Numpy - Applications in Finance

In [1]:
```
import numpy as np

np.set_printoptions(suppress=True)
```

In [2]:
```
datafile = 'http://people.bu.edu/kalathur/datasets/AAPL.csv'
```

In [3]:
```
# Import closing price and volume from the file (columns 5 and 6)
# First column is column 0

c, v = np.loadtxt(datafile, delimiter=',', skiprows=1, usecols=(5,6), unpack=True)

c = np.round(c, 2)
```

In [4]:
```
c[:5]
```

Out[4]:
```
array([156.05, 140.51, 146.5 , 146.18, 148.96])
```

In [5]:
```
v[:5]
```

Out[5]:
```
array([37039700., 91312200., 58607100., 54777800., 41025300.])
```

In [6]:
```
# number of values in the dataset

len(c)
```

Out[6]:
```
178
```

Volume Weighted Average Price (VWAP)

In [7]:
```
vwap = np.average(c, weights=v)
print("VWAP =", vwap)
```

```
VWAP = 187.04154812148687
```

In [8]:
```
# Arithmetic mean

print("mean =", np.mean(c))
```

```
mean = 188.89511235955058
```

Value Range

In [9]:
```
# Import daily high and low price from the file (columns 2 and 3)

h, l = np.loadtxt(datafile, delimiter=',', skiprows=1, usecols=(2,3), unpack=True)

h = np.round(h, 2)
l = np.round(l, 2)
```

In [10]:
```
h[:5]
```

Out[10]:
```
array([158.85, 145.72, 148.55, 148.83, 151.82])
```

In [11]:
```
l[:5]
```

Out[11]:
```
array([154.23, 142.  , 143.8 , 145.9 , 148.52])
```

In [12]:
```
print("highest daily high =", np.max(h))
print("lowest  daily low =", np.min(l))
```

```
highest daily high = 226.42
lowest  daily low = 142.0
```

In [13]:
```
# Spread of data

print("Spread high price", np.ptp(h))
print("Spread low price", np.ptp(l))
```

```
Spread high price 80.69999999999999
Spread low price 80.86000000000001
```

In [14]:
```
print("Spread high price", np.max(h) - np.min(h))
print("Spread low price", np.max(l) - np.min(l))
```

```
Spread high price 80.69999999999999
Spread low price 80.86000000000001
```

Statistics

In [15]:
```
print("median =", np.median(c))
```

```
median = 193.965
```

In [16]:
```
print("variance =", np.var(c))
```

```
variance = 354.60887891995964
```

In [17]:
```
print("Standard deviation =", np.std(c))
```

```
Standard deviation = 18.831061545222553
```

Simple Returns

- differences between consecutive values / value of the previous day

In [18]:
```
returns = np.diff( c ) / c[ : -1]
returns[:5]
```

Out[18]:
```
array([-0.09958347,  0.04263042, -0.0021843 ,  0.01901765,  0.01698443])
```

In [19]:
```
np.max(returns), np.min(returns)
```

Out[19]:
```
(0.06830225711481844, -0.09958346683755219)
```

In [20]:
```
print("Standard deviation =", np.std(returns))
```

```
Standard deviation = 0.018182188529057272
```

Log Returns

- log of all values and calculate differences between them
- log(a) - log(b) = log(a/b)
- measure rate of change
- input should not have zeros or negative numbers

In [21]:
```
logreturns = np.diff( np.log(c) )
```

In [22]:
```
logreturns[:5]
```

Out[22]:
```
array([-0.10489781,  0.04174677, -0.00218669,  0.01883907,  0.0168418 ])
```

In [23]:
```
# Alternatively

logreturns = np.log(c[1:]/c[:-1])

logreturns[:5]
```

Out[23]:
```
array([-0.10489781,  0.04174677, -0.00218669,  0.01883907,  0.0168418 ])
```

Positive Returns

In [24]:
```
pos_ret_indices = np.where(returns > 0)
print("Indices with positive returns\n", pos_ret_indices)
```

```
Indices with positive returns
 (array([  1,   3,   4,   5,   8,   9,  10,  11,  13,  15,  18,  19,  20,
        21,  22,  23,  25,  27,  29,  31,  32,  34,  35,  36,  37,  39,
        40,  44,  45,  46,  47,  48,  49,  50,  52,  53,  57,  58,  59,
        60,  61,  62,  63,  64,  65,  67,  70,  71,  72,  73,  74,  75,
        79,  81,  83,  86,  90,  91,  95, 101, 104, 105, 106, 107, 108,
       109, 113, 114, 116, 120, 123, 124, 125, 128, 129, 131, 132, 135,
       137, 138, 141, 142, 144, 148, 149, 150, 153, 156, 157, 158, 159,
       162, 164, 165, 168, 169, 171, 172, 173, 176]),)
```

In [25]:
```
np.where(logreturns > 0)
```

Out[25]:
```
(array([  1,   3,   4,   5,   8,   9,  10,  11,  13,  15,  18,  19,  20,
        21,  22,  23,  25,  27,  29,  31,  32,  34,  35,  36,  37,  39,
        40,  44,  45,  46,  47,  48,  49,  50,  52,  53,  57,  58,  59,
        60,  61,  62,  63,  64,  65,  67,  70,  71,  72,  73,  74,  75,
        79,  81,  83,  86,  90,  91,  95, 101, 104, 105, 106, 107, 108,
       109, 113, 114, 116, 120, 123, 124, 125, 128, 129, 131, 132, 135,
       137, 138, 141, 142, 144, 148, 149, 150, 153, 156, 157, 158, 159,
       162, 164, 165, 168, 169, 171, 172, 173, 176]),)
```

Volatility

- measures price variation
- annualized volatility is equal to the standard deviation of the log returns as a ratio of its mean, divided by one over the square root of the number of business days in a year

In [26]:
```
annual_volatility = np.std(logreturns)/np.mean(logreturns)
annual_volatility = annual_volatility / np.sqrt(1./252.)

print("Annual volatility", annual_volatility)
```

```
Annual volatility 150.0488985319145
```

In [27]:
```
print("Monthly volatility", annual_volatility * np.sqrt(1./12.))
```

```
Monthly volatility 43.31538597950384
```

Dealing with Dates

In [28]:
```
from datetime import datetime
import calendar
```

Monday 0, Tuesday 1, Wednesday 2, Thursday 3, Friday 4, Saturday 5, Sunday 6

In [29]:
```
list(calendar.day_name)
```

Out[29]:
```
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
```

In [30]:
```
def datestr2num(s):
    return datetime.strptime(s, "%Y-%m-%d").date().weekday()
```

In [31]:
```
datestr2num('2019-9-30')
```

Out[31]:
```
0
```

In [32]:
```
datestr2num('2019-10-1')
```

Out[32]:
```
1
```

In [33]:
```
# Import Date and closing price from the file (columns 0 and 5)

dates, close = np.loadtxt(datafile, delimiter=',',
                          skiprows=1, usecols=(0,5),
                          converters={0: datestr2num},
                          encoding='utf-8',
                          unpack=True)

close = np.round(close, 2)
```

In [34]:
```
close[:5]
```
Out[34]:
```
array([156.05, 140.51, 146.5 , 146.18, 148.96])
```

In [35]:
```
dates[:5]
```
Out[35]:
```
array([2., 3., 4., 0., 1.])
```

In [36]:
```
np.unique(dates)
```
Out[36]:
```
array([0., 1., 2., 3., 4.])
```

In [37]:
```
# averages based on day of week

averages = np.zeros(len(np.unique(dates)))

for i in range(5):
    indices = np.where(dates == i)
    prices  = np.take(close, indices)
    avg = np.mean(prices)
    avg = np.round(avg)
    print("Day", i, "Average", avg)
    averages[i] = avg
```
```
Day 0 Average 190.0
Day 1 Average 189.0
Day 2 Average 189.0
Day 3 Average 188.0
Day 4 Average 188.0
```

In [38]:
```
top = np.max(averages)
print("Highest average:", top)
print("Top day of the week index:", np.argmax(averages))
print("Top day of the week is", calendar.day_name[np.argmax(averages)])
print()

bottom = np.min(averages)
print("Lowest average:", bottom)
print("Bottom day of the week index:", np.argmin(averages))
print("Bottom day of the week is", calendar.day_name[np.argmin(averages)])
```

```
Highest average: 190.0
Top day of the week index: 0
Top day of the week is Monday

Lowest average: 188.0
Bottom day of the week index: 3
Bottom day of the week is Thursday
```

In [ ]:
```

```

Average True Range (ATR)

- provides degree of price volatility
- N-period smoothed moving average of the true range values
- Recommended 14 period smoothing
- Range of a day : $ (high-low) $
- True Range TR = $ max[(high-low), abs(high-close_{prev}), abs(low-close_{prev})] $
- $$ATR_{t} = \frac{ ATR_{t-1} * (N-1) + TR_{t} }{N}$$
- First ATR is the arithmetic mean of the first N TR values

Example: https://school.stockcharts.com/doku.php?id=technical_indicators:average_true_range_atr (https://school.stockcharts.com/doku.php?id=technical_indicators:average_true_range_atr)

		High	Low	Close	H - L	I H - Cp I	I L - Cp I	TR	ATR
	01-Apr-10	48.70	47.79	48.16	0.91			0.91	
	05-Apr-10	48.72	48.14	48.61	0.58	0.56	0.02	0.58	
	06-Apr-10	48.90	48.39	48.75	0.51	0.29	0.22	0.51	
	07-Apr-10	48.87	48.37	48.63	0.50	0.12	0.38	0.50	
	08-Apr-10	48.82	48.24	48.74	0.58	0.19	0.39	0.58	
	09-Apr-10	49.05	48.64	49.03	0.41	0.31	0.11	0.41	
	12-Apr-10	49.20	48.94	49.07	0.26	0.17	0.09	0.26	
	13-Apr-10	49.35	48.86	49.32	0.49	0.28	0.21	0.49	
	14-Apr-10	49.92	49.50	49.91	0.42	0.60	0.18	0.60	
	15-Apr-10	50.19	49.87	50.13	0.32	0.28	0.04	0.32	
	16-Apr-10	50.12	49.20	49.53	0.92	0.01	0.93	0.93	
	19-Apr-10	49.66	48.90	49.50	0.76	0.13	0.63	0.76	
	20-Apr-10	49.88	49.43	49.75	0.45	0.38	0.07	0.45	
	21-Apr-10	50.19	49.73	50.03	0.46	0.44	0.02	0.46	0.56
1	22-Apr-10	50.36	49.26	50.31	1.10	0.33	0.77	1.10	0.59
2	23-Apr-10	50.57	50.09	50.52	0.48	0.26	0.22	0.48	0.59
3	26-Apr-10	50.65	50.30	50.41	0.35	0.13	0.22	0.35	0.57
4	27-Apr-10	50.43	49.21	49.34	1.22	0.02	1.20	1.22	0.62
5	28-Apr-10	49.63	48.98	49.37	0.65	0.29	0.36	0.65	0.62
6	29-Apr-10	50.33	49.61	50.23	0.72	0.96	0.24	0.96	0.64
7	30-Apr-10	50.29	49.20	49.24	1.09	0.06	1.03	1.09	0.67
8	03-May-10	50.17	49.43	49.93	0.74	0.93	0.19	0.93	0.69
9	04-May-10	49.32	48.08	48.43	1.24	0.61	1.85	1.85	0.78
10	05-May-10	48.50	47.64	48.18	0.86	0.07	0.79	0.86	0.78
11	06-May-10	48.32	41.55	46.57	6.77	0.14	6.63	6.77	1.21
12	07-May-10	46.80	44.28	45.41	2.52	0.23	2.29	2.52	1.30
13	10-May-10	47.80	47.31	47.77	0.49	2.39	1.90	2.39	1.38
14	11-May-10	48.39	47.20	47.72	1.19	0.62	0.57	1.19	1.37
15	12-May-10	48.66	47.90	48.62	0.76	0.94	0.18	0.94	1.34
16	13-May-10	48.79	47.73	47.85	1.06	0.17	0.89	1.06	1.32

In [39]:
```
dates, open , high, low, close = np.loadtxt(datafile, delimiter=',',
                                    skiprows=1, usecols=(0,1,2,3,4),
                                           converters={0: datestr2num},
                                           encoding='utf-8',
                                           unpack=True)

open  = np.round(open, 2)
high  = np.round(high, 2)
low   = np.round(low, 2)
close = np.round(close, 2)
```

In [40]:
```
num_days = len(high)
num_days
```

Out[40]:
```
178
```

In [41]:
```
# ATR Window

N = 14
```

In [42]:
```
previous_close = close[: -1]
```

In [43]:
```
# TR except for first day

truerange = np.maximum(high[1:] - low[1:],
                       np.abs(high[1:] - previous_close),
                       np.abs(previous_close - low[1:]) )

# first day true range = (high[0] - low[0])
truerange = np.insert(truerange, 0, (high[0] - low[0]))

truerange
```

Out[43]:
```
array([ 4.62, 12.2 ,  6.36,  2.93,  3.89,  4.9 ,  3.11,  2.19,  2.05,
        3.39,  2.88,  4.4 ,  2.02,  4.11,  3.44,  2.74,  5.43,  2.67,
        4.02, 11.47,  4.44,  3.05,  5.14,  3.83,  2.72,  3.6 ,  2.24,
        1.96,  1.57,  2.56,  1.88,  1.95,  1.95,  2.39,  2.07,  1.94,
        2.9 ,  2.13,  2.27,  1.99,  2.26,  3.78,  1.46,  1.55,  2.42,
        3.57,  6.21,  3.77,  2.39,  2.39,  3.6 ,  2.6 ,  3.07,  4.76,
        8.17,  6.91,  5.38,  8.3 ,  3.21,  2.03,  1.54,  3.3 ,  3.41,
        3.35,  3.23,  1.41,  3.89,  3.62,  2.56,  2.56,  3.93,  1.84,
        2.81,  4.77,  1.63,  2.6 ,  3.85,  1.43,  2.64,  2.88,  2.11,
        4.29, 14.64,  4.52,  2.69,  5.34,  6.59,  3.59,  5.02,  6.08,
        7.7 ,  4.29,  5.73,  3.63,  4.14,  4.65,  4.91,  3.16,  2.73,
        3.52,  2.68,  3.35,  2.56,  3.  ,  7.65,  6.53,  5.35,  3.32,
        6.7 ,  5.22,  3.42,  2.58,  3.19,  3.29,  2.79,  6.4 ,  2.57,
        2.74,  2.7 ,  1.99,  3.97,  5.42,  2.  ,  2.45,  6.57,  1.77,
        1.75,  2.18,  2.99,  2.7 ,  2.49,  2.68,  2.25,  2.57,  2.61,
        1.82,  2.53,  4.14,  4.64,  1.69,  1.98,  2.51,  2.71,  2.9 ,
        2.85, 12.59, 11.29,  4.8 ,  6.07,  4.73,  5.74,  4.49,  3.47,
        2.9 , 11.66,  3.85,  5.47,  5.42,  6.23,  3.03,  3.29,  3.69,
       11.05,  4.55,  5.02,  2.4 ,  3.79,  3.25,  2.76,  3.78,  4.78,
        1.91,  5.37,  5.07,  7.01,  3.56,  3.77,  2.57])
```

In [44]:
```
len(truerange)
```

Out[44]:
```
178
```

In [ ]:

In [45]:
```
atr = np.zeros(num_days - N + 1)
len(atr)
```

Out[45]:
```
165
```

In [46]:
```
atr[0] = np.mean(truerange[:N])
atr[0]
```

Out[46]:
```
4.21785714285714
```

In [47]:
```
for i in range(1, len(atr)):
   atr[i] = (N - 1) * atr[i - 1] + truerange[N + i - 1]
   atr[i] /= N

print("ATR", atr)
```

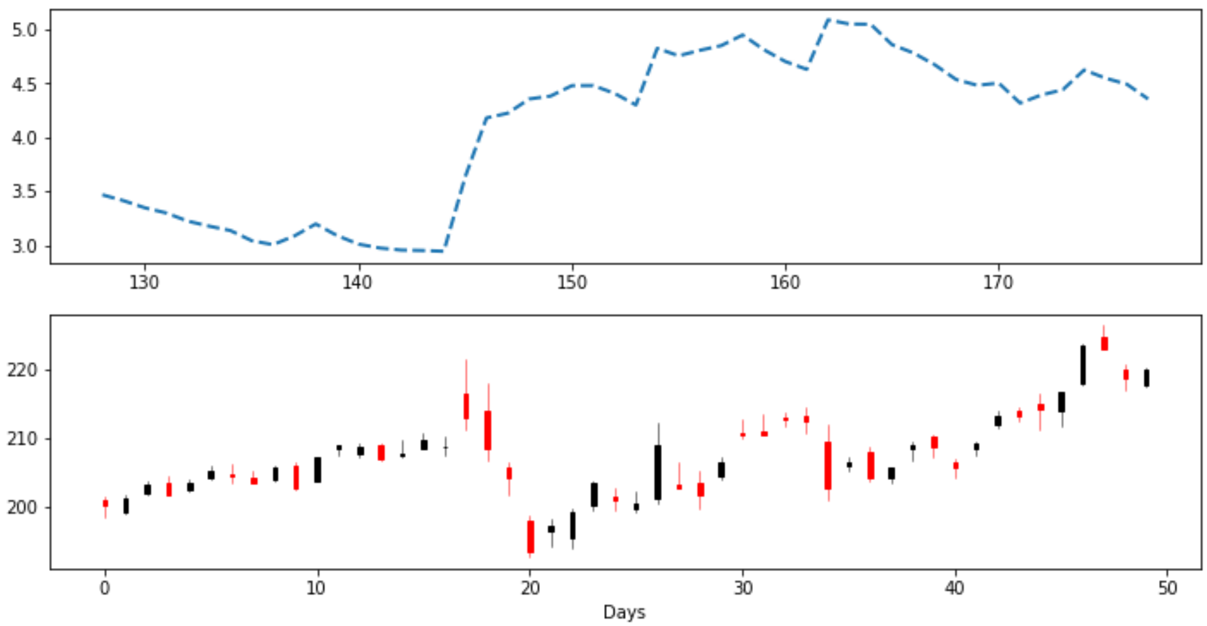
```
ATR [4.21785714 4.16229592 4.06070335 4.15851026 4.05218809 4.04988895
 4.57989688 4.56990424 4.46133965 4.50981539 4.46125715 4.33688164
 4.28424724 4.13822958 3.98264175 3.8103102  3.72100233 3.58950216
 3.47239486 3.36365237 3.29410577 3.20666965 3.11619324 3.10075087
 3.03141152 2.97702498 2.9065232  2.86034297 2.92603276 2.82131613
 2.73050784 2.70832871 2.76987666 3.01559975 3.06948548 3.02095081
 2.97588289 3.02046269 2.99042964 2.99611323 3.12210515 3.48266906
 3.72747842 3.84551567 4.16369312 4.09557219 3.94803132 3.77602908
 3.742027   3.71831079 3.69200287 3.65900267 3.49835962 3.52633393
 3.53302437 3.46352263 3.3989853  3.43691492 3.32284957 3.28621745
 3.39220192 3.26633036 3.21873533 3.26382566 3.13283812 3.09763539
 3.08209001 3.01265501 3.10389394 3.92790151 3.97019426 3.87875181
 3.98312668 4.16933192 4.12795107 4.19166885 4.32654965 4.56751039
 4.54768822 4.63213906 4.5605577  4.53051786 4.5390523  4.56554856
 4.46515224 4.34121279 4.28255474 4.16808654 4.10965179 3.99896237
 3.92760792 4.19349307 4.36038642 4.4310731  4.35171074 4.51944569
 4.56948528 4.48737919 4.35113782 4.2681994  4.19832802 4.09773316
 4.26218079 4.14131073 4.04121711 3.94541589 3.80574332 3.81747594
 3.93194195 3.79394609 3.69794994 3.90309638 3.75073235 3.6078229
 3.50583555 3.46899015 3.41406228 3.34805783 3.30033942 3.22531517
 3.17850695 3.13789931 3.04376364 3.00706624 3.08799008 3.19884793
 3.09107308 3.01171072 2.97587424 2.95688322 2.95282013 2.94547584
 3.63437042 4.1812011  4.22540103 4.3571581  4.38378966 4.48066183
 4.48132884 4.40909107 4.30129885 4.82692036 4.75714033 4.80805888
 4.85176896 4.95021403 4.81305589 4.70426618 4.6318186  5.09026013
 5.05167012 5.04940797 4.86016454 4.78372422 4.67417249 4.53744588
 4.4833426  4.50453242 4.31920867 4.3942652  4.44253197 4.62592254
 4.54978522 4.49408627 4.35665154]
```

Interpreting ATR and stock prices

https://www.tradingview.com/wiki/Average_True_Range_(ATR) (https://www.tradingview.com/wiki/Average_True_Range_(ATR))

In [48]:
```
import matplotlib.pyplot as plt
from mpl_finance import candlestick_ohlc
```

In [49]:
```
# Plot last 50 values

fig, ax = plt.subplots(2, figsize=(12,6))


t = np.arange(N - 1, num_days)
ax[0].plot(t[-50:], atr[-50:], '--', lw=2.0, label='ATR')
candlestick_ohlc(ax[1], zip(np.arange(len(high[-50:])),
                          open[-50:], high[-50:],
                          low[-50:], close[-50:]))

plt.xlabel('Days')
plt.show()
```



Interpreting Moving Averages

[https://www.tradingview.com/wiki/Moving_Average (https://www.tradingview.com/wiki/Moving_Average)](https://www.tradingview.com/wiki/Moving_Average)

Simple Moving Average (SMA)

- For analyzing time-series data
- Moving window of N periods
- Mean of values inside the window
- an unweighted moving average

In [50]:
```
x = np.array([11,12,13,14,15,16,17,18])
x
```
Out[50]:
```
array([11, 12, 13, 14, 15, 16, 17, 18])
```

In [51]:
```
# 5-Day Moving Average
N = 5
```

In [52]:
```
# First day of 5-day SMA
np.sum(x[0:N])/N
```
Out[52]:
```
13.0
```

In [53]:
```
# Second day of 5-day SMA
np.sum(x[1:N+1])/N
```
Out[53]:
```
14.0
```

In [54]:
```
# Third day of 5-day SMA
np.sum(x[2:N+2])/N
```
Out[54]:
```
15.0
```

In [55]:
```
# Fourth day of 5-day SMA
np.sum(x[3:N+3])/N
```
Out[55]:
```
16.0
```

In [56]:
```
# Using np.convolve
```

In [57]:
```
N = 5
weights = np.ones(N)/N
print("Weights", weights)
```
```
Weights [0.2 0.2 0.2 0.2 0.2]
```

In [58]:
```
np.convolve(x, weights)
```

Out[58]:
```
array([ 2.2,  4.6,  7.2, 10. , 13. , 14. , 15. , 16. , 13.2, 10.2,  7. ,
        3.6])
```

In [59]:
```
np.convolve(x, weights)[N-1:-(N-1)]
```

Out[59]:
```
array([13., 14., 15., 16.])
```

In [ ]:

In [ ]:

In [60]:
```
# Using the dataset

# Import Date and closing price from the file (columns 0 and 5)

dates, close = np.loadtxt(datafile, delimiter=',',
                          skiprows=1, usecols=(0,5),
                          converters={0: datestr2num},
                          encoding='utf-8',
                          unpack=True)

close = np.round(close, 2)
```

In [61]:
```
len(close)
```

Out[61]:
```
178
```

In [62]:
```
# 20-day moving window

N = 20
```

In [63]:
```
weights = np.ones(N)/N
```

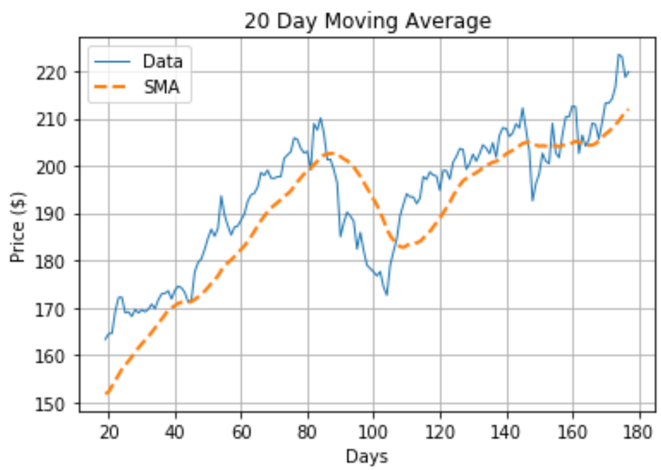
In [64]:
```
sma = np.convolve(c, weights)[N-1:-(N-1)]
len(sma)
```

Out[64]:
```
159
```

In [65]:
```
len(close[N-1:])
```

Out[65]:
```
159
```

In [66]:
```
t = np.arange(N - 1, len(close))

plt.plot(t, close[N-1:], lw=1.0, label="Data")

plt.plot(t, sma, '--', lw=2.0, label="SMA")

plt.title("20 Day Moving Average")
plt.xlabel("Days")
plt.ylabel("Price ($)")
plt.grid()
plt.legend()
plt.show()
```



In [67]:
```
fig, ax = plt.subplots(1, figsize=(12,8))

candlestick_ohlc(ax, zip(np.arange(len(high[N-1:])),
                          open[N-1:], high[N-1:],
                          low[N-1:], close[N-1:]))

t1 = np.arange(0, len(close) - N + 1)

plt.plot(t1, close[N-1:], lw=2.0, label="Data")

plt.plot(t1, sma, '--', lw=2.0, label="SMA")


plt.xlabel('Days')

plt.grid()
plt.legend()
plt.show()
```

Weighted Moving Average (WMA)

In [68]:
```
x = np.array([5,6,7,8,9])
weights = np.array([5,4,3,2,1])
np.convolve(x, weights)/np.sum(weights)
```
Out[68]:
```
array([1.66666667, 3.33333333, 4.93333333, 6.4       , 7.66666667,
       5.33333333, 3.33333333, 1.73333333, 0.6       ])
```

In [69]:
```
x = np.array([11,12,13,14,15,16,17,18])
x
```
Out[69]:
```
array([11, 12, 13, 14, 15, 16, 17, 18])
```

In [70]:
```
# 5-Day Moving Average
N = 5
```

In [71]:
```
weights = np.arange(1, N+1)
weights
```
Out[71]:
```
array([1, 2, 3, 4, 5])
```

In [72]:
```
# First day of 5-day WMA
np.sum(x[0:N] * weights)/sum(weights)
```
Out[72]:
```
13.666666666666666
```

In [73]:
```
# Second day of 5-day WMA
np.sum(x[1:N+1] * weights)/sum(weights)
```
Out[73]:
```
14.666666666666666
```

In [74]:
```
# Third day of 5-day WMA
np.sum(x[2:N+2] * weights)/sum(weights)
```
Out[74]:
```
15.666666666666666
```

In [75]:
```
# Fourth day of 5-day WMA
np.sum(x[3:N+3] * weights)/sum(weights)
```
Out[75]:
```
16.666666666666668
```

In [76]:
```
# Same as

(np.convolve(x, weights[::-1])[N-1:-(N-1)])/sum(weights)
```

Out[76]:
```
array([13.66666667, 14.66666667, 15.66666667, 16.66666667])
```

In [77]:
```
# Using the dataset

dates, close = np.loadtxt(datafile, delimiter=',',
                          skiprows=1, usecols=(0,5),
                          converters={0: datestr2num},
                          encoding='utf-8',
                          unpack=True)

close = np.round(close, 2)
```

In [78]:
```
# 20-day moving window

N = 20
```

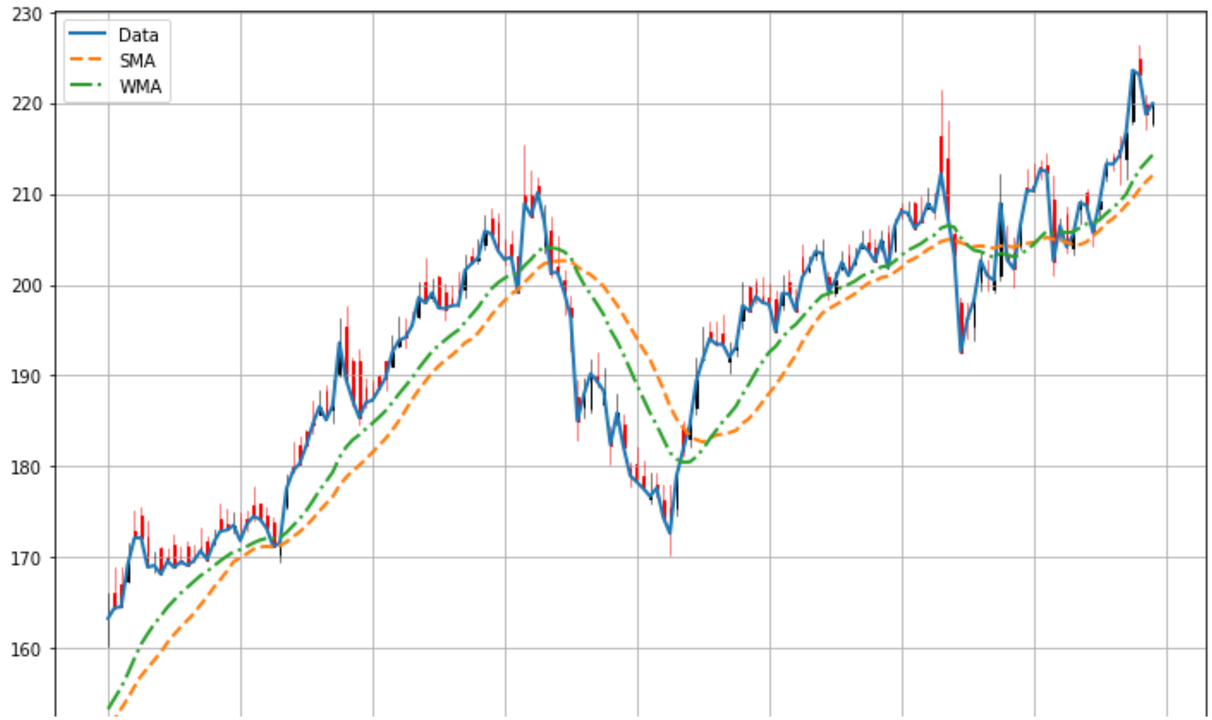
In [79]:
```
weights = np.arange(1, N+1)
weights
```

Out[79]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20])
```

In [80]:
```
wma = (np.convolve(c, weights[::-1])[N-1:-(N-1)])/sum(weights)
wma[:5]
```

Out[80]:
```
array([153.29871429, 154.5117619 , 155.69233333, 157.20319048,
       158.88204762])
```

In [81]:
```
fig, ax = plt.subplots(1, figsize=(12,8))

candlestick_ohlc(ax, zip(np.arange(len(high[N-1:])),
                         open[N-1:], high[N-1:],
                         low[N-1:], close[N-1:]))

t1 = np.arange(0, len(close) - N + 1)

plt.plot(t1, close[N-1:], lw=2.0, label="Data")

plt.plot(t1, sma, '--', lw=2.0, label="SMA")

plt.plot(t1, wma, '-.', lw=2.0, label="WMA")


plt.xlabel('Days')

plt.grid()
plt.legend()
plt.show()
```



In [ ]:

Exponential Moving Average (EMA)

- For analyzing time-series data
- Alternative to SMA
- Moving window of N periods
- Uses exponentially decreasing weights
- Gives higher weights to recent prices

In [82]:
```
x = np.array([11,12,13,14,15,16,17,18])
x
```
Out[82]:
```
array([11, 12, 13, 14, 15, 16, 17, 18])
```

In [83]:
```
N = 5
weights = np.exp(np.linspace(0, 1, N))
weights
```
Out[83]:
```
array([1.        , 1.28402542, 1.64872127, 2.11700002, 2.71828183])
```

In [84]:
```
# Normalize weights
weights /= weights.sum()
print("Weights", weights)
```
```
Weights [0.11405072 0.14644403 0.18803785 0.24144538 0.31002201]
```

In [85]:
```
np.convolve(x, weights[::-1])[N-1:-(N-1)]
```
Out[85]:
```
array([13.48694393, 14.48694393, 15.48694393, 16.48694393])
```

In [ ]:

In [86]:
```
# Using the dataset

# Import Date and closing price from the file (columns 0 and 5)

dates, close = np.loadtxt(datafile, delimiter=',',
                          skiprows=1, usecols=(0,5),
                          converters={0: datestr2num},
                          encoding='utf-8',
                          unpack=True)

close = np.round(close, 2)
```

In [87]:
```
# 20-day moving window

N = 20
```

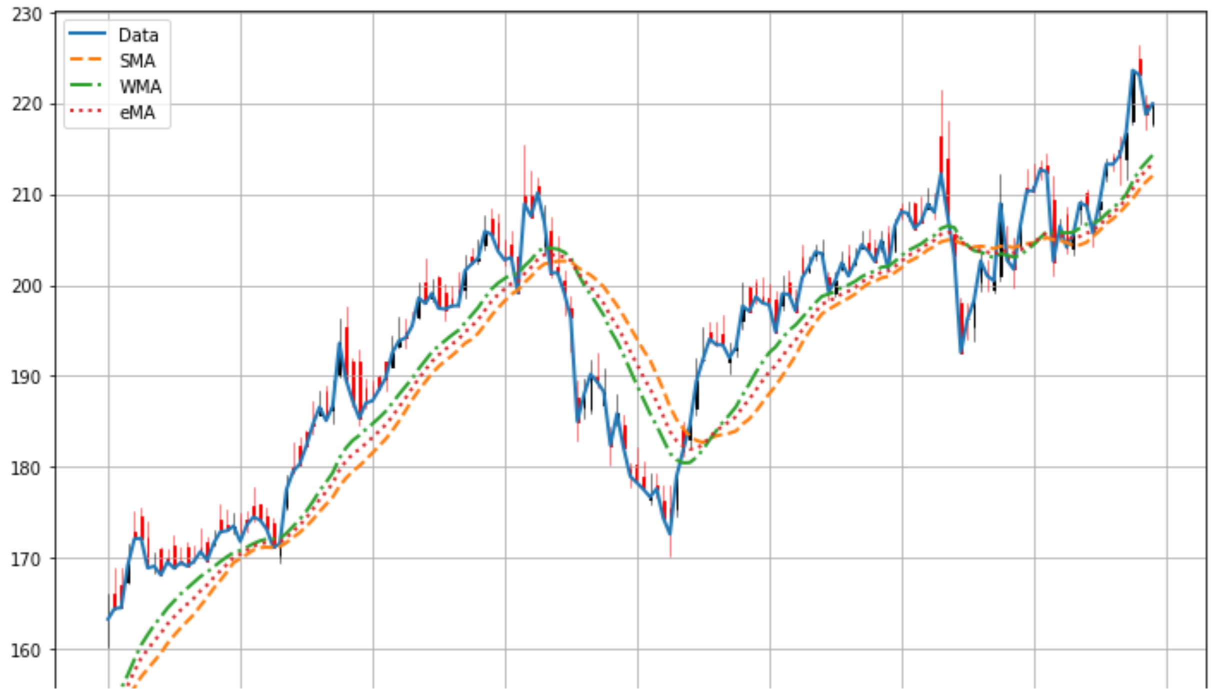
In [88]:
```
weights = np.exp(np.linspace(0, 1, N))

# Normalize weights
weights /= weights.sum()
```

In [89]:
```
ema = np.convolve(c, weights[::-1])[N-1:-(N-1)]
len(ema)
```

Out[89]: 159

In [ ]:
```
```

In [90]:
```
fig, ax = plt.subplots(1, figsize=(12,8))

candlestick_ohlc(ax, zip(np.arange(len(high[N-1:])),
                         open[N-1:], high[N-1:],
                         low[N-1:], close[N-1:]))

t1 = np.arange(0, len(close) - N + 1)

plt.plot(t1, close[N-1:], lw=2.0, label="Data")

plt.plot(t1, sma, '--', lw=2.0, label="SMA")

plt.plot(t1, wma, '-.', lw=2.0, label="WMA")

plt.plot(t1, ema, ':', lw=2.0, label="eMA")


plt.xlabel('Days')

plt.grid()
plt.legend()
plt.show()
```



In [ ]:

```

```