Table of Contents

[Rolling Moving Average 1](#_Toc83410944)

[Trailing Mean 1](#_Toc83410945)

[Centered Moving Average 1](#_Toc83410946)

[Weighted Moving Average (Rolling Mean) 3](#_Toc83410947)

[Using Moving Averages to Buy and Sell Stock 3](#_Toc83410948)

[Exponentially Weighted Moving Average (Exponential Smoothing) 5](#_Toc83410949)

[Back-Testing 8](#_Toc83410950)

**References:**

<https://machinelearningmastery.com/moving-average-smoothing-for-time-series-forecasting-python/>

<https://machinelearningmastery.com/autoregression-models-time-series-forecasting-python/>

## Rolling Moving Average

The rolling moving average is a naïve and effective technique that can be used in time series forecasting. Rolling moving averages also help to reduce volatility when trying to understand trends, cycles and seasonal components.

The rolling moving average, also called the rolling mean, can be calculated in several ways.

### Trailing Mean

A trailing mean is calculated with an evenly weighted average of sample observations.

Trailing mean =

### Centered Moving Average

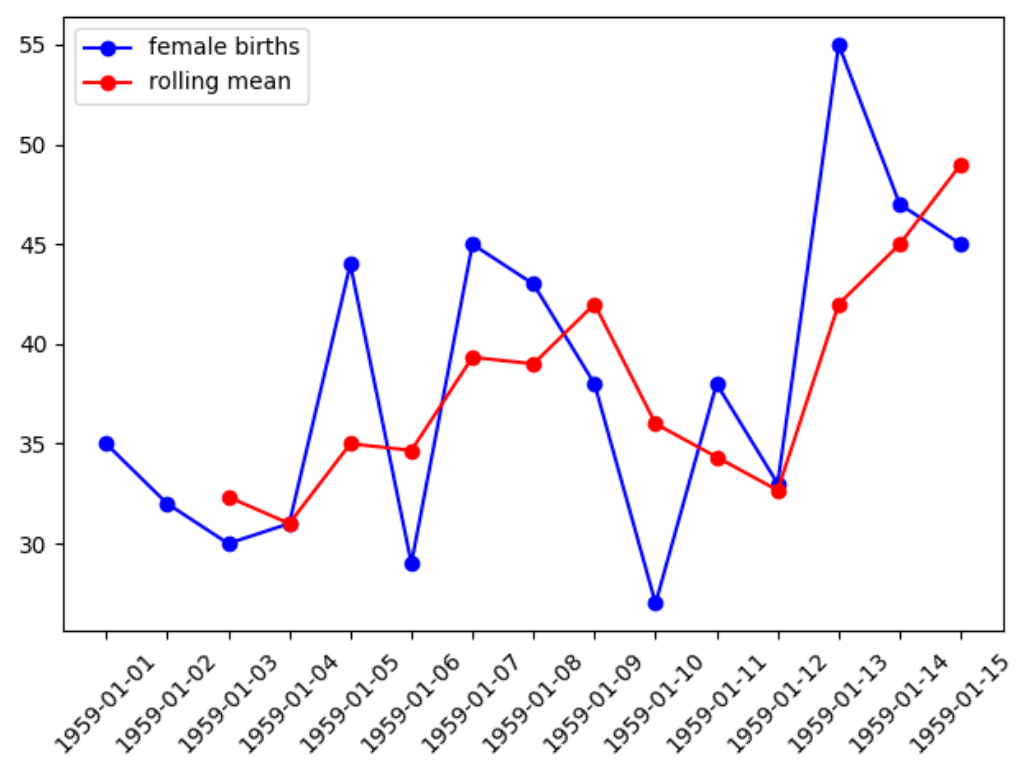
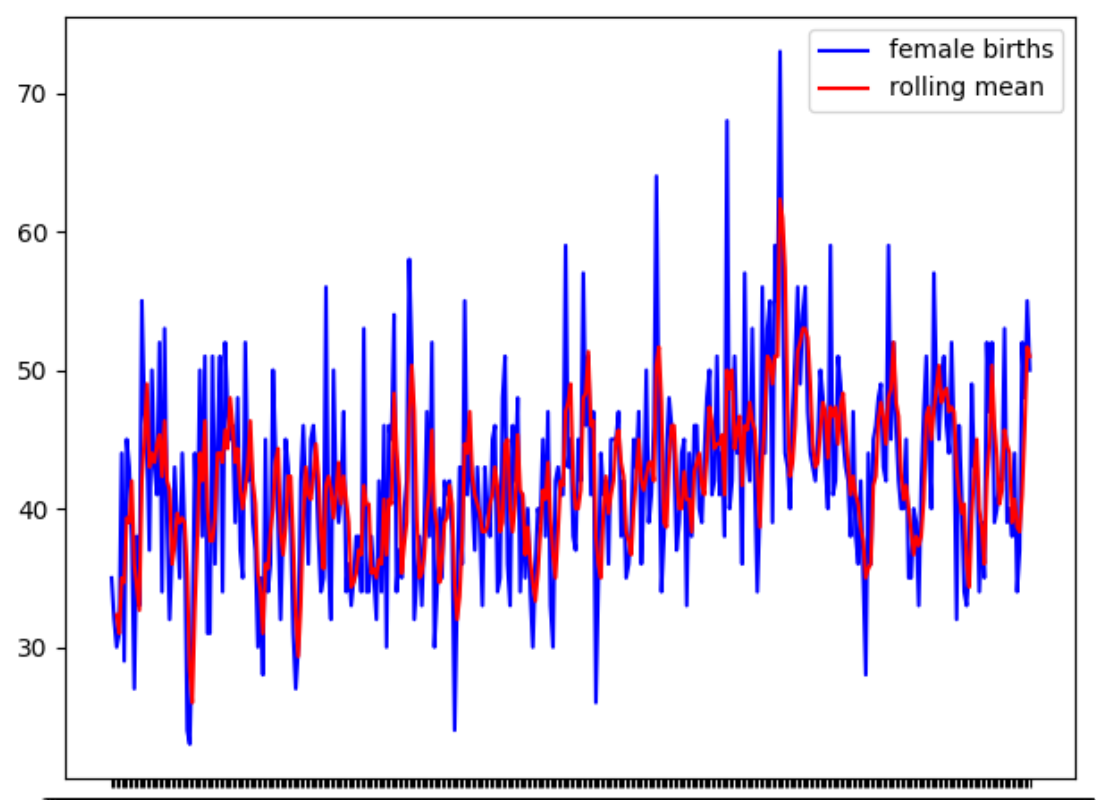
The centered moving average is calculated by averaging an equally weighted number of observations about the current time.

Centered moving average =

Example : Rolling Moving Average of Daily Female Births

This example shows how to calculate the trailing moving average of daily female births. Figure 1 shows plots with the rolling mean and actual values. Note that the rolling mean is typically less volatile but it still follows the same trend.

Figure : Rolling Mean and Actual Values for all Values and for the 1st 15 days



|  |
| --- |
| from pandas import read\_csv  import matplotlib.pyplot as plt  PATH = "/Users/pm/Desktop/DayDocs/2019\_2020/PythonForDataAnalytics/workingData/"  FILE = 'daily-total-female-births.csv'  series = read\_csv(PATH + FILE, header=0, index\_col=0)  print(series.head())  series.plot(rot=45)  plt.show()  # Calculate rolling moving average 3 steps back.  print("\n\*\*\* Rolling mean")  rolling = series.rolling(window=3)  rolling\_mean = rolling.mean()  print(rolling\_mean.head(5))  # Plot actual and rolling mean values.  plt.plot(series, color='blue', label='female births')  plt.plot(rolling\_mean, color='red', label='rolling mean')  plt.legend()  plt.show() |

Example :Manually Calculating the Trailing Moving Average

The moving average in Example 1 is calculated by taking the average of recent target values. Table 1 shows the manual calculations needed to obtain a rolling moving average that is based on the 3 most recent values.

Table : Rolling Moving Average Calculation

|  |  |  |
| --- | --- | --- |
| Original Series | obs(t) = (t-2 + t-1 + t)/3 | Rolling moving average. |
| 1959-01-01 35  1959-01-02 32  1959-01-03 30  1959-01-04 31  1959-01-05 44 | =(35+32+30)/3=32.33333  =(32+30+31)/3=31.00000 | 1959-01-01 NaN  1959-01-02 NaN  1959-01-03 32.333333  1959-01-04 31.000000  1959-01-05 35.000000 |

Exercise (2 marks)

Perform the manual calculation to calculate the rolling moving average for January 5, 1959. Show your result here:

|  |
| --- |
| =(30 + 31+ 44)/ 3 = 35.3333333 |

## Weighted Moving Average (Rolling Mean)

Weighted moving averages smooth the fluctuations but they lag behind the actual data.

Trailing mean =

The rolling mean can be calculated with the following code:

|  |
| --- |
| rolling\_mean = df['Close'].rolling(window=20).mean() |

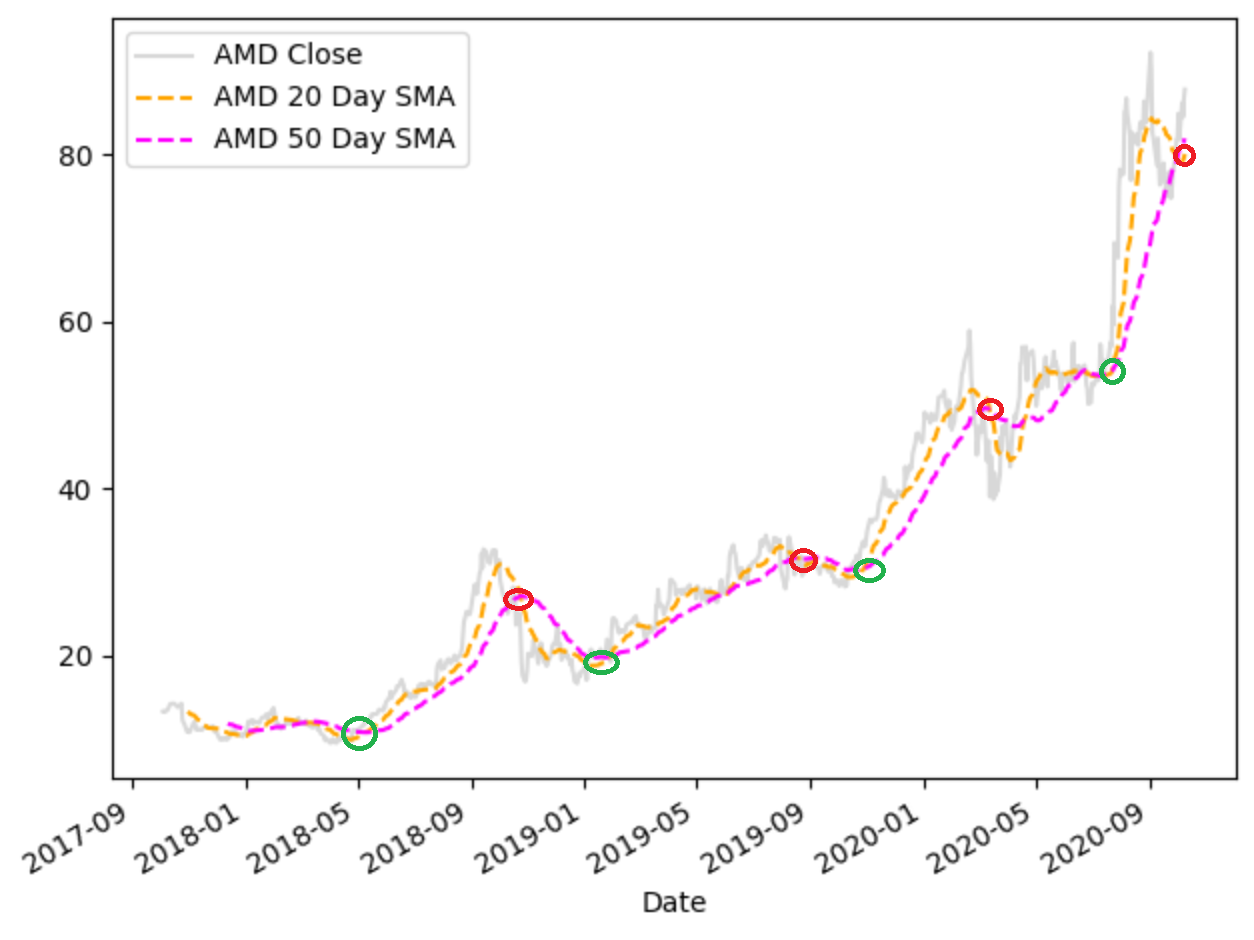
## Using Moving Averages to Buy and Sell Stock

A common stock trading strategy involves using moving averages to predict when to buy or sell. When the short term moving average cross above the long term moving average buy. When the short term moving average falls below the long term moving average sell (refer to Figure 2).

Example : Weighted Moving Average

This example shows how a 20-day versus a 50-day simple moving average for AMD stock closing prices. Both moving average plots are noticeably smoother than the actual time closing prices plot. The shorter moving average is a closer match to the actual data since it uses more current data.

Figure : Moving Average Trading Strategy



Here is the code:

|  |
| --- |
| import datetime  import pandas\_datareader as pdr  import matplotlib.pyplot as plt  def getStock(stk, ttlDays):  numDays = int(ttlDays)  dt = datetime.date.today()  dtPast = dt + datetime.timedelta(days=-numDays)  df = pdr.get\_data\_yahoo(stk,  start=datetime.datetime(dtPast.year, dtPast.month,  dtPast.day),  end=datetime.datetime(dt.year, dt.month, dt.day))  return df  df = getStock('AMD', 1100)  print(df)  rolling\_mean = df['Close'].rolling(window=20).mean()  rolling\_mean2 = df['Close'].rolling(window=50).mean()  #plt.figure(figsize=(10,30))  df['Close'].plot(label='AMD Close ', color='gray', alpha=0.3)  rolling\_mean.plot(label='AMD 20 Day SMA', style='--', color='orange')  rolling\_mean2.plot(label='AMD 50 Day SMA', style='--',color='magenta')  plt.legend()  plt.show() |

Exercise (1 mark)

The green circle at January 2019 indicates one of the following. Please highlight the correct answer: a) Buy b) Sell

The red circle near September 2019 indicates one of the following. Please highlight the correct answer: a) Buy b) Sell

Exercise (1 mark)

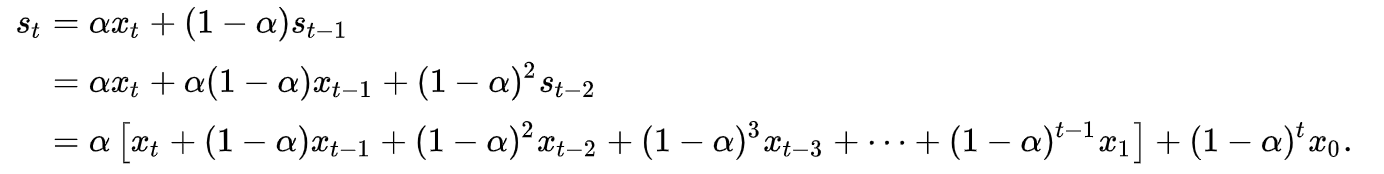
Indicate if the following is true or false.

A buy is indicated when the long-term moving average crosses above the short-term moving average during an upwards stock price trend.

1. True b) False

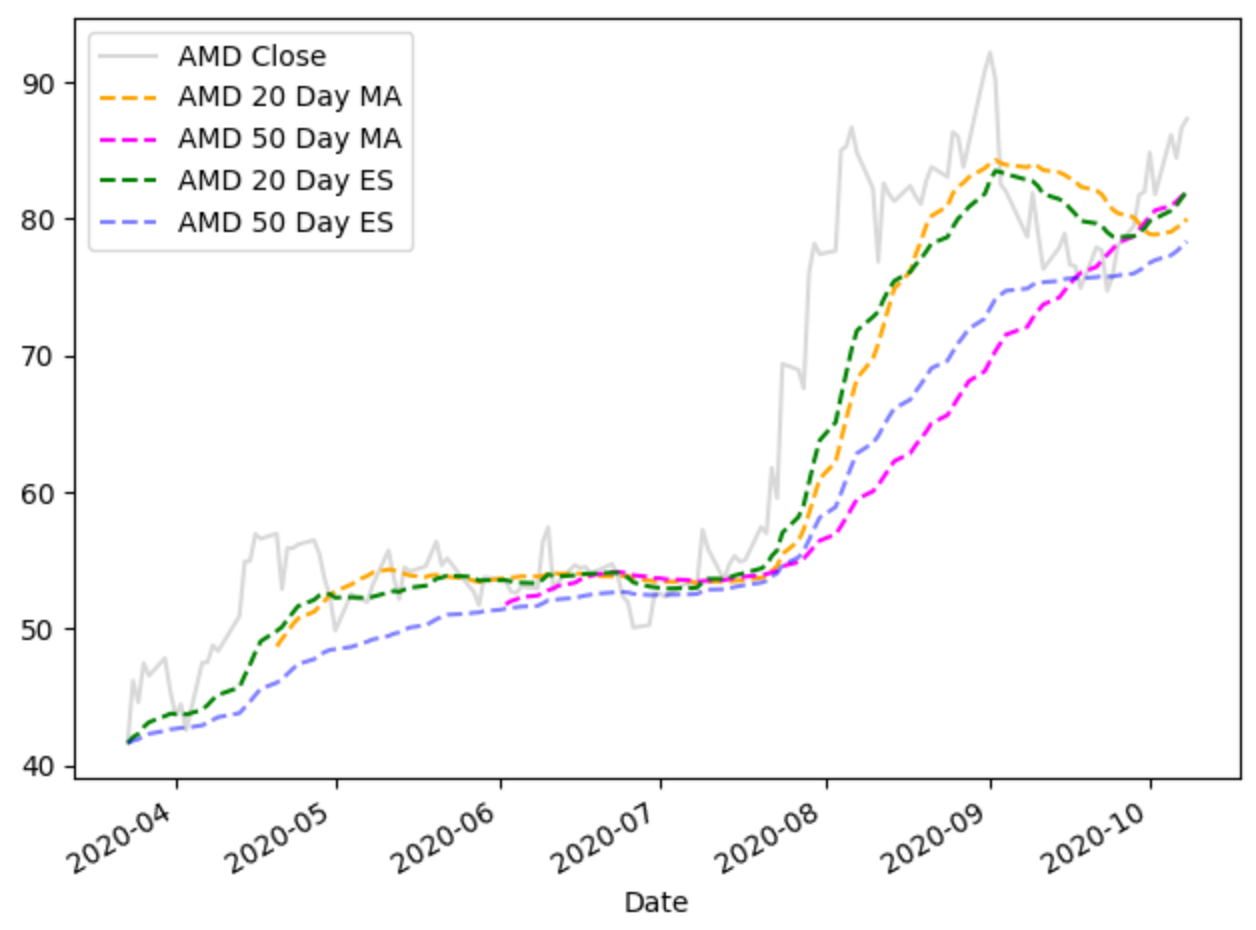
## Exponentially Weighted Moving Average (Exponential Smoothing)

Exponential smoothing provides a slightly better fit to the fluctuations in the time series because it places a higher weight on more recent time steps. Exponential smoothing lags behind the actual data less than the simple moving average.



The **50-day** exponential smoothing line (blue) is better able to represent the actual data fluctuations compared to the **50-day** moving average line (pink). The **20-day** exponential smoothing line (green) is better able to represent the actual data fluctuations compared to the **20-day** moving average line (yellow). Refer to Figure 3.

Figure : Moving Average Versus Exponential Smoothing



Exponential smoothing is implemented with the following code:

|  |
| --- |
| exp20 = df['Close'].ewm(span=20, adjust=False).mean() |

Example : Exponential Smoothing

This example implements the code that is used to calculate and present the moving averages and exponentially smoothed data that is plotted in .

|  |
| --- |
| import datetime  import pandas\_datareader as pdr  import matplotlib.pyplot as plt  def getStock(stk, ttlDays):  numDays = int(ttlDays)  dt = datetime.date.today()  dtPast = dt + datetime.timedelta(days=-numDays)  df = pdr.get\_data\_yahoo(stk,  start=datetime.datetime(dtPast.year, dtPast.month,  dtPast.day),  end=datetime.datetime(dt.year, dt.month, dt.day))  return df  df = getStock('AMD', 200)  # Calculating the moving averages.  rolling\_mean = df['Close'].rolling(window=20).mean()  rolling\_mean2 = df['Close'].rolling(window=50).mean()  # Calculate the exponentially smoothed series.  exp20 = df['Close'].ewm(span=20, adjust=False).mean()  exp50 = df['Close'].ewm(span=50, adjust=False).mean()  #plt.figure(figsize=(10,30))  df['Close'].plot(label='AMD Close ', color='gray', alpha=0.3)  rolling\_mean.plot(label='AMD 20 Day MA', style='--', color='orange')  rolling\_mean2.plot(label='AMD 50 Day MA', style='--',color='magenta')  exp20.plot(label='AMD 20 Day ES', style='--',color='green')  exp50.plot(label='AMD 50 Day ES', style='--',color='blue', alpha=0.5)  plt.legend()  plt.show() |

Exercise (3 marks)

In the same graph, plot the 50-day moving average for Microsoft stock closing prices. (Microsoft uses stock symbol MSFT). Also plot the 50-day exponentially weighted moving average. Please label the graph properly and adjust the names of your variables as needed as well. Show your plot here:

|  |
| --- |
| Chart, line chart  Description automatically generated |

Please show your code here:

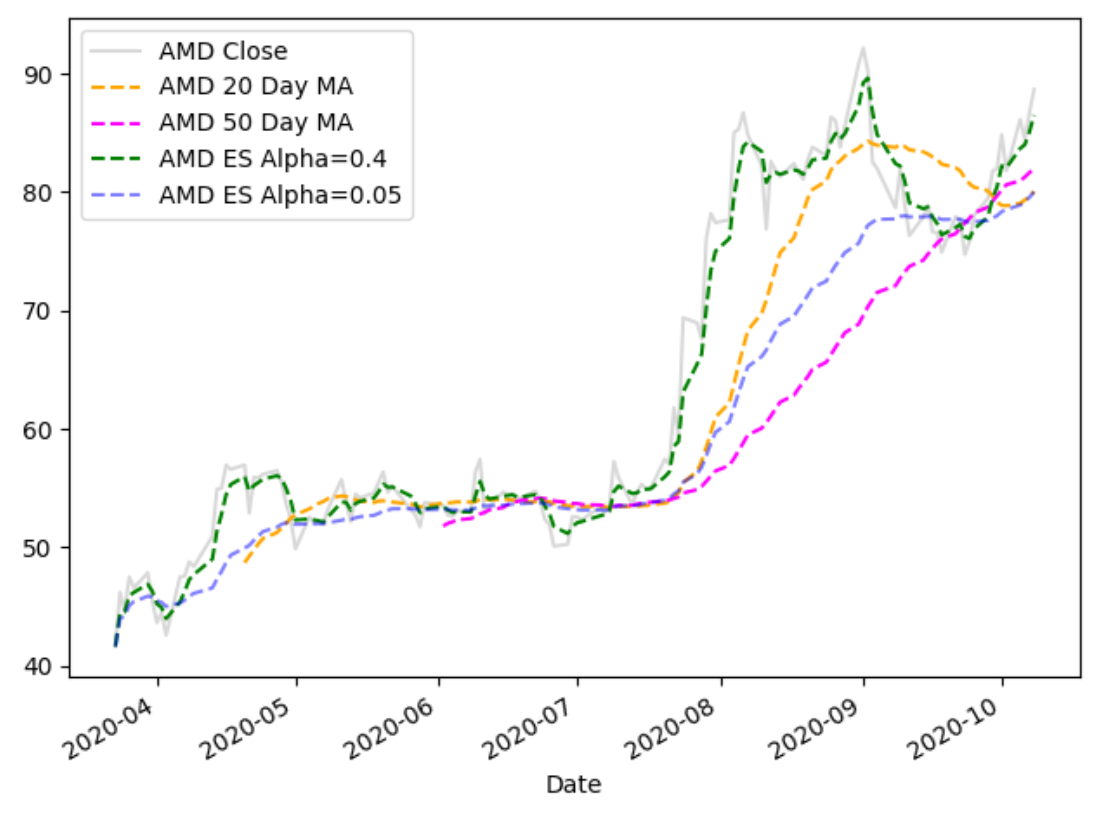
|  |
| --- |
| import datetime import pandas\_datareader as pdr import matplotlib.pyplot as plt  def getStock(stk, ttlDays):  numDays = int(ttlDays)  dt = datetime.date.today()  dtPast = dt + datetime.timedelta(days=-numDays)  df = pdr.get\_data\_yahoo(stk,  start=datetime.datetime(dtPast.year, dtPast.month,  dtPast.day),  end=datetime.datetime(dt.year, dt.month, dt.day))  return df  df = getStock('MSFT', 50)  # Calculating the moving averages. # rolling\_mean = df['Close'].rolling(window=20).mean() rolling\_mean2 = df['Close'].rolling(window=50).mean()  # Calculate the exponentially smoothed series. # exp20 = df['Close'].ewm(span=20, adjust=False).mean() exp50 = df['Close'].ewm(span=50, adjust=False).mean()  #plt.figure(figsize=(10,30)) df['Close'].plot(label='Microsoft Close ', color='gray', alpha=0.3) # rolling\_mean.plot(label='Microsoft 20 Day MA', style='--', color='orange') # rolling\_mean2.plot(label='Microsoft 50 Day MA', style='--',color='magenta') # exp20.plot(label='Microsoft 20 Day ES', style='--',color='green') exp50.plot(label='Microsoft 50 Day ES', style='--',color='blue', alpha=0.5) plt.legend() plt.show() |

Is the exponentially smoothed data more quick to respond to rises and falls than the simple moving average? Explain why or why not.

|  |
| --- |
| Exponential smoothing provides better fit because it places a higher weight on more recent time steps. |

Example : Adjusting the Alpha Value

With exponential smoothing, instead of specifying days you can also set the alpha value to assign a weight to the first time-lag. Higher alpha values assign more importance to the more current time lags.



Starting with , replace the code which calculates the exponentially weighted moving average with this code:

|  |
| --- |
| exp20 = df['Close'].ewm(alpha=0.4).mean()  exp50 = df['Close'].ewm(alpha=0.05).mean() |

Exercise (1 mark)

Based on your observation of Example 5, state if the values that are modified with an exponentially weighted moving average when using a larger alpha value are more responsive to fluctuations in the actual closing prices compared to the exponentially weighted data that uses a smaller alpha value. Clearly explain why or why not.

|  |
| --- |
| It is because the higher alpha tracks the date more closely by giving more wight to recent data. |

## Back-Testing

It is difficult to validate time series data so back-testing is often used with time series data. Back testing involves an automated test which measures the results of a strategy over the long term with existing data.

Effective back tests must satisfy two conditions.

1. Back tests must be fully automated.
2. Back tests must be completely objective.

There is no standard way to write a back test – it does require creativity.

Example : Back-Testing the Buying and Selling of Stock Based on the Cross Over of Moving Averages

This example tests the buy and sell strategy which involves purchasing stock with the short-term moving average crosses over the long-term moving average for AMD stock. When the starting balance is **$10,000** the test indicates that buying and holding AMD stock will become **$33,878.40** if the stock is bought at the start of the trial and sold at the end. However, if moving averages are used to trigger buy and sell stock dates the balance will only become **$26,605.14**. In other words, the moving average strategy is less effective than the buy and hold strategy.

Figure 4 shows the points where the buy and sell activity occurs for the moving average stategy.

Figure : Buy and Sell Moving Average Strategy for AMD Stock

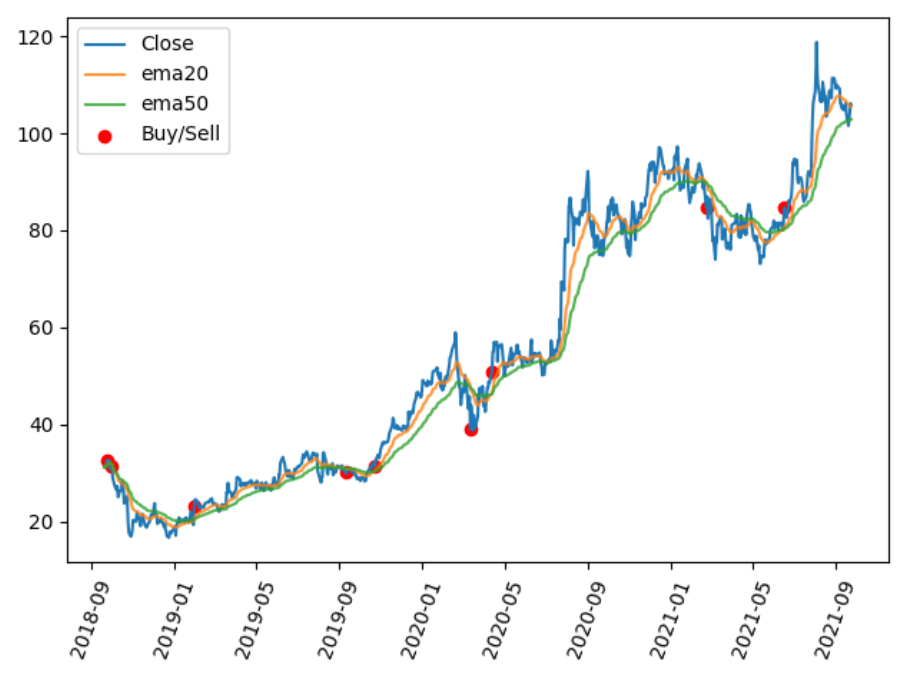


Table : Buy and Hold Earnings Versus Moving Average Triggered Buy and Sell Earnings

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Lt: 50

St: 20

-------------------------------------------------------

**Buy and hold closing balance: $33878.40**

-------------------------------------------------------

buyDt buy$ sellDt sell$ balance

0 2018-09-24 32.610001 2018-10-01 31.420000 9635.859837

1 2019-01-30 23.090000 2019-09-10 30.230000 12613.239582

2 2019-10-23 31.360001 2020-03-12 39.009998 15688.538662

3 2020-04-13 50.939999 2021-02-23 84.739998 26065.138428

**Moving average strategy closing balance: $26065.14**

|  |
| --- |
| import datetime  import pandas\_datareader as pdr  import matplotlib.pyplot as plt  import pandas as pd  import numpy as np  # Show all columns.  pd.set\_option('display.max\_columns', None)  # Increase number of columns that display on one line.  pd.set\_option('display.width', 1000)  def getStock(stk, ttlDays):  numDays = int(ttlDays)  dt = datetime.date.today()  dtPast = dt + datetime.timedelta(days=-numDays)  df = pdr.get\_data\_yahoo(stk,  start=datetime.datetime(dtPast.year, dtPast.month,  dtPast.day),  end=datetime.datetime(dt.year, dt.month, dt.day))  return df  def getNewBalance(startBalance, startPrice, endPrice):  qty = int(startBalance / startPrice)  cashLeftOver = startBalance - qty \* startPrice  endValue = qty \* endPrice  balance = cashLeftOver + endValue  return balance  def showBuyAndHoldEarnings(df, balance):  startClosePrice = df.iloc[0]['Close']  endClosePrice = df.iloc[len(df) - 1]['Close']  newBalance = getNewBalance(balance, startClosePrice, endClosePrice)  print("Buy and hold closing balance: $" + str(round(newBalance, 2)))  def showStrategyEarnings(df, balance, lt, st):  buyPrice = 0  buyDate = None  sellDate = None  bought = False  buySellDates = []  prices = []  dfStrategy = pd.DataFrame(columns=['buyDt', 'buy$', 'sellDt',  'sell$', 'balance' ])  dates = list(df.index)  for i in range(0, len(df)):  if(df.iloc[i]['Buy'] and not bought):  buyPrice = df.iloc[i]['Close']  buyDate = dates[i]  bought = True  buySellDates.append(buyDate)  prices.append(buyPrice)  elif(df.iloc[i]['Sell'] and bought):  sellPrice = df.iloc[i]['Close']  balance = getNewBalance(balance, buyPrice, sellPrice)  sellDate = dates[i]  buySellInfo = { 'buyDt':buyDate, 'buy$':buyPrice,  'sellDt':sellDate, 'sell$':sellPrice,  'balance':balance,}  dfStrategy = dfStrategy.append(buySellInfo, ignore\_index=True)  bought = False  buySellDates.append(sellDate)  prices.append(sellPrice)    print(dfStrategy)  print("\nMoving average strategy closing balance: $" + str(round(balance, 2)))  return buySellDates, prices  def showBuyAndSellDates(df, startBalance):  strategyDates, strategyPrices = showStrategyEarnings(df, startBalance, lt, st)  plt.plot(df.index, df['Close'], label='Close')  plt.plot(df.index, df['ema20'], label='ema20', alpha=0.4)  plt.plot(df.index, df['ema50'], label='ema50', alpha=0.4)  plt.scatter(strategyDates, strategyPrices, label='Buy/Sell', color='red')  plt.xticks(rotation=70)  plt.legend()  plt.show()    def showInvestmentDifferences(dfStock, lt, st):  df = dfStock.copy()  df['ema50'] = df['Close'].ewm(span=lt).mean()  df['ema20'] = df['Close'].ewm(span=st).mean()  # Remove nulls.  df.dropna(inplace=True)  df.round(3)  own\_positions = np.where(df['ema20'] > df['ema50'], 1, 0)  df['Position'] = own\_positions  df.round(3)  df['Buy'] = (df['Position'] == 1) & (df['Position'].shift(1) == 0)  df['Sell'] = (df['Position'] == 0) & (df['Position'].shift(1) == 1)  START\_BALANCE = 10000  print("-------------------------------------------------------")  showBuyAndHoldEarnings(df, START\_BALANCE)  print("-------------------------------------------------------")  showBuyAndSellDates(df, START\_BALANCE)    longterms = [50]  shortterms = [30]  dfStock = getStock('AMD', 1100)  for lt in longterms:  for st in shortterms:  print("\b\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("Lt: " + str(lt))  print("St: " + str(st))  showInvestmentDifferences(dfStock, lt,st) |

Exercise (2 marks)

Using the back test, try to find a different combination of long and short-term durations which yield a better return for AMD stock when using the moving average strategy. What combination of long and short-term periods offers better results?

|  |
| --- |
| Rectangle  Description automatically generated with medium confidence |

Show the text output here:

|  |
| --- |
| Text  Description automatically generated |

Show the graph here:

|  |
| --- |
| Chart, histogram  Description automatically generated |

Exercise (4 marks)

Implement the back test for XOM (Exxon).

|  |
| --- |
| import datetime import pandas\_datareader as pdr import matplotlib.pyplot as plt import pandas as pd import numpy as np from statistics import mean  # Show all columns. pd.set\_option('display.max\_columns', None)  # Increase number of columns that display on one line. pd.set\_option('display.width', 1000)  averages = []  def getStock(stk, ttlDays):  numDays = int(ttlDays)  dt = datetime.date.today()  dtPast = dt + datetime.timedelta(days=-numDays)  df = pdr.get\_data\_yahoo(stk,  start=datetime.datetime(dtPast.year, dtPast.month,  dtPast.day),  end=datetime.datetime(dt.year, dt.month, dt.day))  return df   def getNewBalance(startBalance, startPrice, endPrice):  qty = int(startBalance / startPrice)  cashLeftOver = startBalance - qty \* startPrice  endValue = qty \* endPrice  balance = cashLeftOver + endValue  return balance   def showBuyAndHoldEarnings(df, balance):  startClosePrice = df.iloc[0]['Close']  endClosePrice = df.iloc[len(df) - 1]['Close']  newBalance = getNewBalance(balance, startClosePrice, endClosePrice)  print("Buy and hold closing balance: $" + str(round(newBalance, 2)))   def showStrategyEarnings(df, balance, lt, st):  buyPrice = 0  buyDate = None  sellDate = None  bought = False   buySellDates = []  prices = []   dfStrategy = pd.DataFrame(columns=['buyDt', 'buy$', 'sellDt',  'sell$', 'balance'])  dates = list(df.index)  for i in range(0, len(df)):  if (df.iloc[i]['Buy'] and not bought):  buyPrice = df.iloc[i]['Close']  buyDate = dates[i]  bought = True  buySellDates.append(buyDate)  prices.append(buyPrice)   elif (df.iloc[i]['Sell'] and bought):  sellPrice = df.iloc[i]['Close']  balance = getNewBalance(balance, buyPrice, sellPrice)  sellDate = dates[i]  buySellInfo = {'buyDt': buyDate, 'buy$': buyPrice,  'sellDt': sellDate, 'sell$': sellPrice,  'balance': balance, }  dfStrategy = dfStrategy.append(buySellInfo, ignore\_index=True)  bought = False  buySellDates.append(sellDate)  prices.append(sellPrice)   print(dfStrategy)  averages.append(round(balance, 2))  print("\nMoving average strategy closing balance: $" + str(round(balance, 2)))  return buySellDates, prices   def showBuyAndSellDates(df, startBalance):  strategyDates, strategyPrices = showStrategyEarnings(df, startBalance, lt, st)  plt.plot(df.index, df['Close'], label='Close')  plt.plot(df.index, df['ema20'], label='ema20', alpha=0.4)  plt.plot(df.index, df['ema50'], label='ema50', alpha=0.4)  plt.scatter(strategyDates, strategyPrices, label='Buy/Sell', color='red')  plt.xticks(rotation=70)  plt.legend()  plt.tight\_layout()  plt.show()   def showInvestmentDifferences(dfStock, lt, st):  df = dfStock.copy()  df['ema50'] = df['Close'].ewm(span=lt).mean()  df['ema20'] = df['Close'].ewm(span=st).mean()   # Remove nulls.  df.dropna(inplace=True)  df.round(3)  own\_positions = np.where(df['ema20'] > df['ema50'], 1, 0)  df['Position'] = own\_positions  df.round(3)   df['Buy'] = (df['Position'] == 1) & (df['Position'].shift(1) == 0)  df['Sell'] = (df['Position'] == 0) & (df['Position'].shift(1) == 1)   START\_BALANCE = 10000   print("-------------------------------------------------------")  showBuyAndHoldEarnings(df, START\_BALANCE)  print("-------------------------------------------------------")  showBuyAndSellDates(df, START\_BALANCE)   longterms = range(50,110,20) shortterms = range(10,30,5) dfStock = getStock('XOM', 1100)  for lt in longterms:  for st in shortterms:  print("\b\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("Lt: " + str(lt))  print("St: " + str(st))  showInvestmentDifferences(dfStock, lt, st)  print(f"Average moving average strategy closing balance between 50-100 long term days in 10 day"  f" increments and 10-30 short term days in 5 day increments is ${mean(averages)}.") |

What conclusion can be made about buying and selling with the moving average strategy for Exxon? Does the moving average perform better than buy and hold for the period under examination?

|  |
| --- |
| Moving average perform better than buy and hold. Moving average value was $164909.43 but the buy and hold value was $12480.8 |

Show the text output here:

|  |
| --- |
| Shape, rectangle  Description automatically generatedText  Description automatically generated |

Show the graph here:

|  |
| --- |
|  |

Exercise (2 marks)

This is a subjective answer. What kind of changes do you think could be made to improve the results of the buy and sell strategy with moving averages?

|  |
| --- |
| We can fine the better performance with moving average for stock market because we can make more combination to find the better work. |