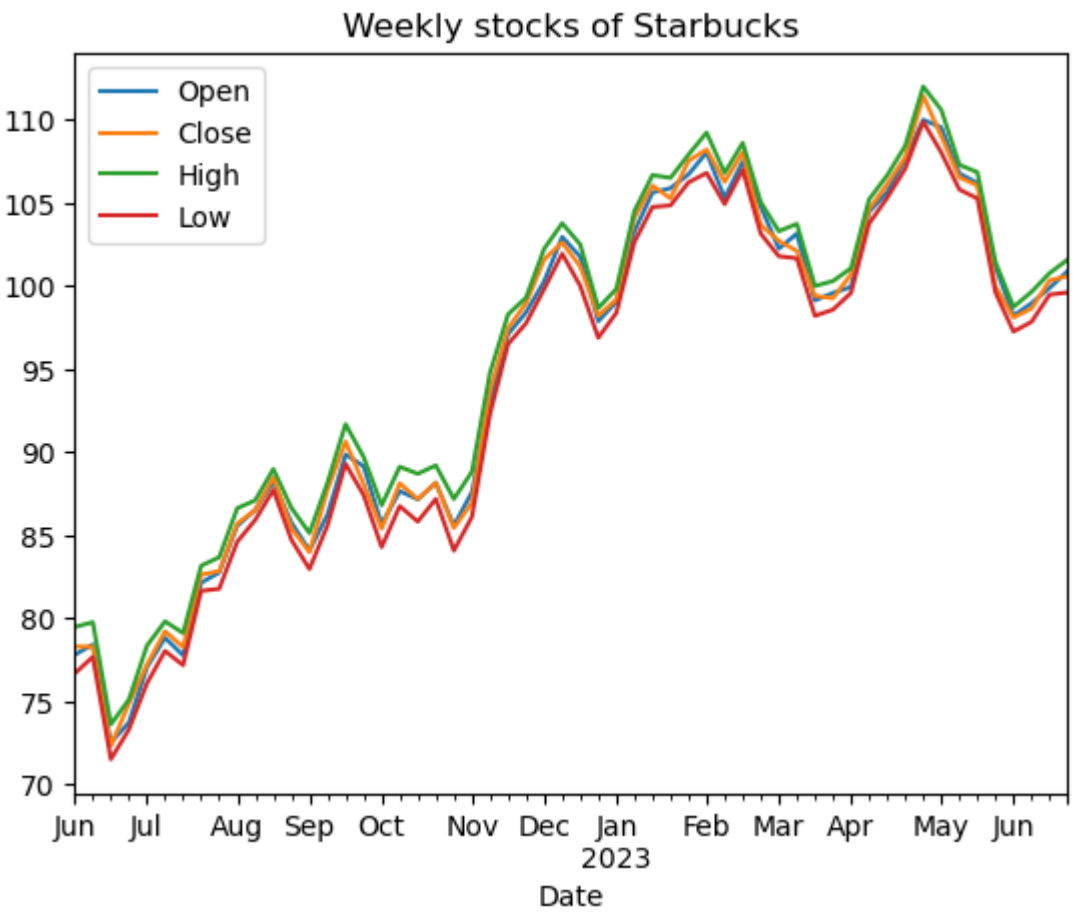
In [107]:
```
(sb
 .resample('w')
 ['High','Low','Open','Adj Close']
 .mean()
 .plot()
)
plt.title('Weekly stocks of Starbucks')
plt.show()
```



In [108]:
```
(sb
 .iloc[-120:,:]
  .resample('w')
 ['High','Low','Open','Adj Close']
 .mean()
 .plot()
)
plt.title('Weeklyly stocks of Starbucks for current year')
plt.show()
```

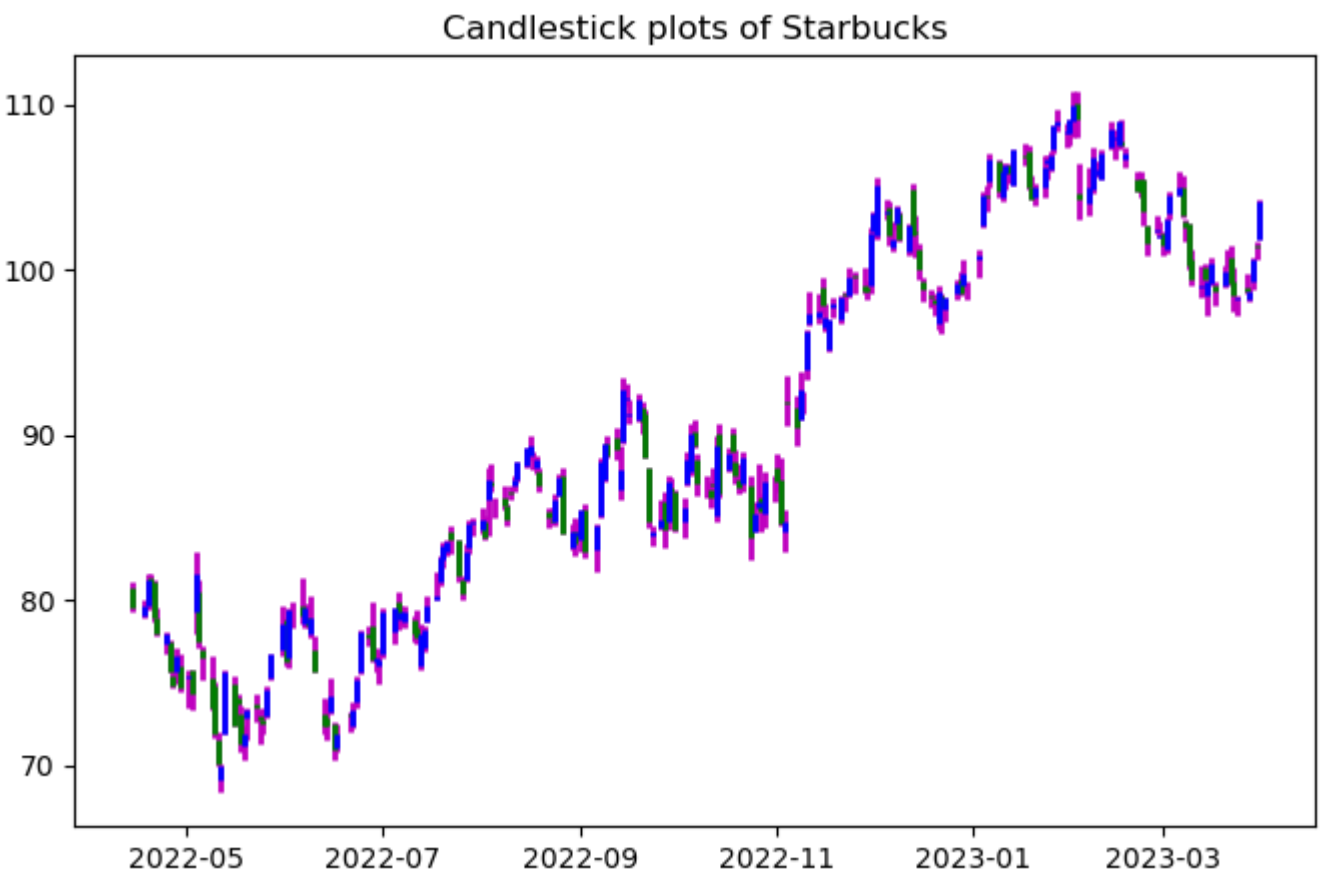


In [109]:
```
#Last one year weekly data
(sb
 .resample('w')
 ['Open','Close','High', 'Low']
 .mean()
 .loc['jun 2022':'jun 2023']
 .plot()
)
plt.title('Weekly stocks of Starbucks')
plt.show()
```

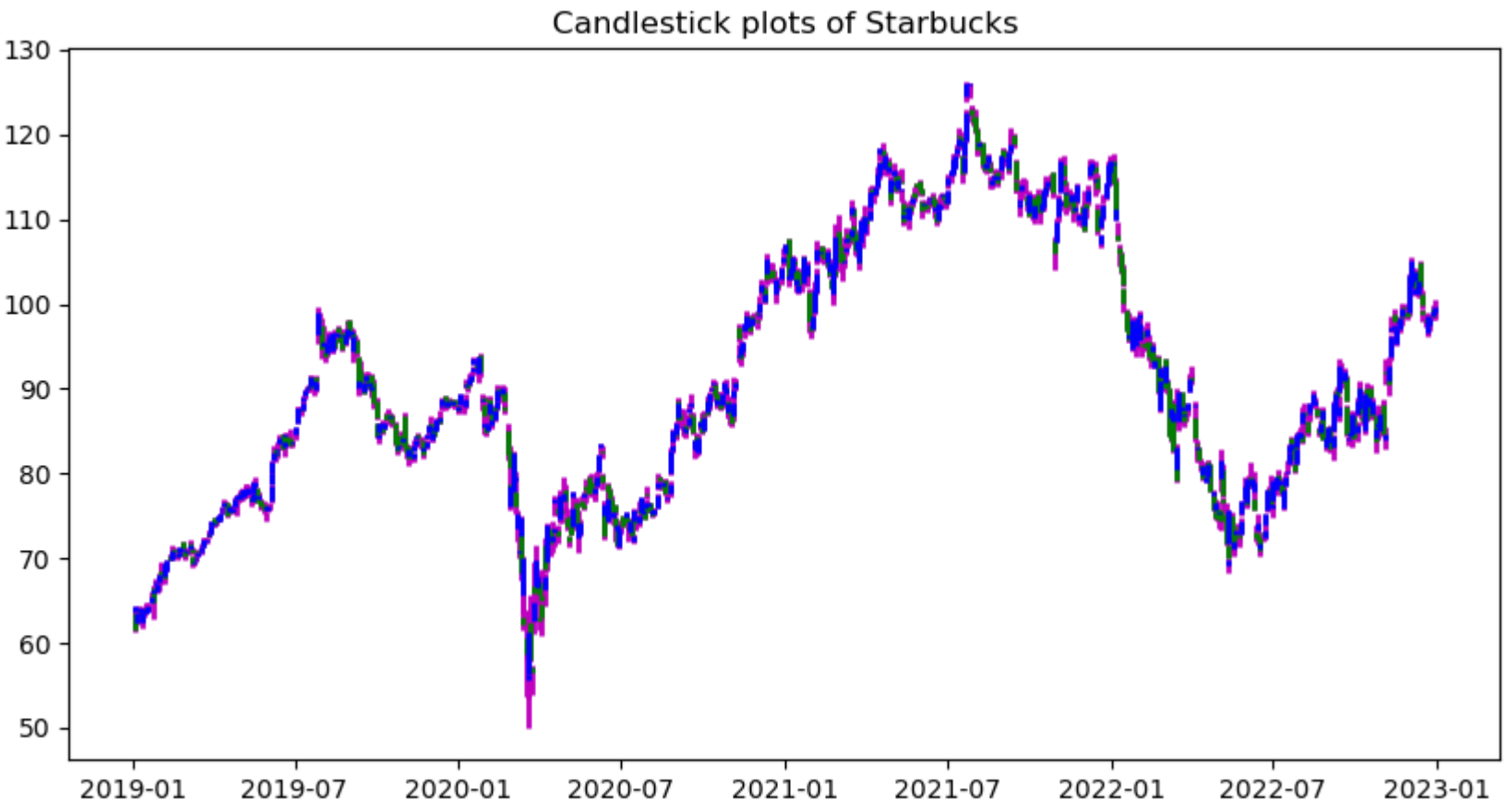


Candlestick Plot

In [110]:
```
# last 300 entries
fig, ax = plt.subplots (figsize = (8,5))

(sb.iloc[-300:,]
 .resample('d')
 .agg({'Open' : "first", 'High' : "max", 'Low':"min", 'Close' : "last"})
 .loc['jan 2020' : 'march 2023']
 .pipe(candle_plot, ax)
)
plt.title('Candlestick plots of Starbucks')
plt.show()
```



In [111]:
```
fig, ax = plt.subplots (figsize = (10,5))
(sb
 .resample('d')
 .agg({'Open' : "first", 'High' : "max", 'Low':"min", 'Close' : "last"})
 .loc['jan 2019' : 'dec 2022']
 .pipe(candle_plot, ax)
)
plt.title('Candlestick plots of Starbucks')
plt.show()
```



Return

In [112]:
```
sb.pct_change()
```

Out[112]:

Date	Adj Close	Close	High	Low	Open	Volume
2013-01-02	NaN	NaN	NaN	NaN	NaN	NaN
2013-01-03	0.006727	0.006727	0.011091	0.013638	0.008793	0.105731
2013-01-04	0.005779	0.005779	0.007013	0.005636	0.008353	-0.256230
2013-01-07	0.000539	0.000539	-0.003750	-0.005424	-0.002341	-0.200836
2013-01-08	-0.001795	-0.001795	-0.001255	0.001091	0.003249	0.102454
...	...	...	...	...	...	...
2023-06-20	-0.005890	-0.005890	-0.000195	-0.005715	-0.001470	-0.529701
2023-06-21	0.005925	0.005925	-0.000293	-0.005352	-0.012370	0.000436
2023-06-22	-0.010013	-0.010013	-0.008003	-0.007174	0.008151	0.111948
2023-06-23	-0.024888	-0.024888	-0.018792	-0.021277	-0.017452	2.063423
2023-06-26	0.000610	0.000610	-0.010930	-0.000410	-0.013146	-0.933062

2638 rows × 6 columns

In [113]:
```
sb.pct_change(periods = 2)
```

Out[113]:

Date	Adj Close	Close	High	Low	Open	Volume
2013-01-02	NaN	NaN	NaN	NaN	NaN	NaN
2013-01-03	NaN	NaN	NaN	NaN	NaN	NaN
2013-01-04	0.012545	0.012545	0.018182	0.019351	0.017219	-0.177591
2013-01-07	0.006321	0.006321	0.003237	0.000182	0.005992	-0.405606
2013-01-08	-0.001257	-0.001257	-0.005000	-0.004339	0.000900	-0.118958
...	...	...	...	...	...	...
2023-06-20	-0.001085	-0.001085	0.004902	0.002384	0.005131	-0.146056
2023-06-21	0.000000	0.000000	-0.000488	-0.011037	-0.013822	-0.529496
2023-06-22	-0.004147	-0.004147	-0.008293	-0.012488	-0.004320	0.112433
2023-06-23	-0.034652	-0.034652	-0.026645	-0.028298	-0.009443	2.406368
2023-06-26	-0.024293	-0.024293	-0.029516	-0.021678	-0.030369	-0.794940

2638 rows × 6 columns

In [114]:
```
sb.pct_change(periods = 3)
```

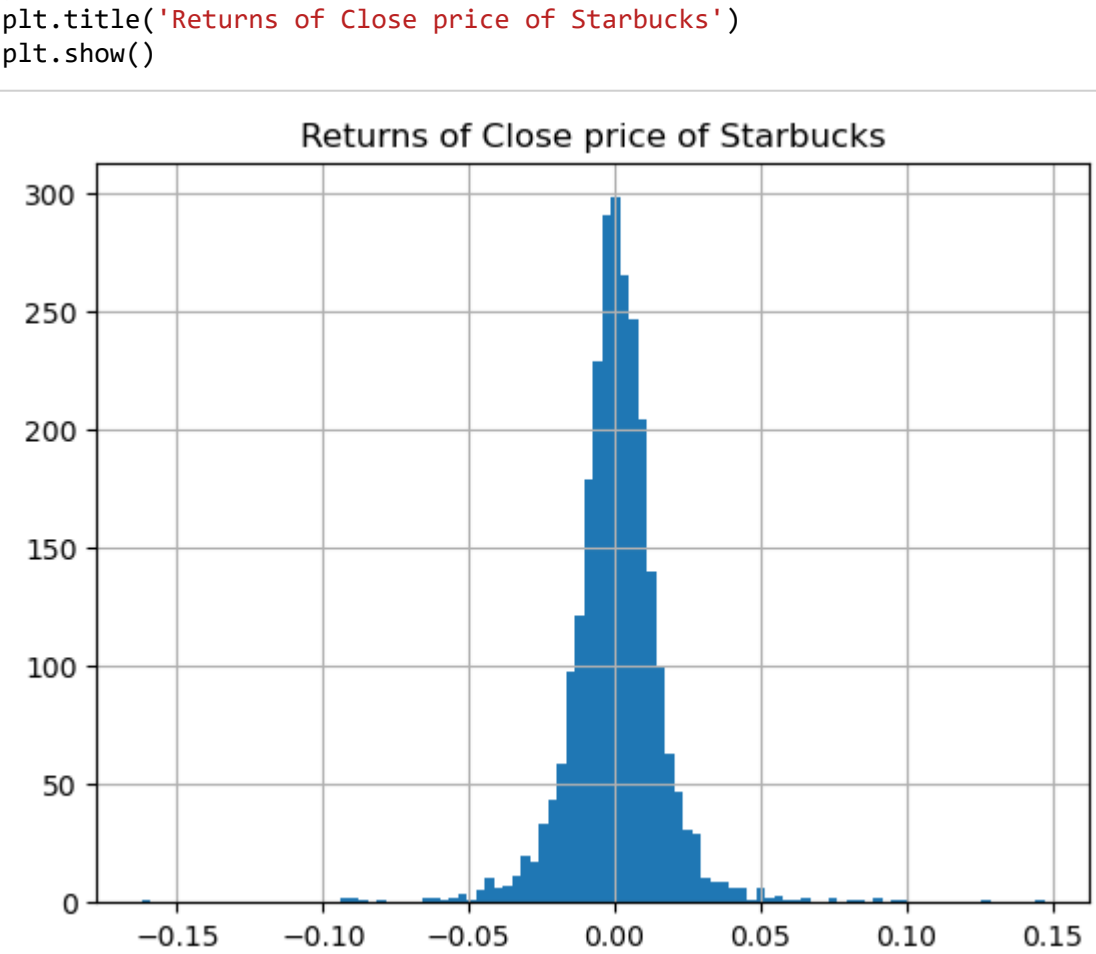
Out[114]:

Date	Adj Close	Close	High	Low	Open	Volume
2013-01-02	NaN	NaN	NaN	NaN	NaN	NaN
2013-01-03	NaN	NaN	NaN	NaN	NaN	NaN
2013-01-04	NaN	NaN	NaN	NaN	NaN	NaN
2013-01-07	0.013091	0.013091	0.014364	0.013822	0.014838	-0.342760
2013-01-08	0.004515	0.004515	0.001978	0.001273	0.009261	-0.344708
...	...	...	...	...	...	...
2023-06-20	0.006060	0.006060	0.011148	0.017240	0.023102	-0.406606
2023-06-21	0.004833	0.004833	0.004608	-0.002980	-0.007302	-0.145684
2023-06-22	-0.010013	-0.010013	-0.008487	-0.018132	-0.005784	-0.476824
2023-06-23	-0.028933	-0.028933	-0.026929	-0.033499	-0.021696	2.407853
2023-06-26	-0.034063	-0.034063	-0.037283	-0.028697	-0.022465	-0.771984

2638 rows × 6 columns

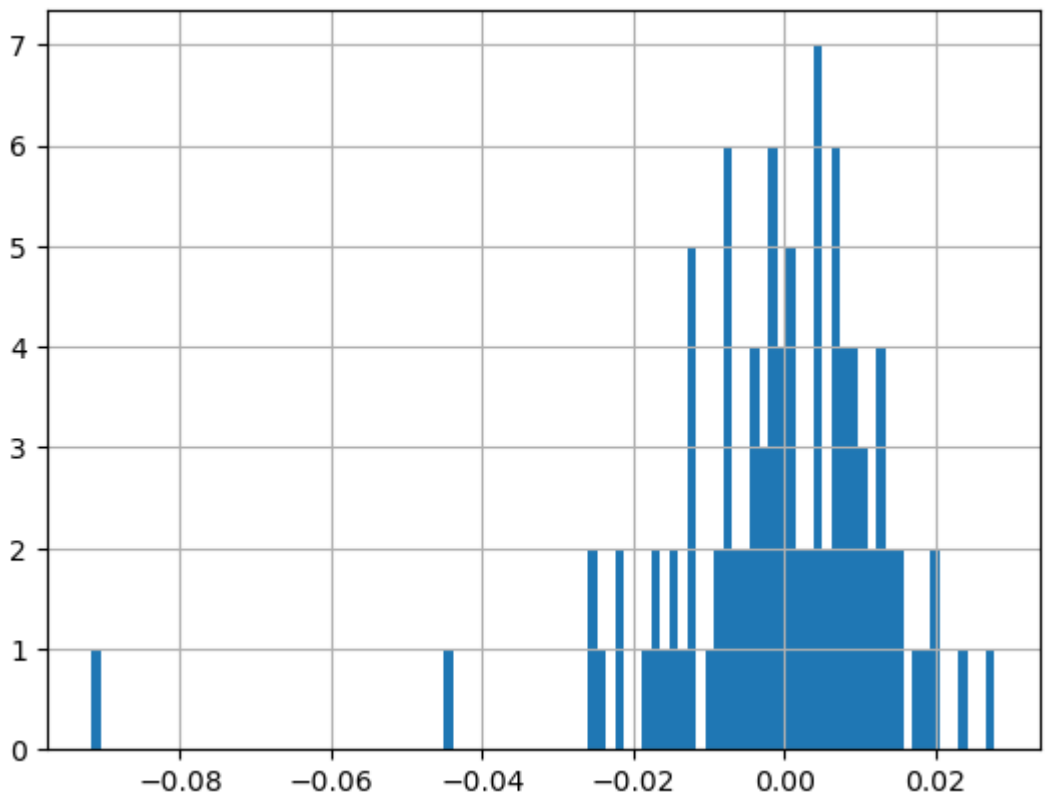
Plotting return

In [115]:
```
(sb
.pct_change()
.Close
.hist(bins=100)
)
plt.title('Returns of Close price of Starbucks')
plt.show()
```



In [116]:
```
# For last 100 entries:
(sb
.pct_change()
.Close
.iloc[-100:]
.hist(bins=100)
)
```

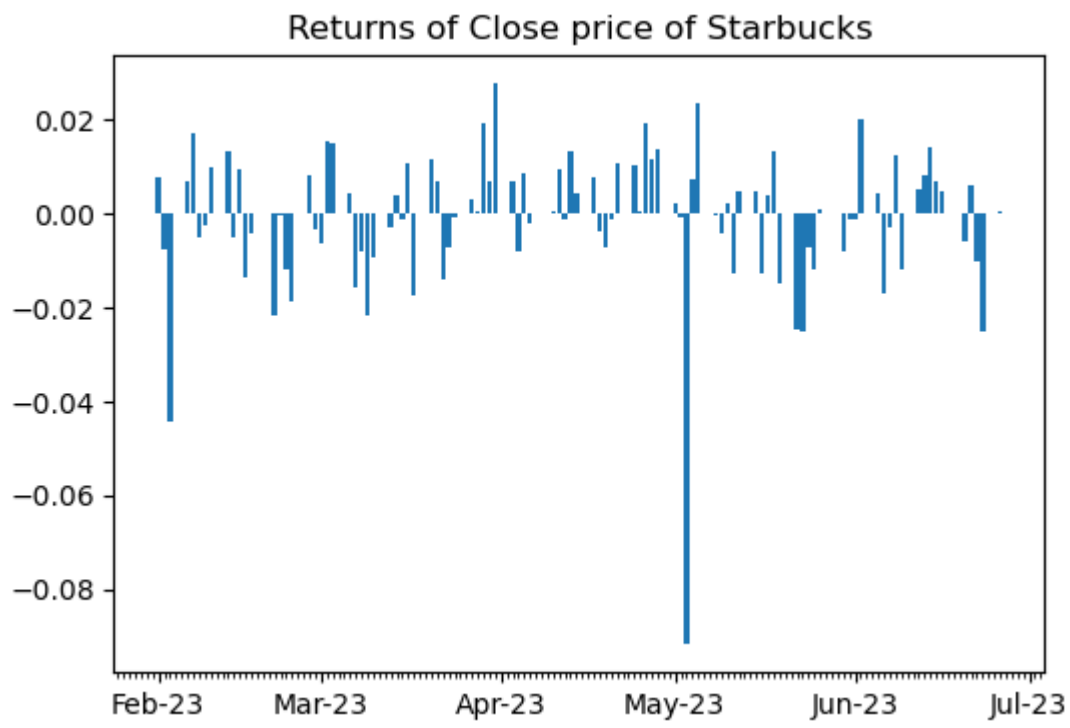
Out[116]: <Axes: >



In [ ]:

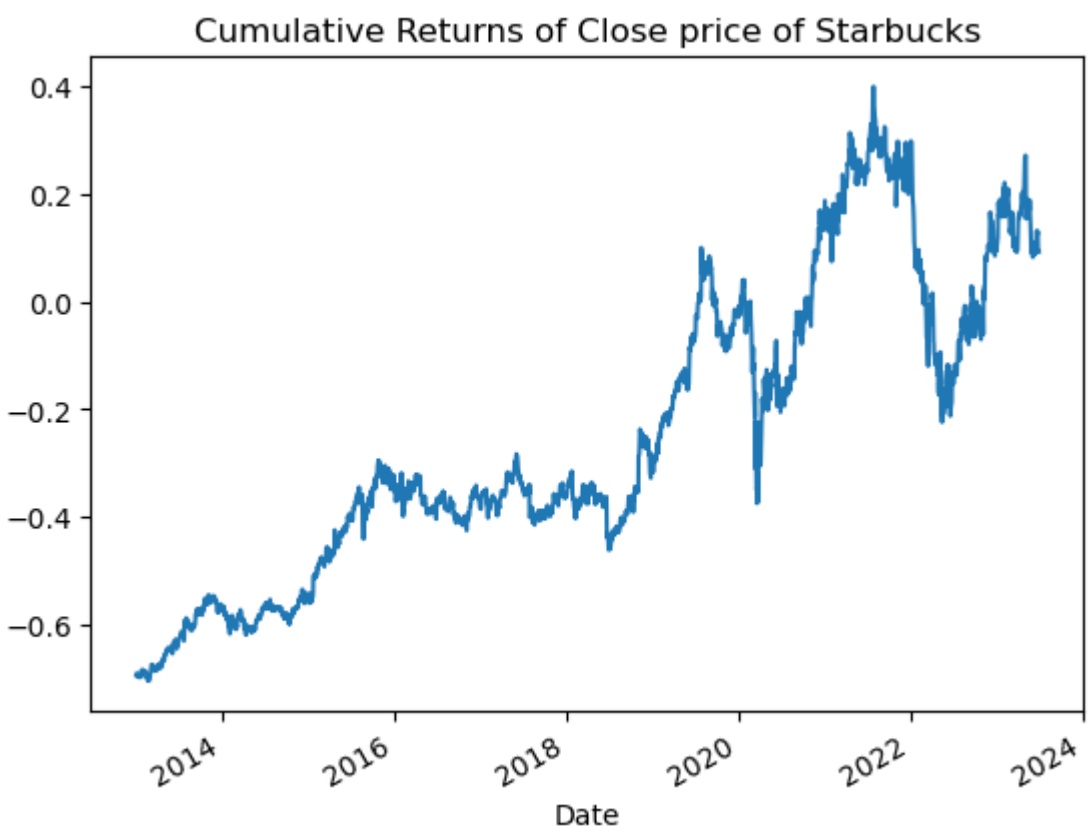
Fix the barplot

In [117]:
```
fig, ax = plt.subplots (figsize = (6,4))
_=(sb
.pct_change()
.Close
.iloc[-100:]
.pipe(bar_index, ax)
)
plt.title('Returns of Close price of Starbucks')
plt.show()
```



Cumulative Return

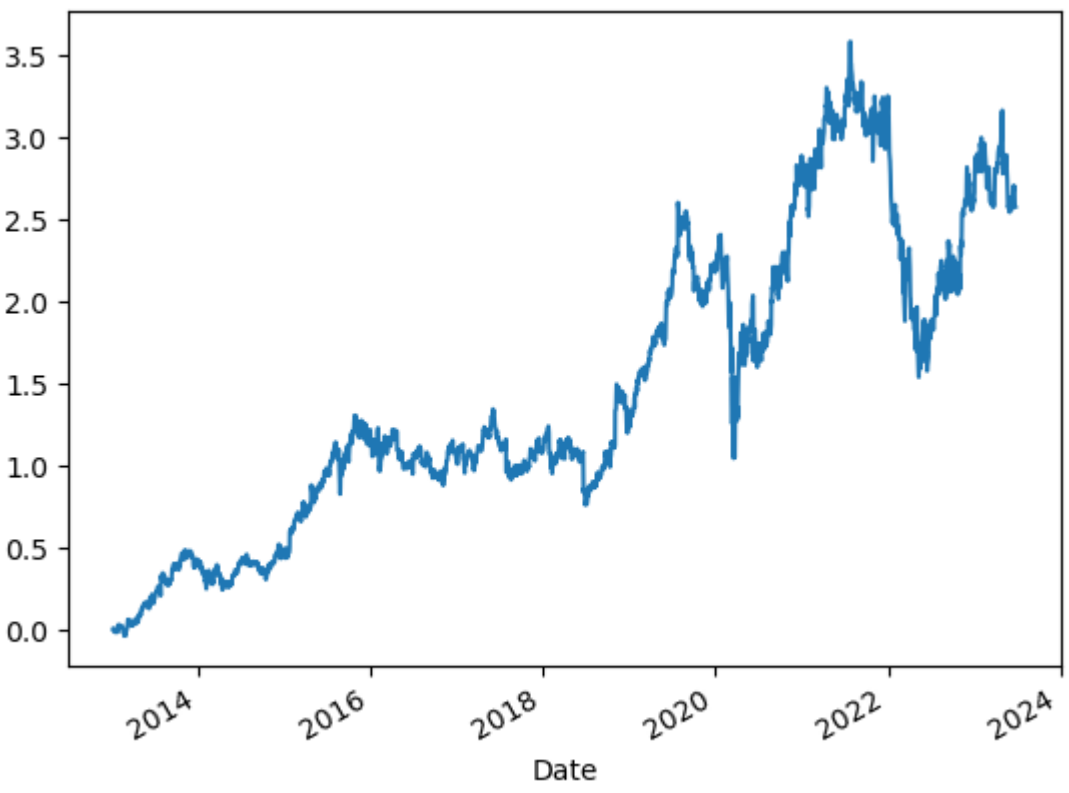
In [118]:
```
(sb
 .Close
 .sub(mc.Close[0])
 .div(mc.Close[0])
 .plot())
plt.title('Cumulative Returns of Close price of Starbucks')
plt.show()
```



- alternate method:

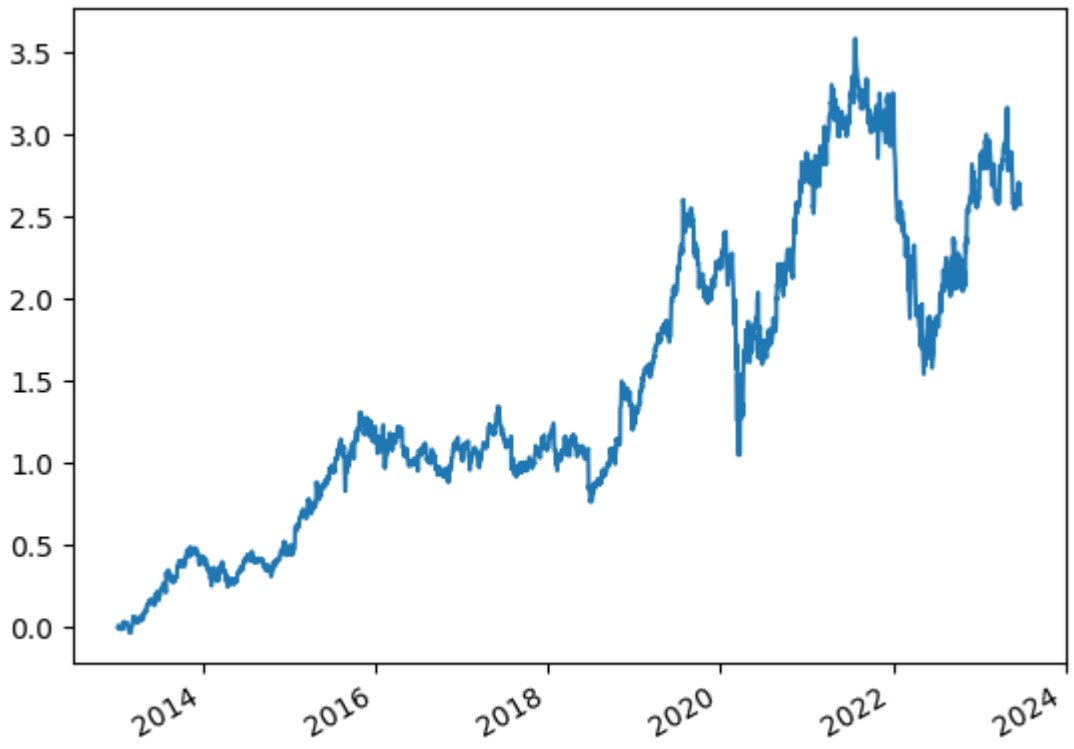
In [119]:
```
(sb
 .Close
 .pct_change()
 .add(1)
 .cumprod()
 .sub(1)
 .plot())
```

Out[119]: <Axes: xlabel='Date'>



In [120]:
```
(sb
    .pipe(cum_returns, 'Close')
    .plot()
)
```

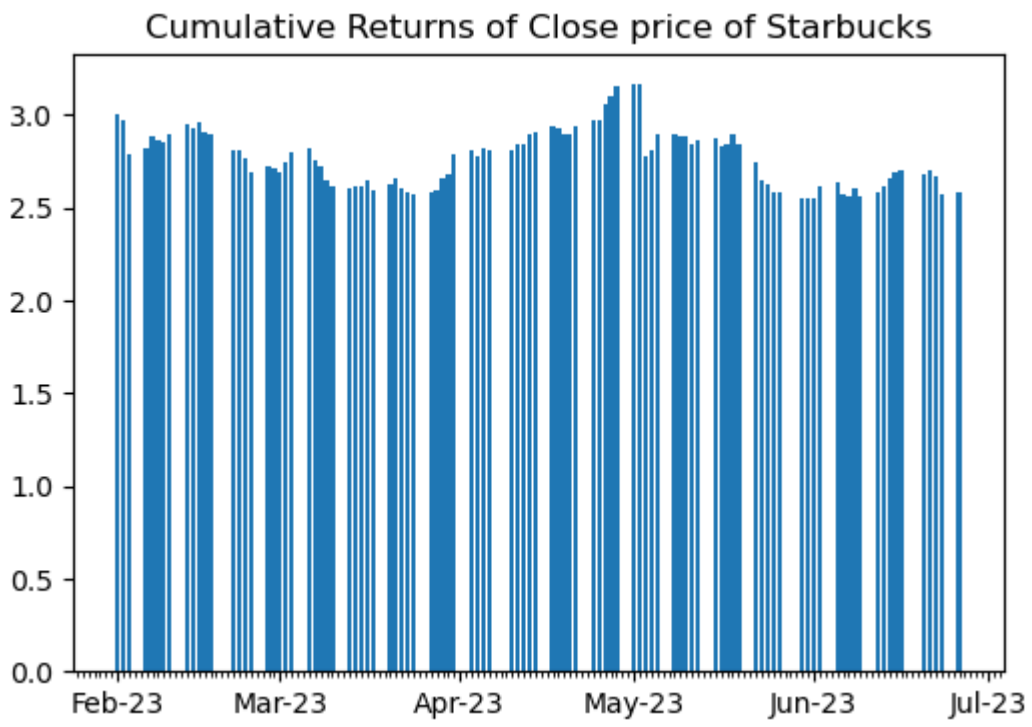
Out[120]: <Axes: xlabel='Date'>



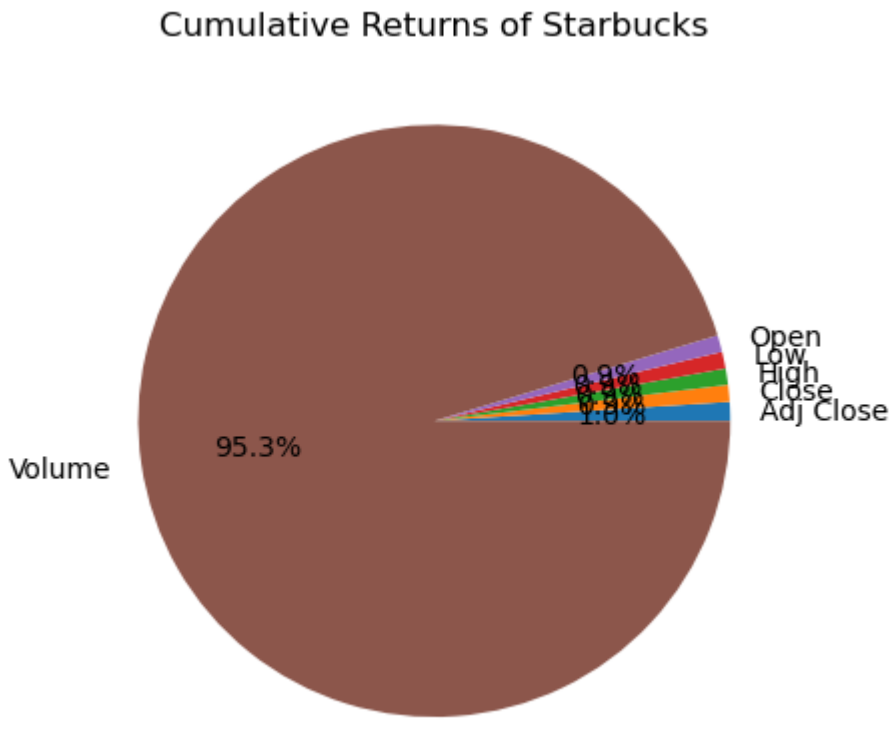
In [121]:
```
(lambda data : get_returns(data, 'Close'))(sb)
```

Out[121]:
```
Date
2013-01-02    0.000000
2013-01-03    0.006727
2013-01-04    0.012545
2013-01-07    0.013091
2013-01-08    0.011273
                ...
2023-06-20    2.682545
2023-06-21    2.704364
2023-06-22    2.667273
2023-06-23    2.576000
2023-06-26    2.578182
Name: Close, Length: 2638, dtype: float64
```

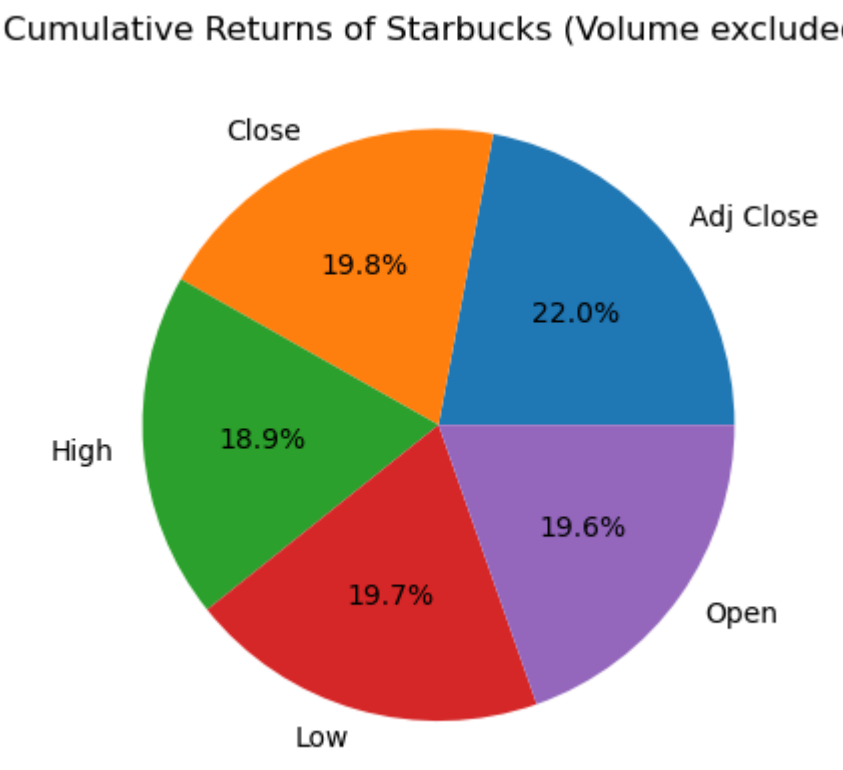
In [122]:
```
# craeting a new column cum_return
(sb
.assign(cum_returns = lambda data: cum_returns(data, 'Close'))
)

fig, ax = plt.subplots (figsize = (6,4))

_=(sb
.pipe(cum_returns, 'Close')
.iloc[-100:]
.pipe(bar_index, ax)
  )
plt.title('Cumulative Returns of Close price of Starbucks')
plt.show()
```



In [123]:
```
percentage_change = sb.pct_change().mean()
plt.pie(percentage_change, labels=percentage_change.index, autopct='%1.1f%%')
plt.title('Cumulative Returns of Starbucks')
plt.show()
```



In [124]:
```
percentage_change = sb.iloc[:, :-1].pct_change().mean()
plt.pie(percentage_change, labels=percentage_change.index, autopct='%1.1f%%')

plt.title('Cumulative Returns of Starbucks (Volume excluded)')
plt.show()
```



Volatility

In [125]:
```
#Average closing value of Starbucks
(sb
.Close
.mean()
)
```

Out[125]: 67.74605192281334

In [126]:
```
(sb
.Close
.std()
)
```

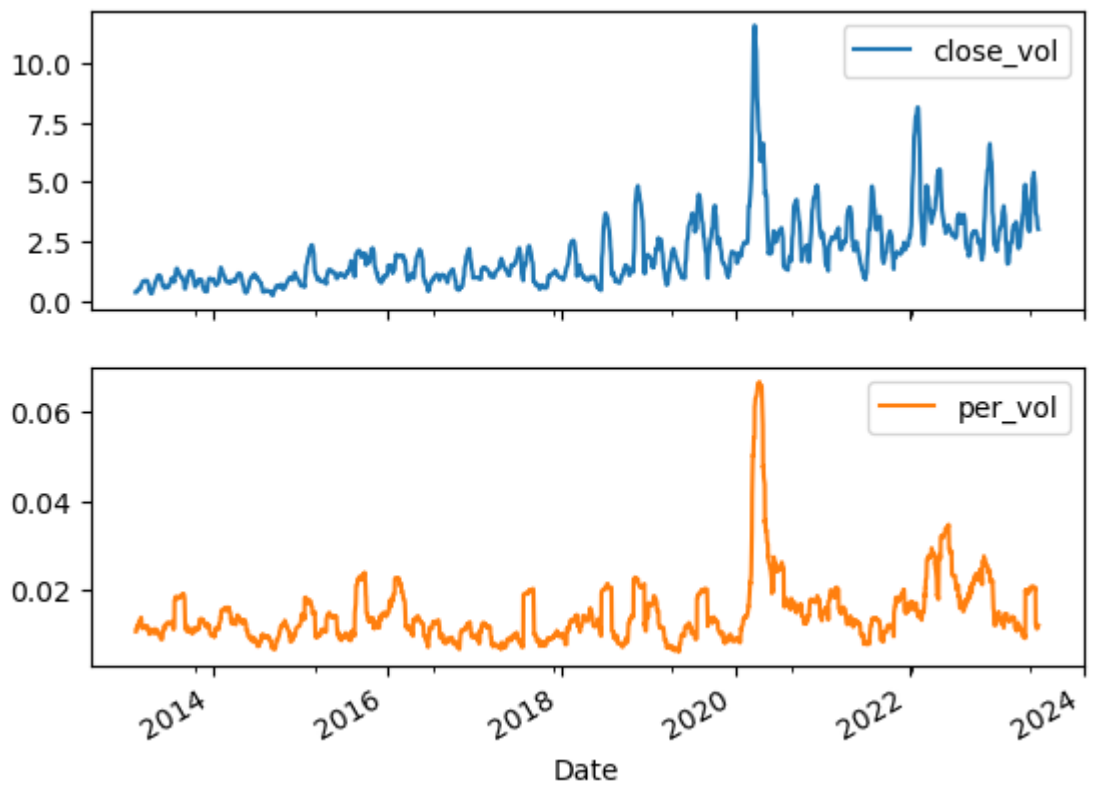
Out[126]: 24.76638198669708

In [127]:
```
(sb
.assign(pct_change_close = sb.Close.pct_change())
.pct_change_close
.std()
)
```

Out[127]: 0.016233576766246448

In [128]:
```
(sb
.assign(close_vol = sb.rolling(30).Close.std(),
 per_vol = sb.Close.pct_change().rolling(30).std())
 .iloc[:, -2:]
 .plot(subplots= True)
)
```

Out[128]:
```
array([<Axes: xlabel='Date'>, <Axes: xlabel='Date'>], dtype=object)
```



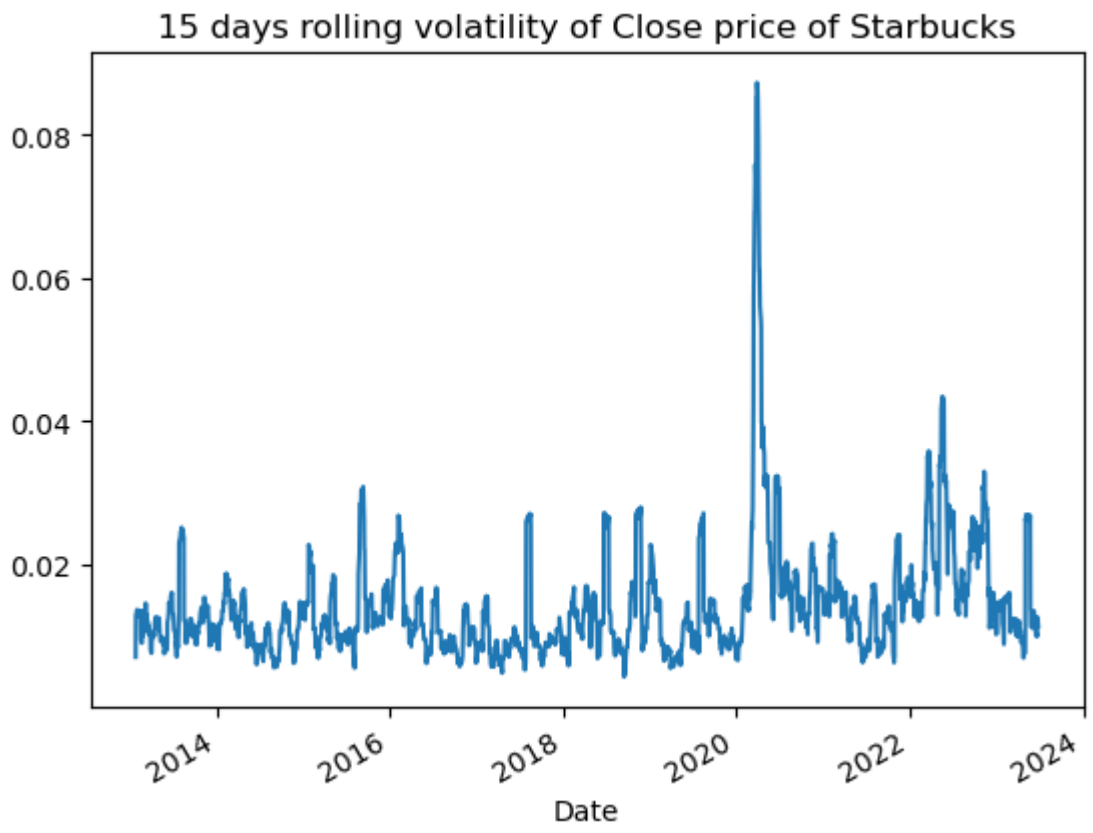
In [129]:
```
#15 Days volatility
(sb
.assign(pct_change_close = sb.Close.pct_change())
 .resample('15D')
.std()
)
```

Out[129]:

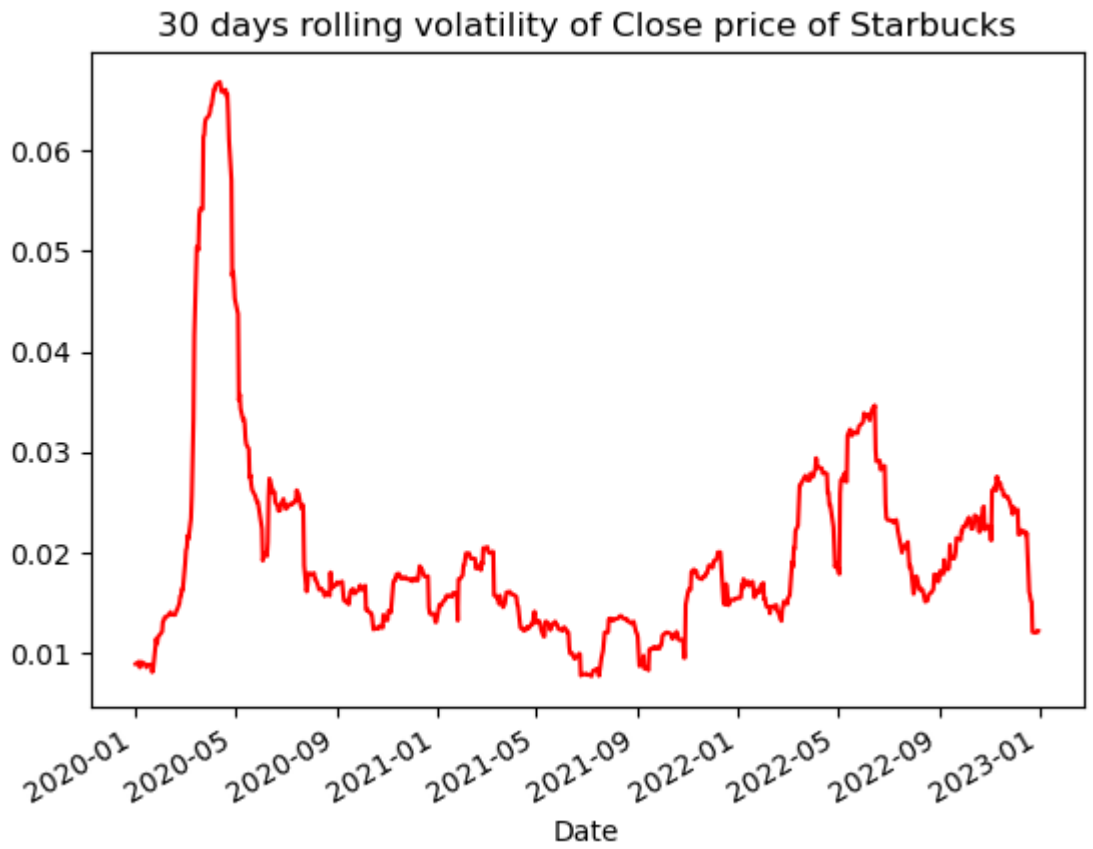
Date	Adj Close	Close	High	Low	Open	Volume	pct_change_close
2013-01-02	0.214345	0.258148	0.249172	0.253587	0.261506	2.561484e+06	0.007590
2013-01-17	0.344251	0.414600	0.476272	0.424578	0.408961	6.840298e+06	0.015015
2013-02-01	0.251316	0.316326	0.203500	0.302819	0.179862	3.302632e+06	0.009752
2013-02-16	0.297467	0.356914	0.264474	0.312698	0.321637	1.540294e+06	0.014122
2013-03-03	0.416207	0.499384	0.539166	0.582249	0.649783	3.165683e+06	0.010836
...	...	...	...	...	...	...	...
2023-04-25	3.970207	3.989976	3.375142	3.668166	3.371369	5.416460e+06	0.031185
2023-05-10	2.559229	2.600762	2.245084	2.316734	2.014826	1.127770e+06	0.012826
2023-05-25	0.825511	0.825511	0.872937	0.854674	0.802815	1.330124e+06	0.011022
2023-06-09	1.532499	1.532499	1.577239	1.494888	1.472604	4.287329e+06	0.011988
2023-06-24	NaN	NaN	NaN	NaN	NaN	NaN	NaN

256 rows × 7 columns

In [130]:
```
#15 Days  rolling volatility
(sb
.assign(pct_change = sb.Close.pct_change())
.rolling (window = 15, min_periods = 15)
.std()
['pct_change']
.plot()
)
plt.title('15 days rolling volatility of Close price of Starbucks')
plt.show()
```



In [131]:
```
#30 Days volatility for 2020- 2022
(sb
 .assign(pct_change_30 = sb.Close.pct_change())
 .rolling (window = 30, min_periods = 30)
 ['pct_change_30']
 .std()
 .loc['jan 2020' : 'dec 2022']
 .plot(color = 'r')
)
plt.title('30 days rolling volatility of Close price of Starbucks')
plt.show()
```



Rolling Windows

Simple Moving Average

Shifting method and built - in method

In [132]:
```
(sb
 .assign(s1 = sb.Close.shift(1),
         s2 = sb.Close.shift(2),
        ))
```

Out[132]:

Date	Adj Close	Close	High	Low	Open	Volume	s1	s2
2013-01-02	22.833767	27.500000	27.500000	27.129999	27.295000	13267600	NaN	NaN
2013-01-03	22.987379	27.684999	27.805000	27.500000	27.535000	14670400	27.500000	NaN
2013-01-04	23.120226	27.844999	28.000000	27.655001	27.764999	10911400	27.684999	27.500000
2013-01-07	23.132690	27.860001	27.895000	27.504999	27.700001	8720000	27.844999	27.684999
2013-01-08	23.091167	27.809999	27.860001	27.535000	27.790001	9613400	27.860001	27.844999
...	...	...	...	...	...	...	...	...
2023-06-20	101.269997	101.269997	102.489998	100.900002	101.860001	5506400	101.870003	101.379997
2023-06-21	101.870003	101.870003	102.459999	100.360001	100.599998	5508800	101.269997	101.870003
2023-06-22	100.849998	100.849998	101.639999	99.639999	101.419998	6125500	101.870003	101.269997
2023-06-23	98.339996	98.339996	99.730003	97.519997	99.650002	18765000	100.849998	101.870003
2023-06-26	98.400002	98.400002	98.639999	97.480003	98.339996	1256095	98.339996	100.849998

2638 rows × 8 columns

In [133]:
```
(sb
 .assign(s1 = sb['Close'].shift(1),
         s2 = sb['Close'].shift(2),
         ma3 = lambda data_ : data_.loc[:,['Close', 's1','s2']].mean(axis ='columns'),  #manually
         ma3_builtin = sb['Close'].rolling(3).mean()    #builtin method
        )
)
```

Out[133]:

Date	Adj Close	Close	High	Low	Open	Volume	s1	s2	ma3	ma3_builtin
2013-01-02	22.833767	27.500000	27.500000	27.129999	27.295000	13267600	NaN	NaN	27.500000	NaN
2013-01-03	22.987379	27.684999	27.805000	27.500000	27.535000	14670400	27.500000	NaN	27.592500	NaN
2013-01-04	23.120226	27.844999	28.000000	27.655001	27.764999	10911400	27.684999	27.500000	27.676666	27.676666
2013-01-07	23.132690	27.860001	27.895000	27.504999	27.700001	8720000	27.844999	27.684999	27.796666	27.796666
2013-01-08	23.091167	27.809999	27.860001	27.535000	27.790001	9613400	27.860001	27.844999	27.838333	27.838333
...	...	...	...	...	...	...	...	...	...	...
2023-06-20	101.269997	101.269997	102.489998	100.900002	101.860001	5506400	101.870003	101.379997	101.506666	101.506666
2023-06-21	101.870003	101.870003	102.459999	100.360001	100.599998	5508800	101.269997	101.870003	101.670001	101.670001
2023-06-22	100.849998	100.849998	101.639999	99.639999	101.419998	6125500	101.870003	101.269997	101.329999	101.329999
2023-06-23	98.339996	98.339996	99.730003	97.519997	99.650002	18765000	100.849998	101.870003	100.353333	100.353333
2023-06-26	98.400002	98.400002	98.639999	97.480003	98.339996	1256095	98.339996	100.849998	99.196665	99.196665

2638 rows × 10 columns

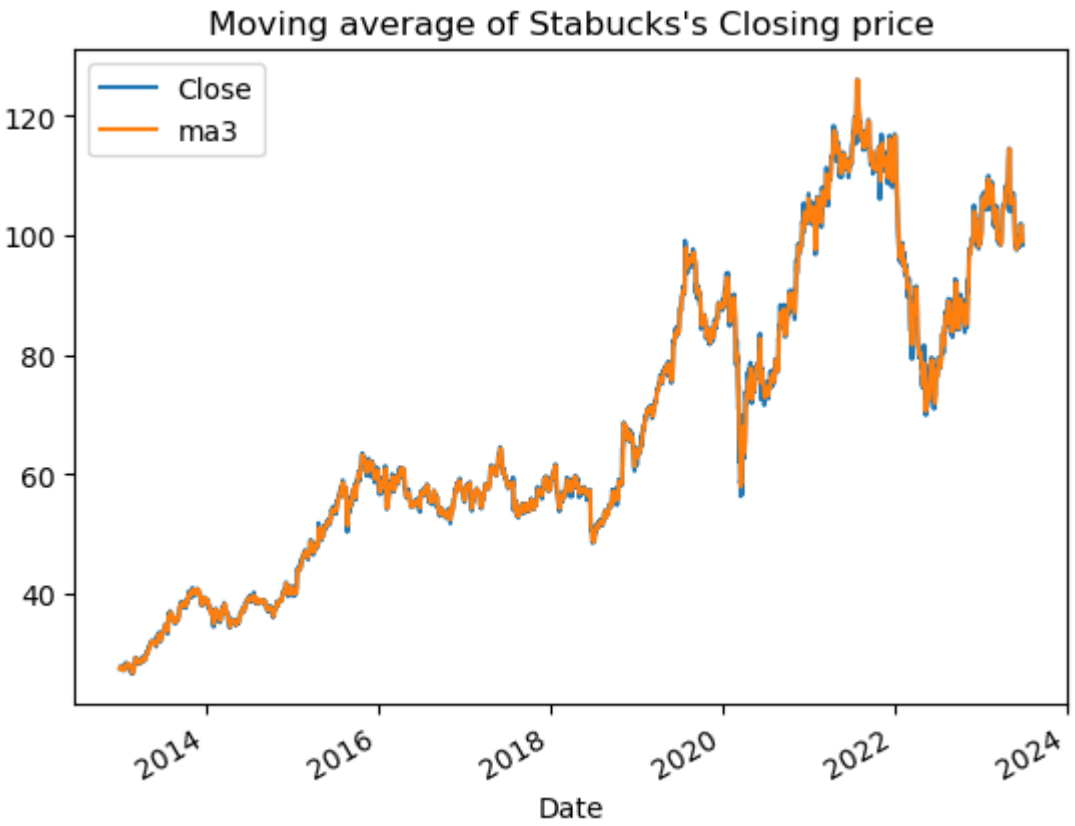
In [134]:
```
(sb
 .assign(s1 = sb['Close'].shift(1),
         s2 = sb['Close'].shift(2),
         ma3 = lambda data_ : data_.loc[:,['Close', 's1','s2']].mean(axis ='columns'),  #manually
         ma3_builtin = sb['Close'].rolling(3).mean()    #builtin method

        )
  [['Close', 'ma3']]
)
```

Out[134]:

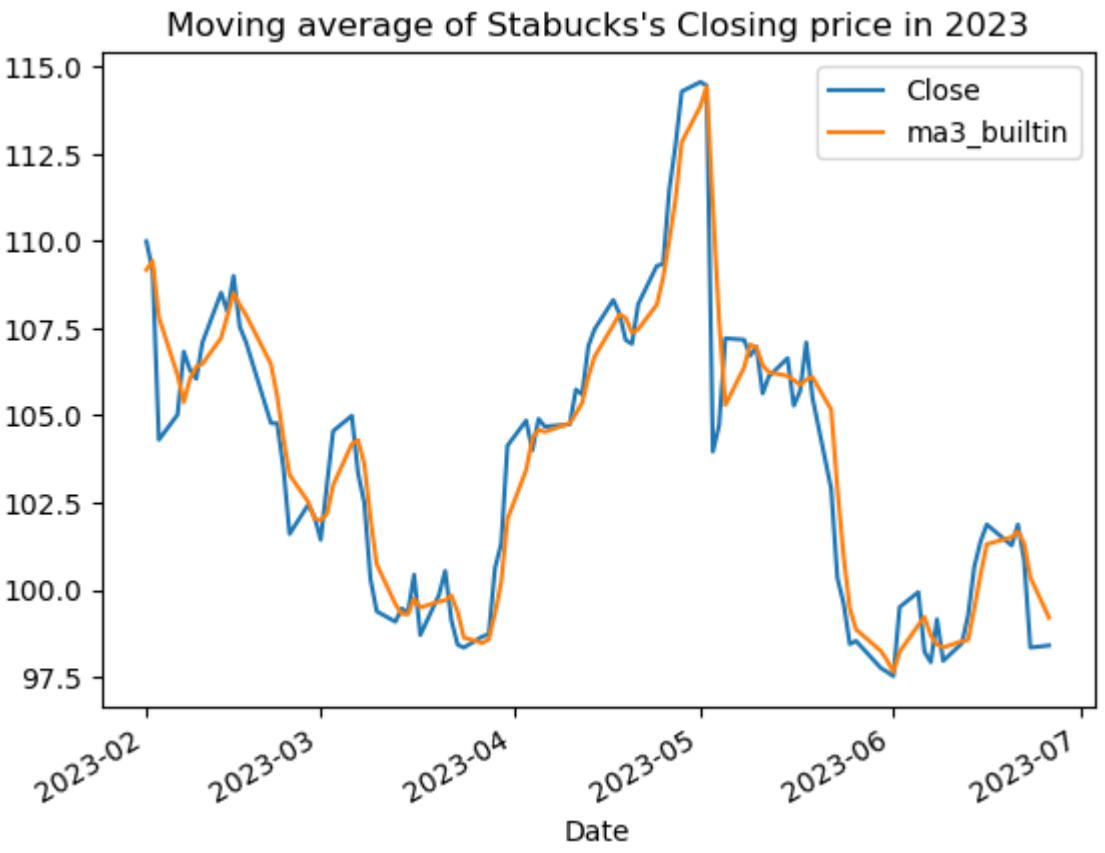
Date	Close	ma3
2013-01-02	27.500000	27.500000
2013-01-03	27.684999	27.592500
2013-01-04	27.844999	27.676666
2013-01-07	27.860001	27.796666
2013-01-08	27.809999	27.838333
...	...	...
2023-06-20	101.269997	101.506666
2023-06-21	101.870003	101.670001
2023-06-22	100.849998	101.329999
2023-06-23	98.339996	100.353333
2023-06-26	98.400002	99.196665

2638 rows × 2 columns

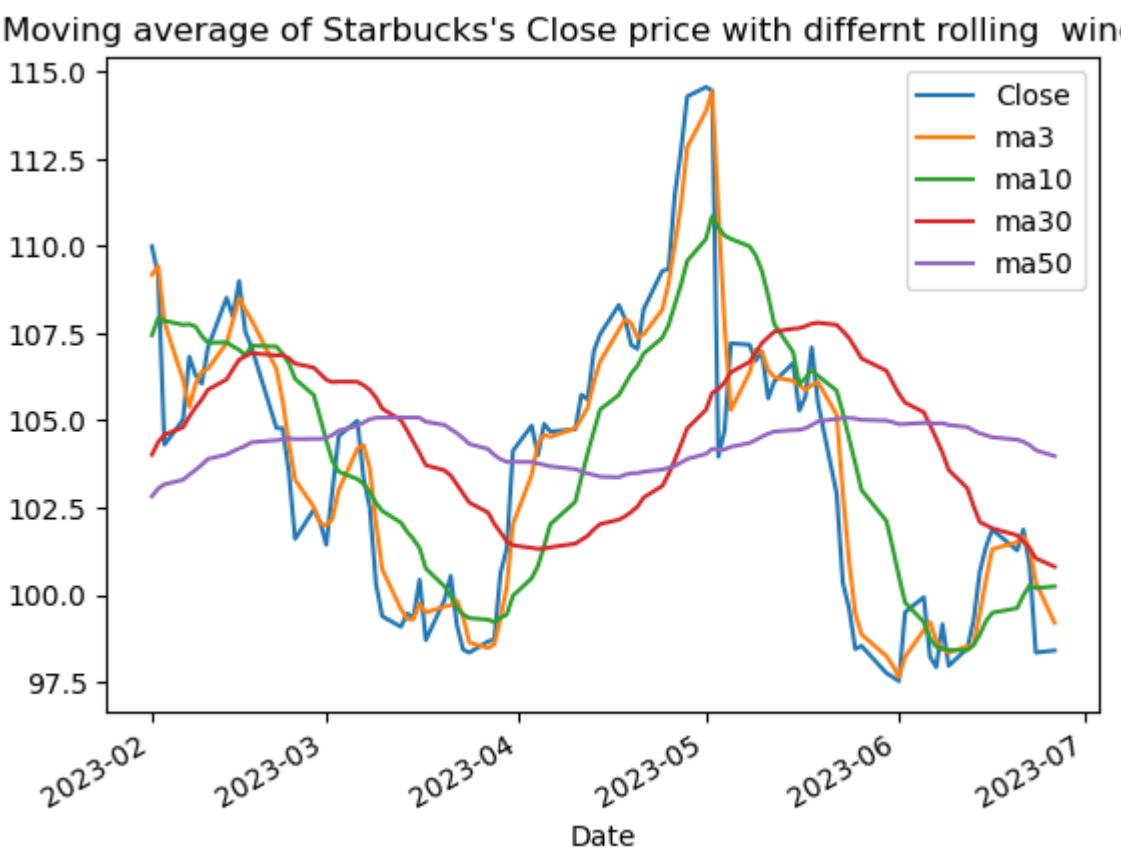
In [135]:
```
(sb
 .assign(s1 = sb['Close'].shift(1),
         s2 = sb['Close'].shift(2),
         ma3 = lambda data_ : data_.loc[:,['Close', 's1','s2']].mean(axis ='columns'),  #manually
         ma3_builtin = sb['Close'].rolling(3).mean()    #builtin method

        )
  [['Close', 'ma3']]
 .plot()
)
plt.title('Moving average of Stabucks\'s Closing price')
plt.show()
```



In [136]:
```
(sb
 .assign(s1 = sb['Close'].shift(1),
         s2 = sb['Close'].shift(2),
         ma3 = lambda data_ : data_.loc[:,['Close', 's1','s2']].mean(axis ='columns'),  #manually
         ma3_builtin = sb['Close'].rolling(3).mean()    #builtin method

        )
  [['Close' ,'ma3_builtin']]
 .iloc[-100:]
 .plot()
)
plt.title('Moving average of Stabucks\'s Closing price in 2023')
plt.show()
```

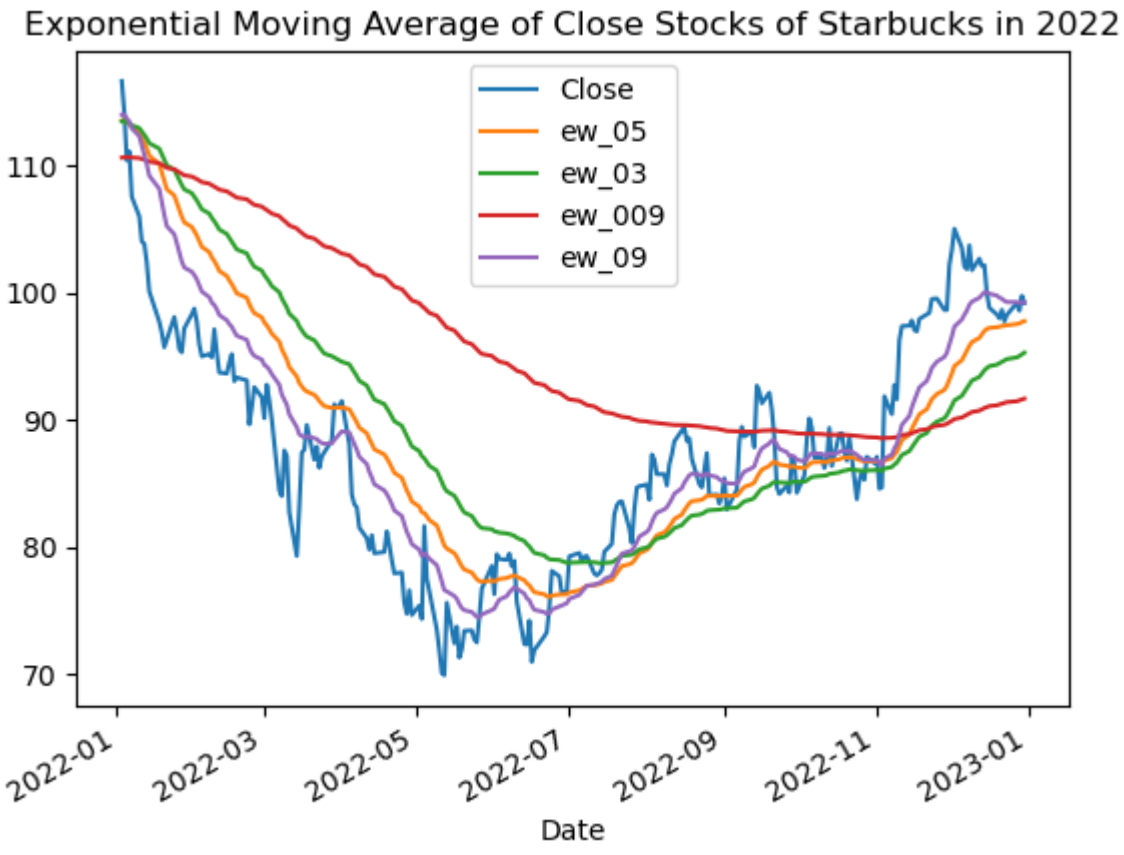


In [137]:
```
(sb
 .assign( ma3 = sb['Close'].rolling(3).mean(),
         ma10 = sb['Close'].rolling(10).mean(),
         ma30 = sb['Close'].rolling(30).mean(),
         ma50 = sb['Close'].rolling(50).mean(),

        )
[['Close', 'ma3','ma10','ma30', 'ma50']]
 .iloc[-100:]
 .plot()
)
plt.title('Moving average of Starbucks\'s Close price with differnt rolling  windows')
plt.show()
```



15 days rolling average better fits compared to 50days and 100 days. By doing a more number of days rolling average it smooths out more.

In [138]:
```
(sb
 .assign(
         ew_03 = sb['Close'].ewm(alpha = .03).mean(),
         ew_05 = sb['Close'].ewm(alpha = .05).mean(),
         ew_09 = sb['Close'].ewm(alpha = .09).mean(),
         ew_009 = sb['Close'].ewm(alpha =.009).mean(),


        )
[['Close','ew_05', 'ew_03','ew_009','ew_09']]
 .loc['jan 2022':'dec 2022']
 .plot()
)
plt.title('Exponential Moving Average of Close Stocks of Starbucks in 2022')
plt.show()
```



Technical

On-balance Volume (OBV)

In [139]:
```
OBV(sb)
```

Out[139]:

Date	Adj Close	Close	High	Low	Open	Volume	OBV
2013-01-02	22.833767	27.500000	27.500000	27.129999	27.295000	13267600	0.0
2013-01-03	22.987379	27.684999	27.805000	27.500000	27.535000	14670400	14670400.0
2013-01-04	23.120226	27.844999	28.000000	27.655001	27.764999	10911400	25581800.0
2013-01-07	23.132690	27.860001	27.895000	27.504999	27.700001	8720000	34301800.0
2013-01-08	23.091167	27.809999	27.860001	27.535000	27.790001	9613400	24688400.0
...	...	...	...	...	...	...	...
2023-06-20	101.269997	101.269997	102.489998	100.900002	101.860001	5506400	386867700.0
2023-06-21	101.870003	101.870003	102.459999	100.360001	100.599998	5508800	392376500.0
2023-06-22	100.849998	100.849998	101.639999	99.639999	101.419998	6125500	386251000.0
2023-06-23	98.339996	98.339996	99.730003	97.519997	99.650002	18765000	367486000.0
2023-06-26	98.400002	98.400002	98.639999	97.480003	98.339996	1256095	368742095.0

2638 rows × 7 columns

In [140]:
```
(sb
.assign(Vol =np.select([sb.Close > sb.Close.shift(1),
                        sb.Close == sb.Close.shift(1),
                        sb.Close <sb.Close.shift(1)],
                       [sb.Volume, 0, -sb.Volume]
                      ),
        obv = lambda data_: data_.Vol.cumsum(),
       )
)
```

Out[140]:

Date	Adj Close	Close	High	Low	Open	Volume	Vol	obv
2013-01-02	22.833767	27.500000	27.500000	27.129999	27.295000	13267600	0	0
2013-01-03	22.987379	27.684999	27.805000	27.500000	27.535000	14670400	14670400	14670400
2013-01-04	23.120226	27.844999	28.000000	27.655001	27.764999	10911400	10911400	25581800
2013-01-07	23.132690	27.860001	27.895000	27.504999	27.700001	8720000	8720000	34301800
2013-01-08	23.091167	27.809999	27.860001	27.535000	27.790001	9613400	-9613400	24688400
...	...	...	...	...	...	...	...	...
2023-06-20	101.269997	101.269997	102.489998	100.900002	101.860001	5506400	-5506400	386867700
2023-06-21	101.870003	101.870003	102.459999	100.360001	100.599998	5508800	5508800	392376500
2023-06-22	100.849998	100.849998	101.639999	99.639999	101.419998	6125500	-6125500	386251000
2023-06-23	98.339996	98.339996	99.730003	97.519997	99.650002	18765000	-18765000	367486000
2023-06-26	98.400002	98.400002	98.639999	97.480003	98.339996	1256095	1256095	368742095

2638 rows × 8 columns

In [141]:
```
(sb
.assign(obv=cal_obv)
)
```

Out[141]:

Date	Adj Close	Close	High	Low	Open	Volume	obv
2013-01-02	22.833767	27.500000	27.500000	27.129999	27.295000	13267600	0
2013-01-03	22.987379	27.684999	27.805000	27.500000	27.535000	14670400	14670400
2013-01-04	23.120226	27.844999	28.000000	27.655001	27.764999	10911400	25581800
2013-01-07	23.132690	27.860001	27.895000	27.504999	27.700001	8720000	34301800
2013-01-08	23.091167	27.809999	27.860001	27.535000	27.790001	9613400	24688400
...	...	...	...	...	...	...	...
2023-06-20	101.269997	101.269997	102.489998	100.900002	101.860001	5506400	386867700
2023-06-21	101.870003	101.870003	102.459999	100.360001	100.599998	5508800	392376500
2023-06-22	100.849998	100.849998	101.639999	99.639999	101.419998	6125500	386251000
2023-06-23	98.339996	98.339996	99.730003	97.519997	99.650002	18765000	367486000
2023-06-26	98.400002	98.400002	98.639999	97.480003	98.339996	1256095	368742095

2638 rows × 7 columns

Acumulation/ Distribution Indicator (A/D)

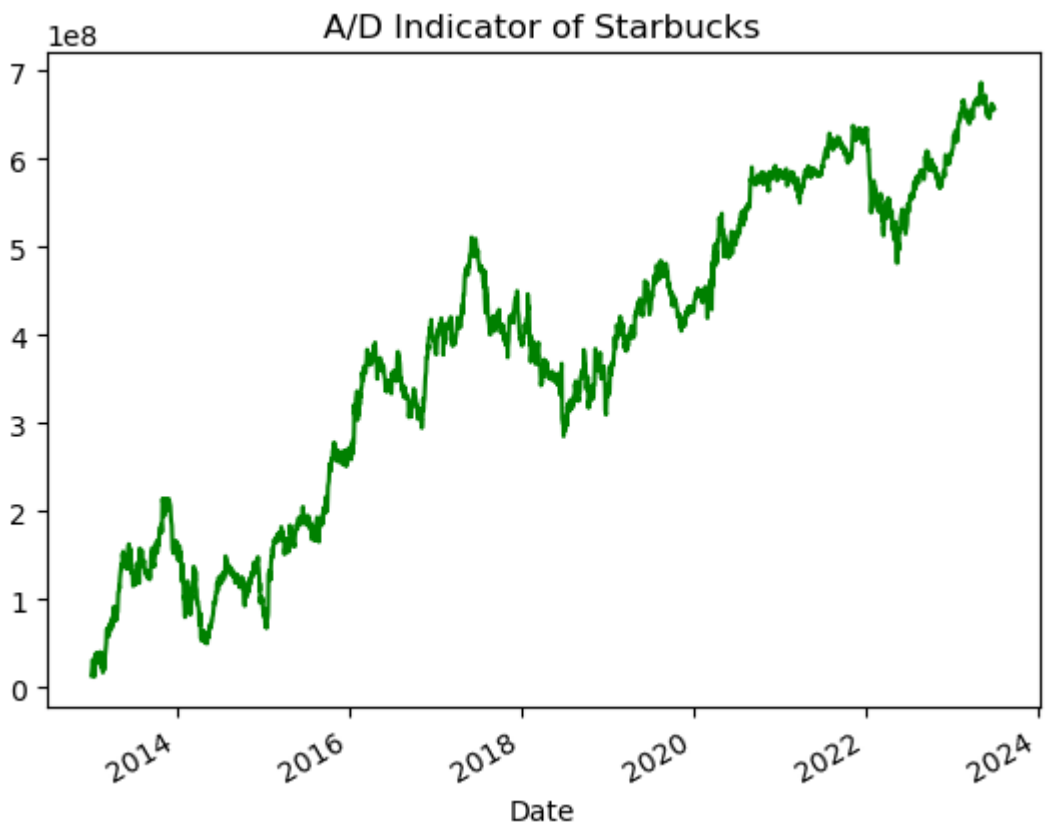
In [142]:
```
(sb
.assign(mfm =((sb.Close-mc.Low)+(sb.Close-mc.High))/(sb.High -mc.Low),
        mfv = lambda data_ : data_.mfm *data_.Volume,
       cmfv = lambda data_ : data_.mfv.cumsum())
)
```

Out[142]:

Date	Adj Close	Close	High	Low	Open	Volume	mfm	mfv	cmfv
2013-01-02	22.833767	27.500000	27.500000	27.129999	27.295000	13267600	2.016497	2.675407e+07	2.675407e+07
2013-01-03	22.987379	27.684999	27.805000	27.500000	27.535000	14670400	2.012509	2.952431e+07	5.627838e+07
2013-01-04	23.120226	27.844999	28.000000	27.655001	27.764999	10911400	2.023188	2.207581e+07	7.835420e+07
2013-01-07	23.132690	27.860001	27.895000	27.504999	27.700001	8720000	2.030478	1.770577e+07	9.605997e+07
2013-01-08	23.091167	27.809999	27.860001	27.535000	27.790001	9613400	2.017550	1.939551e+07	1.154555e+08
...	...	...	...	...	...	...	...	...	...
2023-06-20	101.269997	101.269997	102.489998	100.900002	101.860001	5506400	2.036490	1.121373e+07	4.772137e+10
2023-06-21	101.870003	101.870003	102.459999	100.360001	100.599998	5508800	2.018545	1.111976e+07	4.773249e+10
2023-06-22	100.849998	100.849998	101.639999	99.639999	101.419998	6125500	2.027070	1.241682e+07	4.774491e+10
2023-06-23	98.339996	98.339996	99.730003	97.519997	99.650002	18765000	2.029435	3.808235e+07	4.778299e+10
2023-06-26	98.400002	98.400002	98.639999	97.480003	98.339996	1256095	2.016770	2.533255e+06	4.778552e+10

2638 rows × 9 columns

In [143]:
```
(sb
.assign(ad=cal_ad)
.ad
.plot(color='g')
)
plt.title('A/D Indicator of Starbucks')
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: