DATA604: Simulation and Modeling Techniques

Final Project: Turtle Trading Simulator, Michael O'Donnell, 7.16.20

Requirements:

Using SimPy, write a process simulation that includes waiting time (discrete event simulation). You may use any topic of interest to you. Write the simulation and all of the following in Jupyter.

Each element is worth 5 points and will be graded using the rubric shown here.

1. State the problem and its significance.
2. Provide a flow-chart model.
3. Simulate the process for the appropriate number of iterations (justify)
4. Justify the validity of the model and discuss how you verified it.
5. State your conclusions/ findings from the model.
6. Generate appropriate graphs (more than one) to illustrate the results and provide a PowerPoint presentation to share with your colleagues. Post this to the discussion.

Be sure that your code works!

Project Details:

Problem: create a simulator that implements the famous "Turtle Trading" strategy on any stock for any time frame and displays the results. The rules of the "Turle Trading" strategy (the original Trend Trading strategy) are:

1. each trading unit is 1% of your total investment dollars
2. enter at a stock's 55-day high with 1 unit
3. add another unit if the stock climbs to .5N (N is the Average True Range)
4. exit if the stock dips below latest entry price minus N

Import needed libraries

In [188]:

```
# Configure Jupyter so figures appear in the notebook
%matplotlib inline

# Configure Jupyter to display the assigned value after an assignment
%config InteractiveShell.ast_node_interactivity='last_expr_or_assign'

# import functions from the modsim.py module
from modsim import *
```

In [ ]:

```
try:
    import yfinance
except ImportError:
    !pip install yfinance
```

In [ ]:

```
try:
    import yahoofinancials
except ImportError:
    !pip install yahoofinancials
```

In [392]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
from yahoofinancials import YahooFinancials
```

Define and test functions for dataframe, state, and system objects

In [432]:

```
# create function that will create a dataframe for a single stock

def create_stock_df(stock, start, end, SMA):

    # get the data from yahoo finance
    df = yf.download(stock,
                     start=start,
                     end=end,
                     progress=False)

    # add extra columns for day, stock title,
    # simple moving average, and closing price average difference
    df['day'] = range(1, len(df) + 1)
    df['stock'] = stock
    df['SMA_x'] = df.iloc[:,4].rolling(window=SMA).mean()
    df['shifted_close'] = df['Close'].shift(1)
    df['close_difference'] = df['Close'] - df['shifted_close']

    # reset the index
    df = df.reset_index()

    # return the dataframe
    return df
```

In [436]:

```
# test create_stock_df function

google_df = create_stock_df('GOOG', '2014-01-01', '2020-7-10', 55)
google_df.head(3)
```

Out[436]:

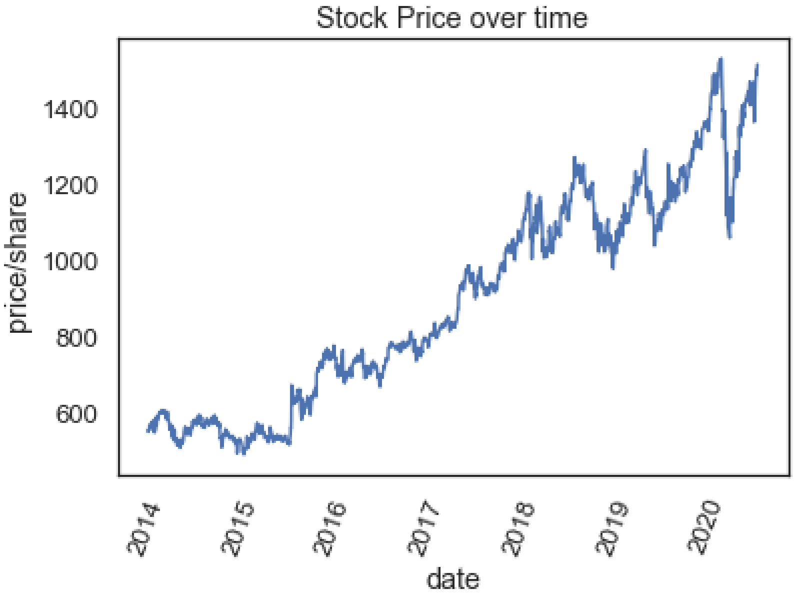
	Date	Open	High	Low	Close	Adj Close	Volume	day	stock	SM
0	2014-01-02	555.647278	556.788025	552.060730	554.481689	554.481689	3656400	1	GOOG	
1	2014-01-03	555.418152	556.379578	550.401978	550.436829	550.436829	3345800	2	GOOG	
2	2014-01-06	554.426880	557.340942	551.154114	556.573853	556.573853	3551800	3	GOOG	

In [437]:

```
# create a function to plot the stock dataframe's closing prices

def plot_stock_price(df):

    x = df['Date']
    y = df['Close']

    # plotting the points
    plt.plot(x, y)

    # naming the axes
    plt.xlabel('date')
    plt.ylabel('price/share')

    # rotate the tick marks
    plt.xticks(rotation=70)

    # title
    plt.title('Stock Price over time')

    # function to show the plot
    plt.show()
```

In [438]:

```
# test the plot_stock_price function

plot_stock_price(google_df)
```



In [441]:

```
# create a function that defines a state object
# for financial information that will change during simulation

# input your:
# 1. total dollars to invest
# 2. your entry signal in days
# 3. your exit signal (based on N, i.e. .5N)
def create_state_object(dollars, entry, exit):

    financial_state = State(dollars = dollars,
                            shares = 0,
                            total_value = dollars,
                            x_day_high = 0,
                            x_day_low = 0,
                            current_price = 0,
                            ATR = 0,
                            SMA_x = 0,
                            x = entry,
                            exit_x = exit,
                            status = 'out',
                            entry_price = 0,
                            exit_price = 0)

    return financial_state
```

In [442]:

```
# test the create_state_object

financial_state = create_state_object(100000, 55, 1)
```

Out[442]:

```
                 values
dollars          100000
shares                0
total_value      100000
x_day_high            0
x_day_low             0
current_price         0
ATR                   0
SMA_x                 0
x                    55
exit_x                1
status              out
entry_price           0
exit_price            0
```

In [526]:

```
# define a function that will create a system
# of parameters that will not change during the simulationcreate a system

def make_system(df, state, starting_dollars,
                unit_size, add_unit_signal):

    return System(t_0 = get_first_label(df),
                  t_end = get_last_label(df),
                  starting_dollars = starting_dollars,
                  unit_size = starting_dollars*unit_size,
                  add_unit_signal = add_unit_signal,
                  entry_signal = state.x,
                  exit_signal = state.exit_x,
                  stock = get_first_value(df['stock']),
                  financials = state)
```

In [444]:

```
# test the make_system function

system = make_system(google_df,
                     financial_state,
                     100000,
                     .01,
                     .5)
```

Out[444]:

	values
t_0	0
t_end	1640
starting_dollars	100000
unit_size	1000
add_unit_signal	0.5
entry_signal	55
exit_signal	1
stock	GOOG
financials	dollars 100000 shares ...

Define update function and simulator function

In [540]:

```
# The update function takes the state during the current time step
# and returns the state during the next time step.

def update_func(df, state, t, system):

    d = state.dollars
    shares = state.shares
    #current_price = state.current_price
    x = state.x
    exit_x = state.exit_x
    status = state.status
    entry_price = state.entry_price
    exit_price = state.exit_price
    add_unit_signal = system.add_unit_signal

    if t <= x+2:

        xdh = max(df['Close'][1:x])
        xdl = min(df['Close'][1:x])
        sma_x = df['SMA_x'][t]
        atr = (xdh - xdl)/1.5
        current_price = df['Close'][t]

    if t > x+2:

        xdh = max(df['Close'][t-x:t+1])
        xdl = min(df['Close'][t-x:t+1])
        sma_x = df['SMA_x'][t]
        atr = (xdh - xdl)/1.5
        current_price = df['Close'][t]

        # if you see the entry signal and you're out
        if current_price >= xdh and status == 'out':

            entry_price = current_price
            shares = (system.unit_size)//(entry_price)
            d = d - ((system.unit_size)//(entry_price)) * entry_price
            status = 'in'

        # if you see the add unit signal and you're already in
        elif (status == 'in') and (current_price > (entry_price + (atr*add_uni
t_signal))) and (d > current_price):

            entry_price = current_price
            shares = shares + (system.unit_size)//(entry_price)
            d = d - ((system.unit_size)//(entry_price)) * entry_price
            status = 'in'

        # if you're in and you see the exit signal
        elif (current_price < (sma_x - (atr*exit_x))) and (status == 'in'):

            exit_price = current_price
            d = d + (shares * exit_price)
            shares = 0
            status = 'out'
        # you're just cruisin
        else:

            entry_price = entry_price
            exit_price = exit_price
            shares = shares
            d = d

    return State(dollars = d,
                 shares = shares,
                 total_value = d + (shares*current_price),
                 x_day_high = xdh,
                 x_day_low = xdl,
                 current_price = current_price,
                 ATR = atr,
                 SMA_x = sma_x,
                 x = x,
                 exit_x = exit_x,
                 status = status,
                 entry_price = entry_price,
                 exit_price = exit_price)
```

In [501]:

```
# test update_func

#state = update_func(google_df, financial_state, 72, system)
```

In [502]:

```
# test update_func again

#state = update_func(google_df, state, 1005, system)
```

In [503]:

```
# define run simulation function that stores results in a TimeFrame

def run_simulation(df, system, update_func):

    # create a TimeFrame to keep track of financials over time
    frame = TimeFrame(columns = system.financials.index)
    frame.row[system.t_0] = system.financials

    # run the simluation for every day in the date range
    for t in linrange(system.t_0, system.t_end-(state.x+1)):
        frame.row[t+1] = update_func(df, frame.row[t], t, system)

    return frame
```

In [504]:

```
# test the run_simulation function

results = run_simulation(google_df, system, update_func)
```

Out[504]:

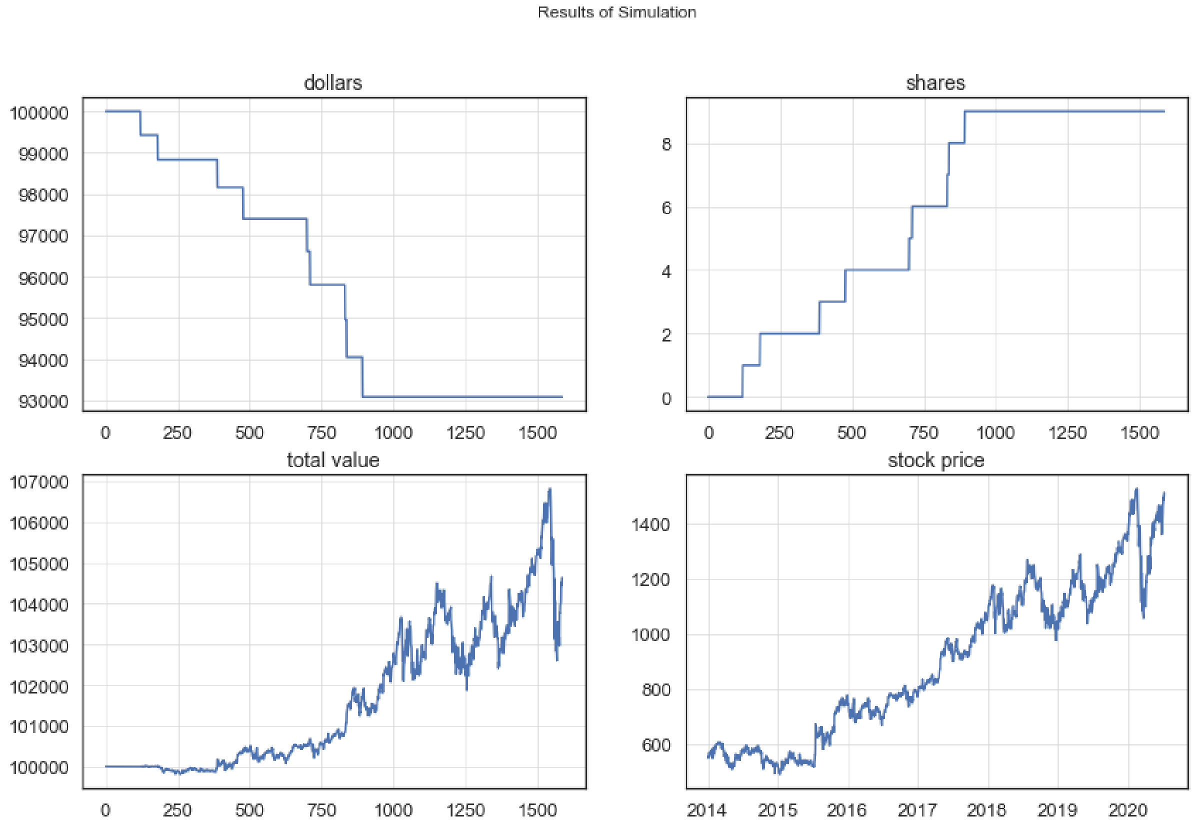
	dollars	shares	total_value	x_day_high	x_day_low	current_price	ATR	SMA_x	x
0	100000	0	100000	0	0	0	0	0	55
1	100000	0	100000	607.807	548.559	554.482	59.2479	NaN	55
2	100000	0	100000	607.807	548.559	550.437	59.2479	NaN	55
3	100000	0	100000	607.807	548.559	556.574	59.2479	NaN	55
4	100000	0	100000	607.807	548.559	567.304	59.2479	NaN	55
...	...	...	...	...	...	...	...	...	...
1580	93086.9	9	104045	1526.69	1056.62	1217.56	470.07	1309.12	55
1581	93086.9	9	104510	1526.69	1056.62	1269.23	470.07	1305.53	55
1582	93086.9	9	104449	1526.69	1056.62	1262.47	470.07	1302.41	55
1583	93086.9	9	104458	1526.69	1056.62	1263.47	470.07	1298.98	55
1584	93086.9	9	104636	1526.69	1056.62	1283.25	470.07	1295.79	55

1585 rows × 13 columns

In [551]:

```
# create a function to plot the results

def plot_results(results, df):

    # call for four plots
    fig, axs = plt.subplots(2, 2, figsize = (14,9))

    # add a title to the figure
    fig.suptitle("Results of Simulation")

    # setup top left plot
    axs[0, 0].plot(results.index, results.dollars)
    axs[0, 0].set_title('dollars')
    axs[0, 0].grid(True, alpha = 0.5)

    # setup top right plot
    axs[0, 1].plot(results.index, results.shares)
    axs[0, 1].set_title('shares')
    axs[0, 1].grid(True, alpha = 0.5)

    # setup bottom left plot
    axs[1, 0].plot(results.index, results.total_value)
    axs[1, 0].set_title('total value')
    axs[1, 0].grid(True, alpha = 0.5)

    # setup bottom right plot
    axs[1, 1].plot(df['Date'], df['Close'])
    axs[1, 1].set_title('stock price')
    axs[1, 1].grid(True, alpha = 0.5)

    # rotate tick marks of final plot
    plt.xticks(rotation=45)
    plt.show()

    # print beginning and ending values
    print("initial investment:", get_first_value(results.total_value))
    print("current total investment value:", round(get_last_value(results.tota
l_value), 2))
```

In [509]:
```
plot_results(results, google_df)
```



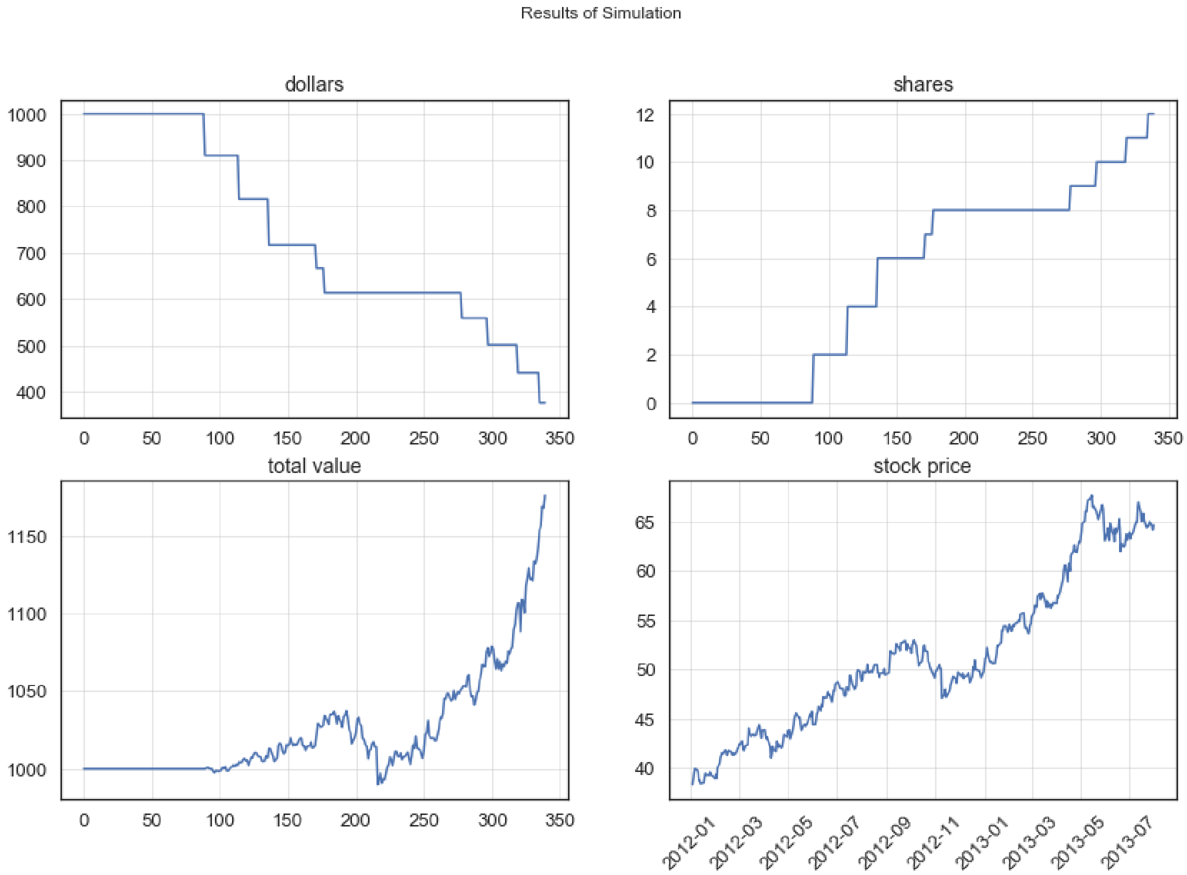
In [403]:
```
# export results to a CSV

#results.to_csv(r'test_results2.csv', index = False)
```

Create a function to run simulation for end user

In [545]:

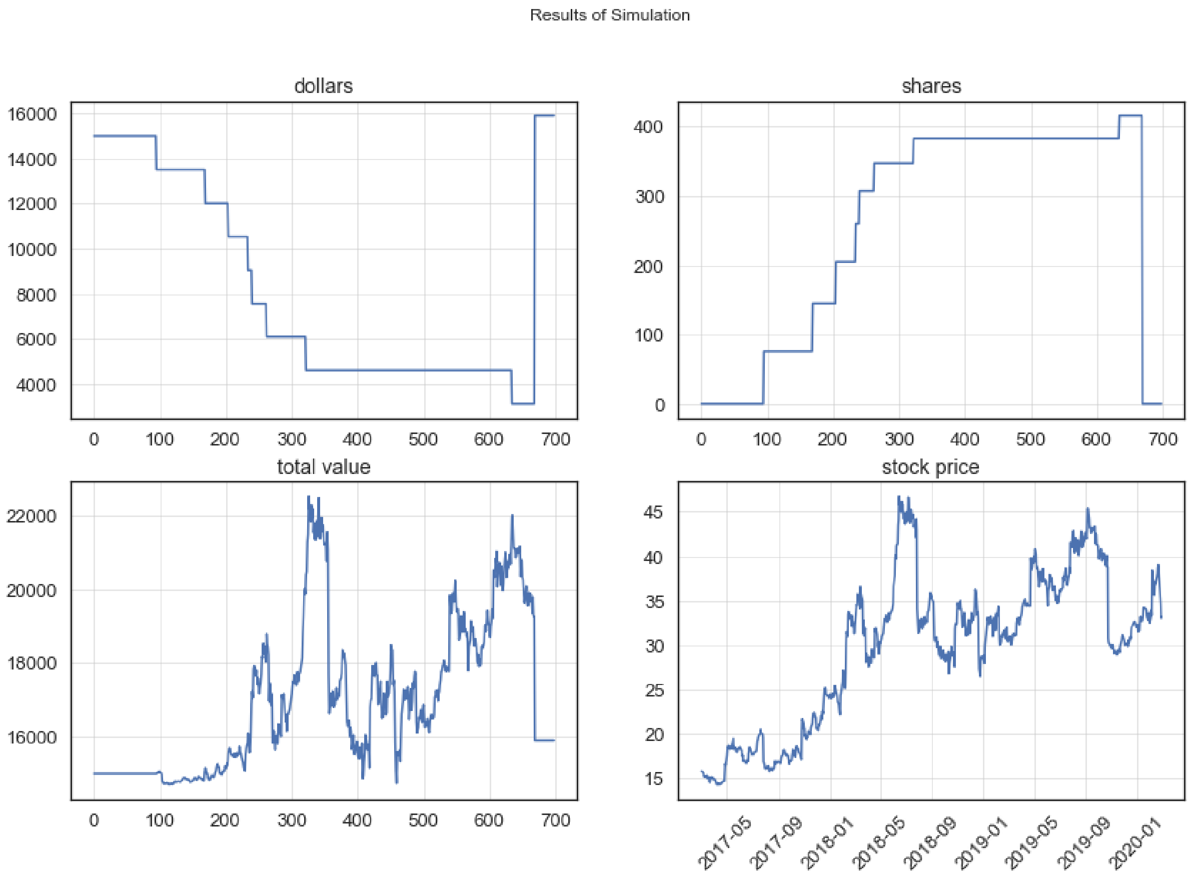
```
# finally, create a function for the end user that will take the parameters:
# 1. stock
# 2. date range
# 3. total investment dollars
# 4. entry signal
# 5. exit signal
# 6. unit size
# 7. add unit signal
# 8. simple moving average length

# and the function will run the functions
# 1. create_stock_df
# 2. create_state_object
# 3. make_system
# 4. run_simulation
# 5. plot_results

def trend_trader_simulator(stock = 'GOOG', start_date = '2014-01-01',
                           end_date = '2020-02-01', investment_dollars = 50000
,
                           entry_signal = 55,
                           exit_signal = 1, unit_size = 0.1,
                           add_unit_signal = .5, update_function = update_func
):

    # create stock dataframe
    TT_df = create_stock_df(stock, start_date, end_date, entry_signal)

    # create financial state object
    TT_financial_state = create_state_object(investment_dollars, entry_signal,
exit_signal)

    # create system object
    TT_system = make_system(TT_df, TT_financial_state, investment_dollars,
                         unit_size, add_unit_signal)

    # run the simulation
    TT_results = run_simulation(TT_df, TT_system, update_function)

    # plot the results
    plot_results(results = TT_results, df = TT_df)
```

```
In [553]: trend_trader_simulator(stock = 'DIS',
                                 start_date = '2012-01-01',
                                 end_date = '2013-08-01',
                                 investment_dollars = 1000,
                                 update_function = update_func)
```



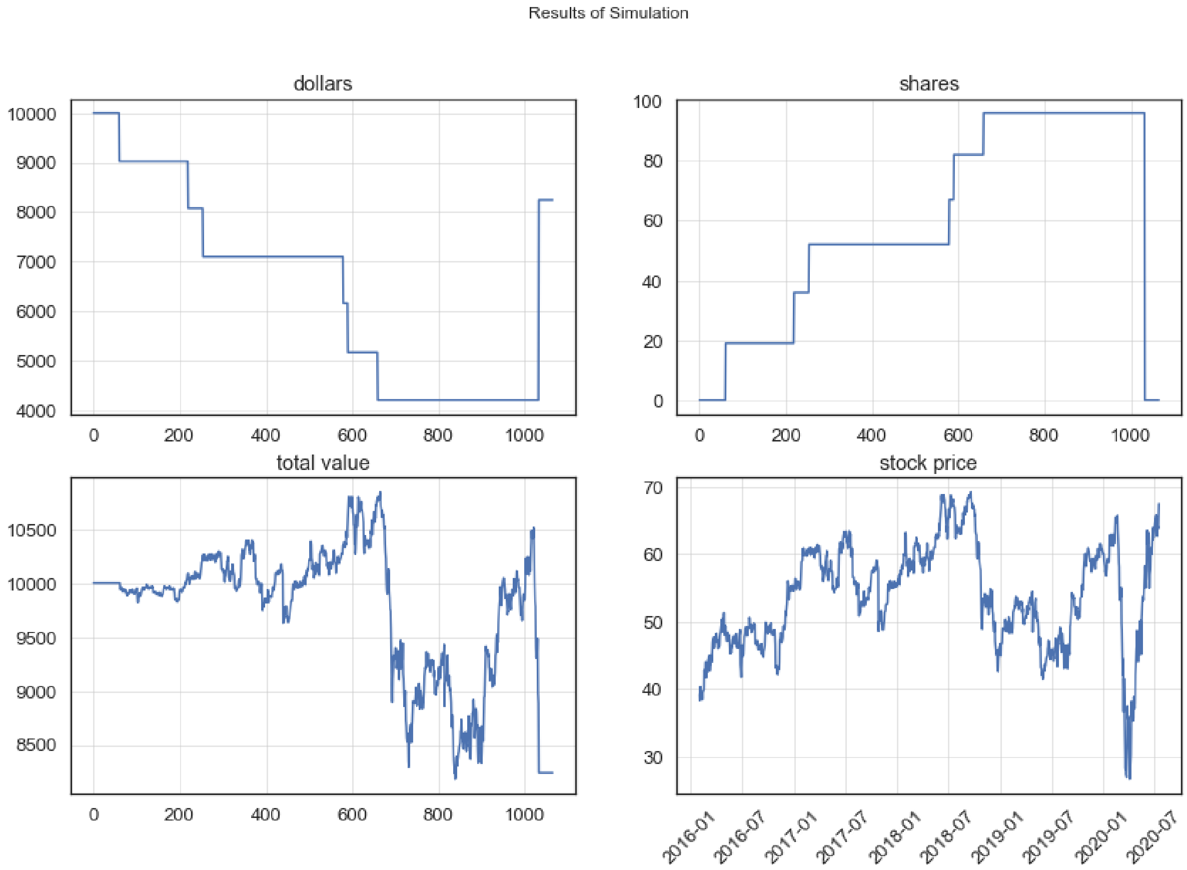
```
initial investment: 1000
current total investment value: 1175.79
```

In [554]:
```
trend_trader_simulator(stock = 'TWTR',
                       start_date = '2017-03-01',
                       end_date = '2020-3-01',
                       investment_dollars = 15000,
                       update_function = update_func)
```



```
initial investment: 15000
current total investment value: 15906.51
```

```
In [569]: trend_trader_simulator(stock = 'BC',
                                 start_date = '2016-02-01',
                                 end_date = '2020-7-16',
                                 investment_dollars = 10000,
                                 update_function = update_func)
```



```
initial investment: 10000
current total investment value: 8241.09
```

Test simulator on many stocks

In [558]:

```
def trend_trader_aggregater(stock = 'GOOG', start_date = '2014-01-01',
                           end_date = '2020-02-01', investment_dollars = 50000
,
                           entry_signal = 55,
                           exit_signal = 1, unit_size = 0.1,
                           add_unit_signal = .5, update_function = update_func
):

    # create stock dataframe
    TT_df = create_stock_df(stock, start_date, end_date, entry_signal)

    # create financial state object
    TT_financial_state = create_state_object(investment_dollars, entry_signal,
exit_signal)

    # create system object
    TT_system = make_system(TT_df, TT_financial_state, investment_dollars,
                         unit_size, add_unit_signal)

    # run the simulation
    TT_results = run_simulation(TT_df, TT_system, update_function)

    # plot the results
    #plot_results(results = TT_results, df = TT_df)
    return round(get_last_value(TT_results.total_value), 2)
```

In [559]:

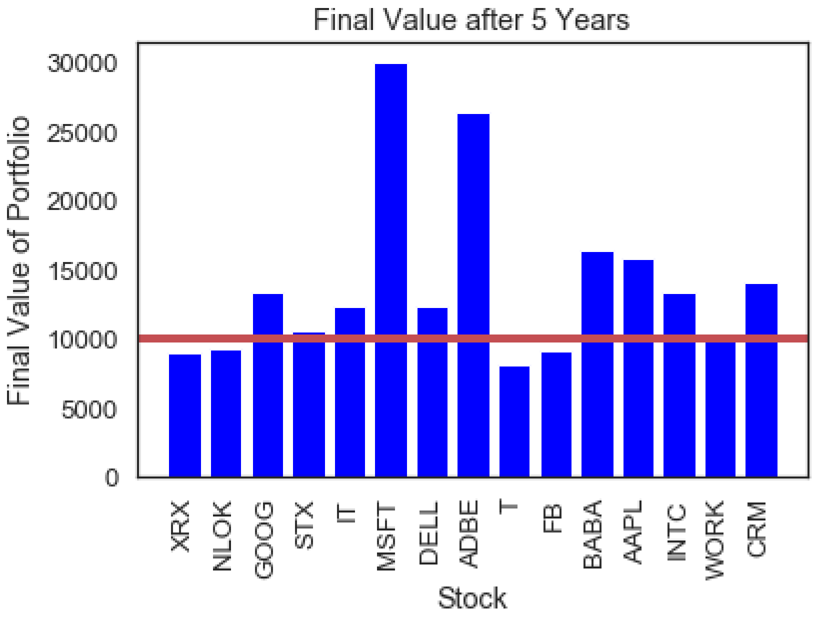
```
tech_stocks = ['XRX', 'NLOK', 'GOOG', 'STX', 'IT',
               'MSFT', 'DELL', 'ADBE', 'T', 'FB',
               'BABA', 'AAPL', 'INTC', 'WORK', 'CRM']

final_value = []

for t in tech_stocks:

    value = trend_trader_aggregater(stock = t,
                       start_date = '2015-01-01',
                       end_date = '2020-7-15',
                       investment_dollars = 10000,
                       update_function = update_func)

    final_value.append(value)

final_value
```

Out[559]:

```
[8996.27,
 9267.7,
 13356.98,
 10572.96,
 12313.48,
 30057.9,
 12436.81,
 26421.21,
 8206.05,
 9165.81,
 16506.07,
 15912.38,
 13464.1,
 9985.94,
 14107.73]
```

In [570]:

```
tech_stocks_df = pd.DataFrame({'Stock':tech_stocks,'Day':final_value})
#tech_stocks_df

plt.figsize = (14,9)


plt.bar(tech_stocks, final_value, color='blue')
plt.xlabel("Stock")
plt.ylabel("Final Value of Portfolio")
plt.title("Final Value after 5 Years")

plt.xticks(rotation=90)

plt.axhline(y=10000,linewidth=4, color='r')

plt.show()
```



(Future State) Create a Sweep Simulation function

In [430]:

```
def sweep_entry_signal_simulation(df, state, update_func):

    sweep_entry_signal = SweepSeries()

    for i in linrange(10, 310, 50):
        state = create_state_object(state.dollars, i, state.exit_x)
        system = make_system(df,
                             state,
                             100000,
                             .02,
                             .5)
        results = run_simulation(df, system, update_func)
        sweep_entry_signal[i] = round(get_last_value(results.total_value), 2)

    return sweep_entry_signal
```

In [431]:

```
# test sweep function

sweep_results = sweep_entry_signal_simulation(google_df, financial_state, update_func)
```

Out[431]:

	values
10	100088.37
60	100088.37
110	100088.37
160	100088.37
210	100088.37
260	100088.37