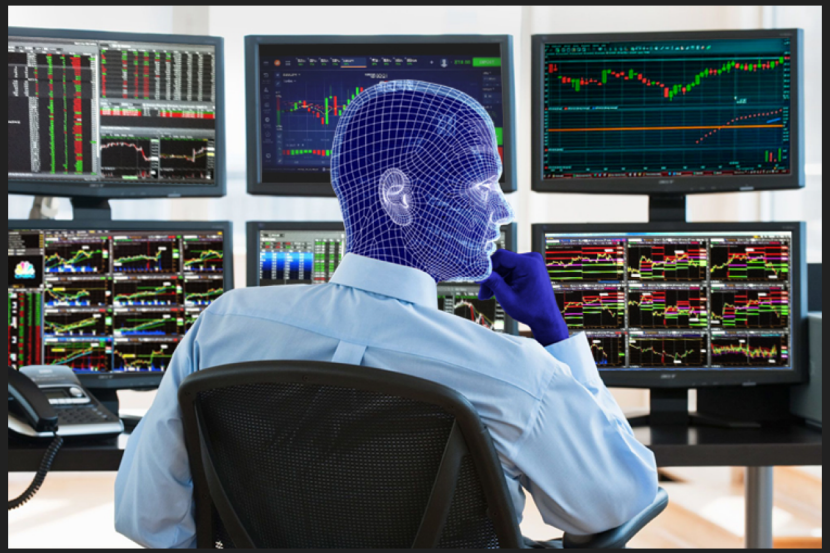
Exploiting Wall Street

An algorithmic approach to investing

Goldman Sachs Fired 99% of Traders And Replaced Them With Robots

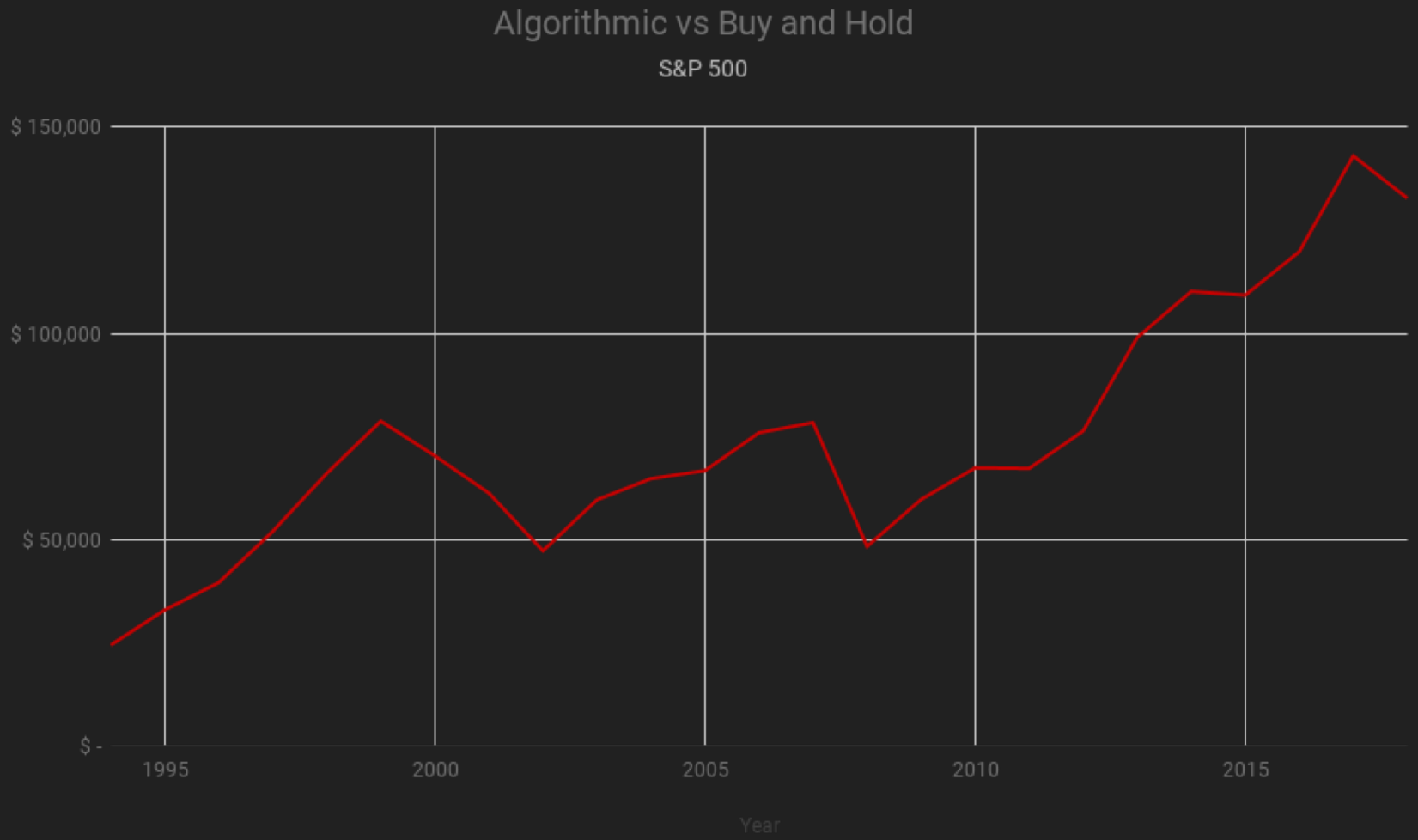


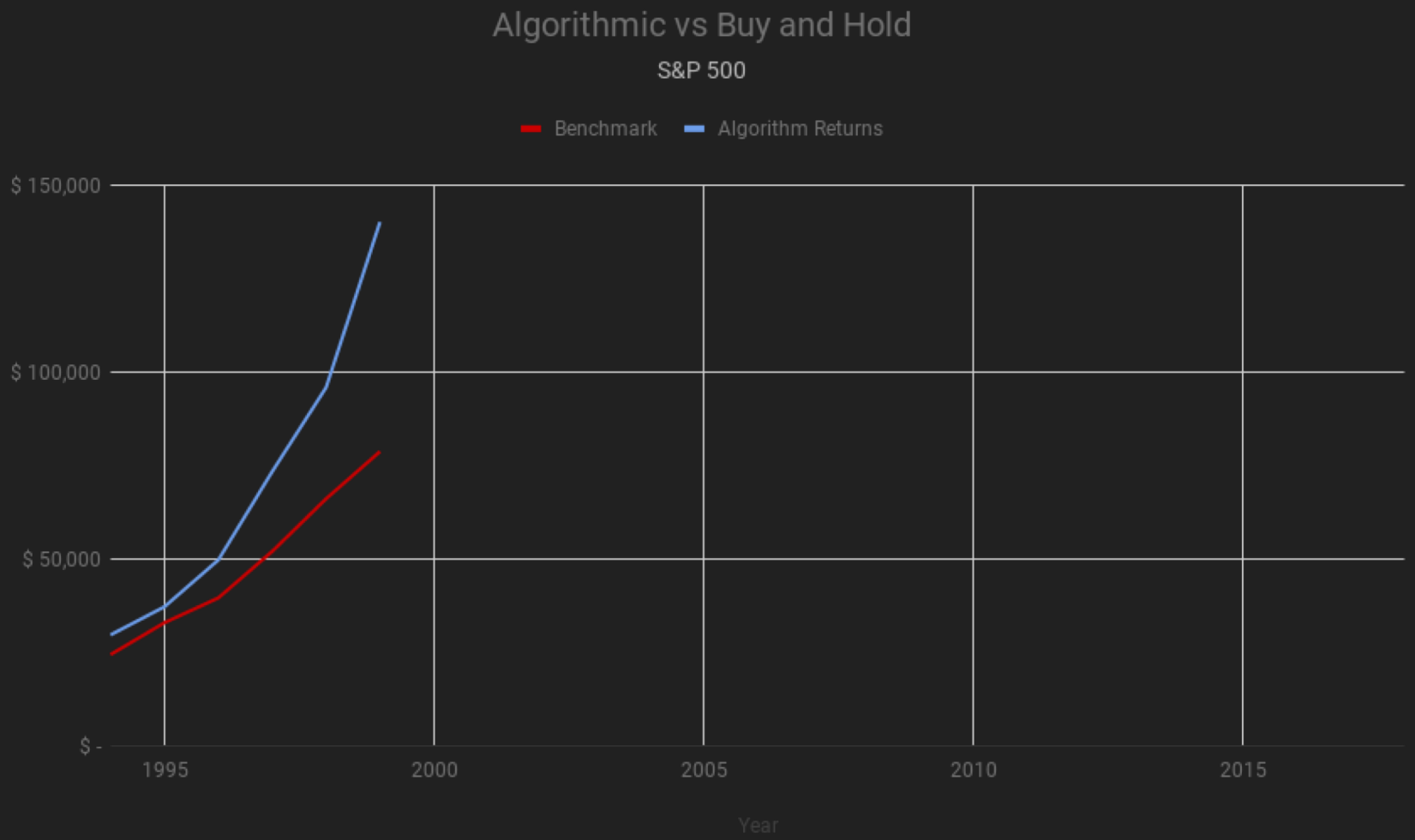
What disadvantages do you have as an investor?

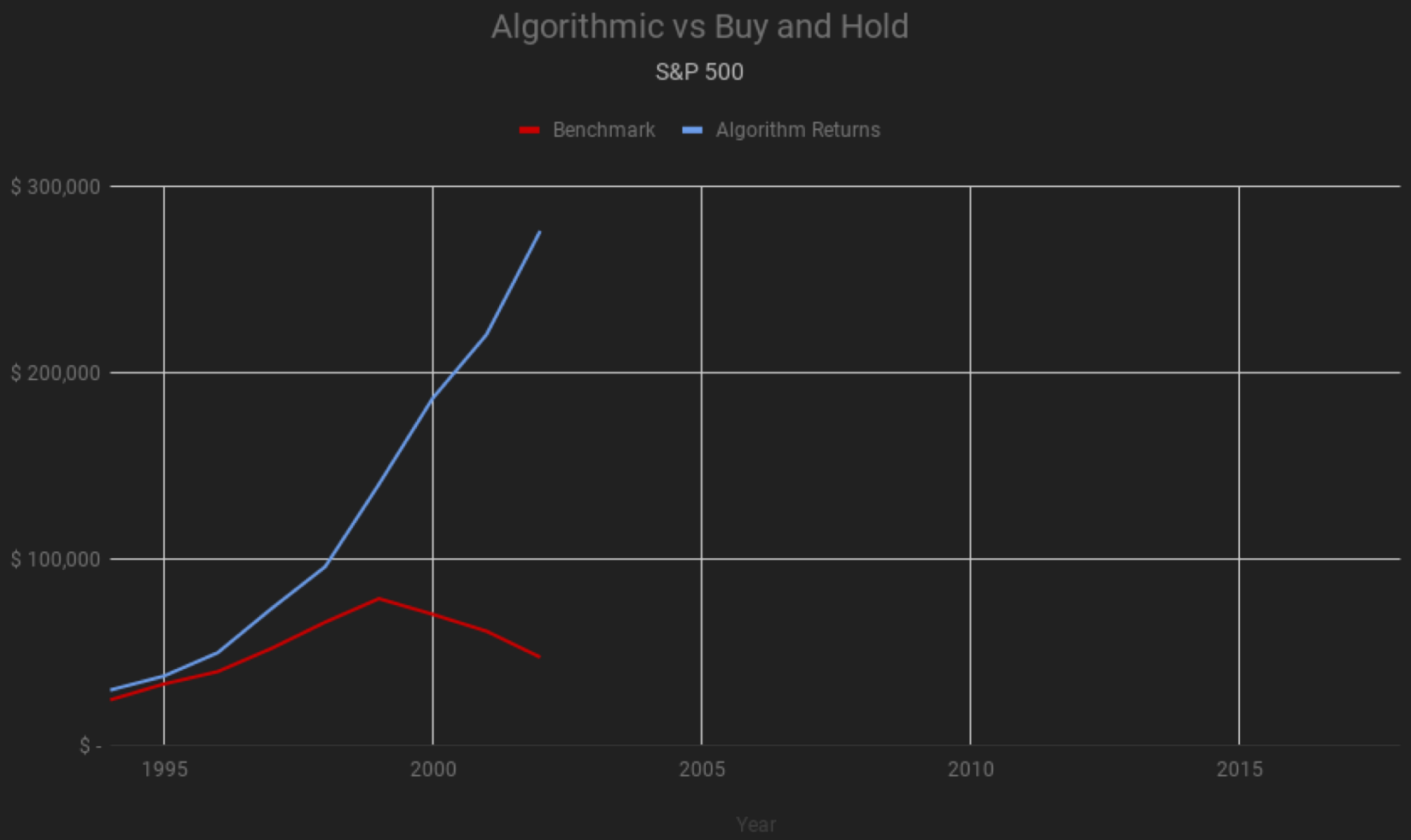
Beating the market: Everybody tries to do beat it, but few succeed.

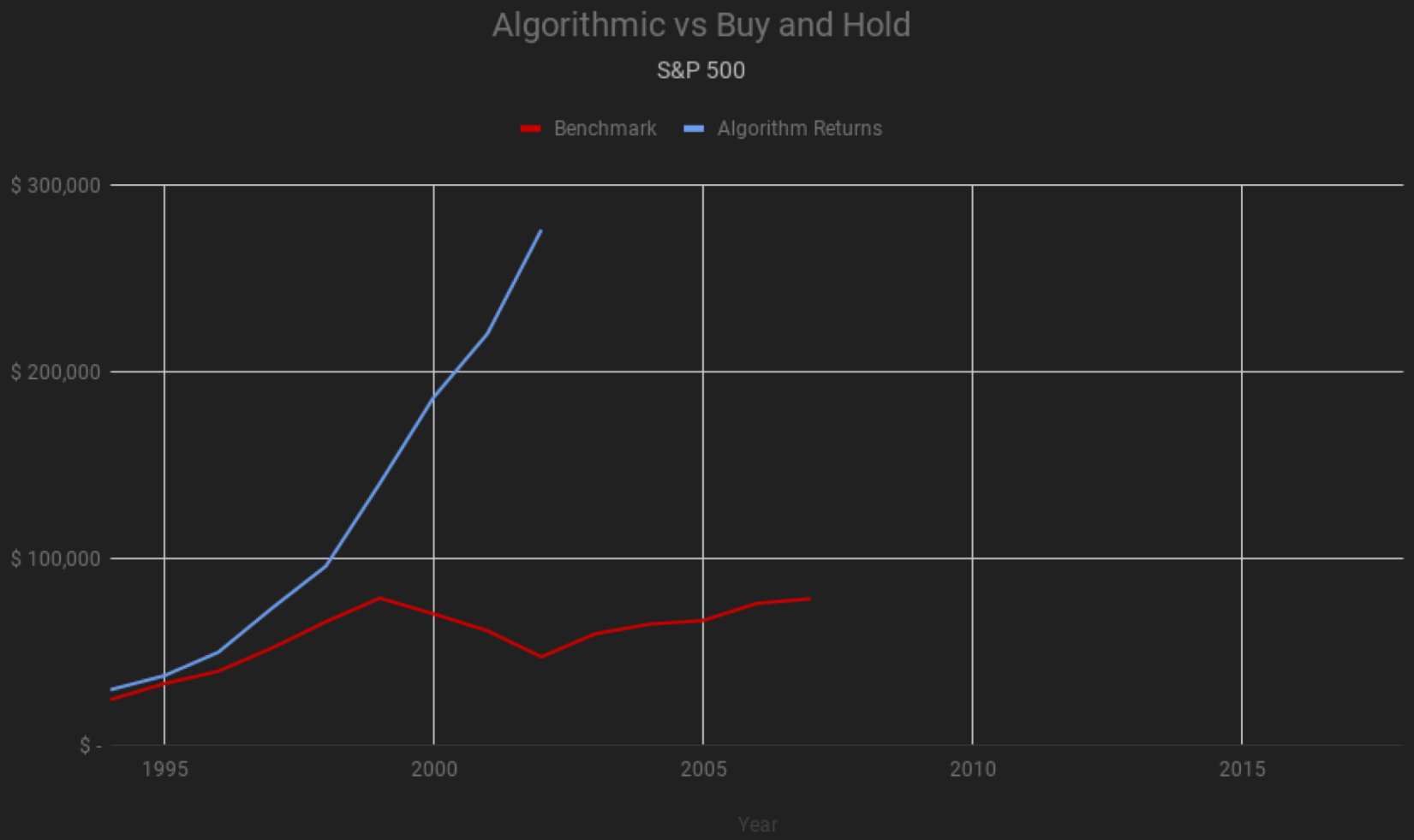
“Yes, you may be able to beat the market, but with investment fees, taxes, and human emotion working against you, you're more likely to do so through luck than skill. If you can merely **match** the S&P 500, minus a small fee, you'll be doing better than most investors.”

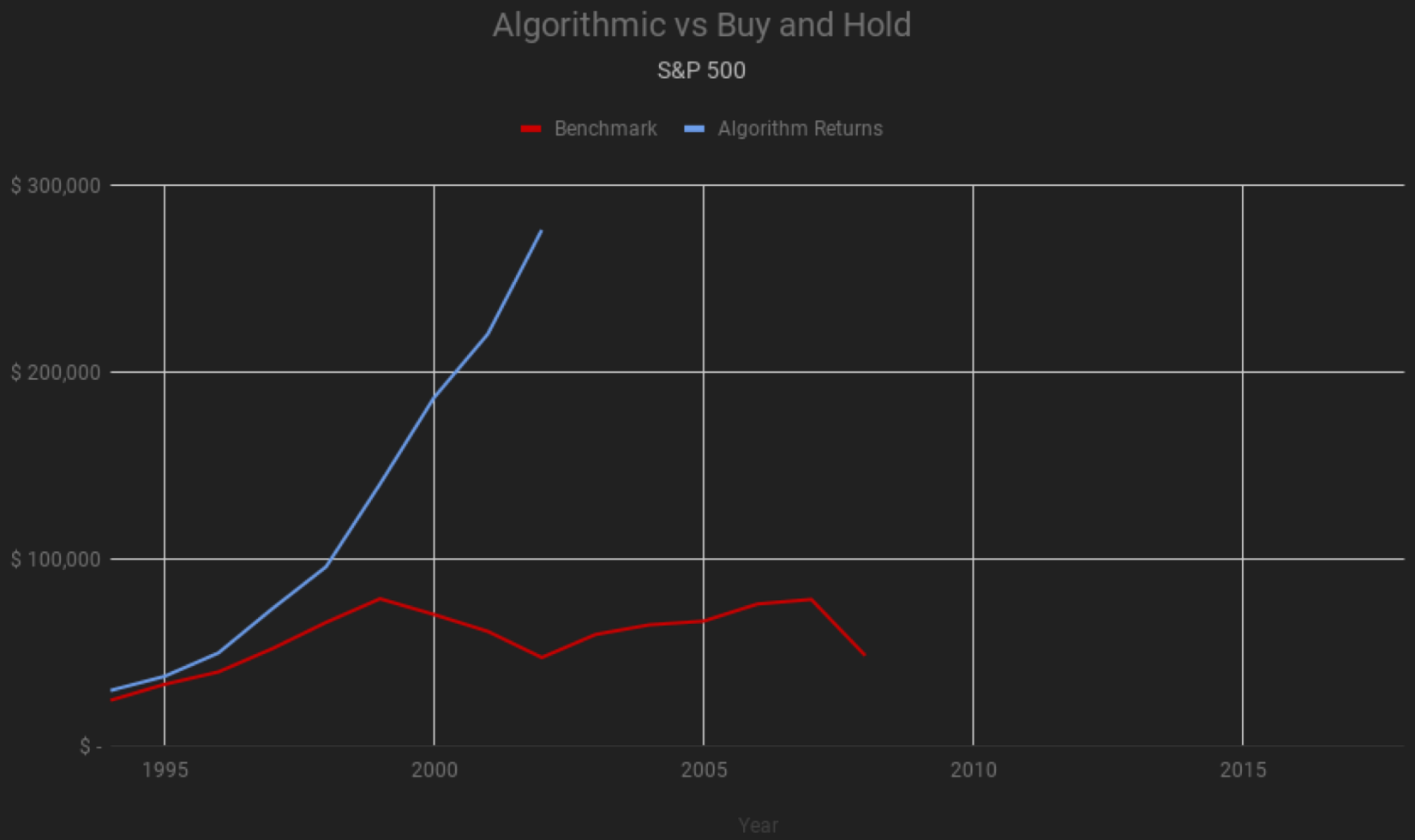
-investopedia

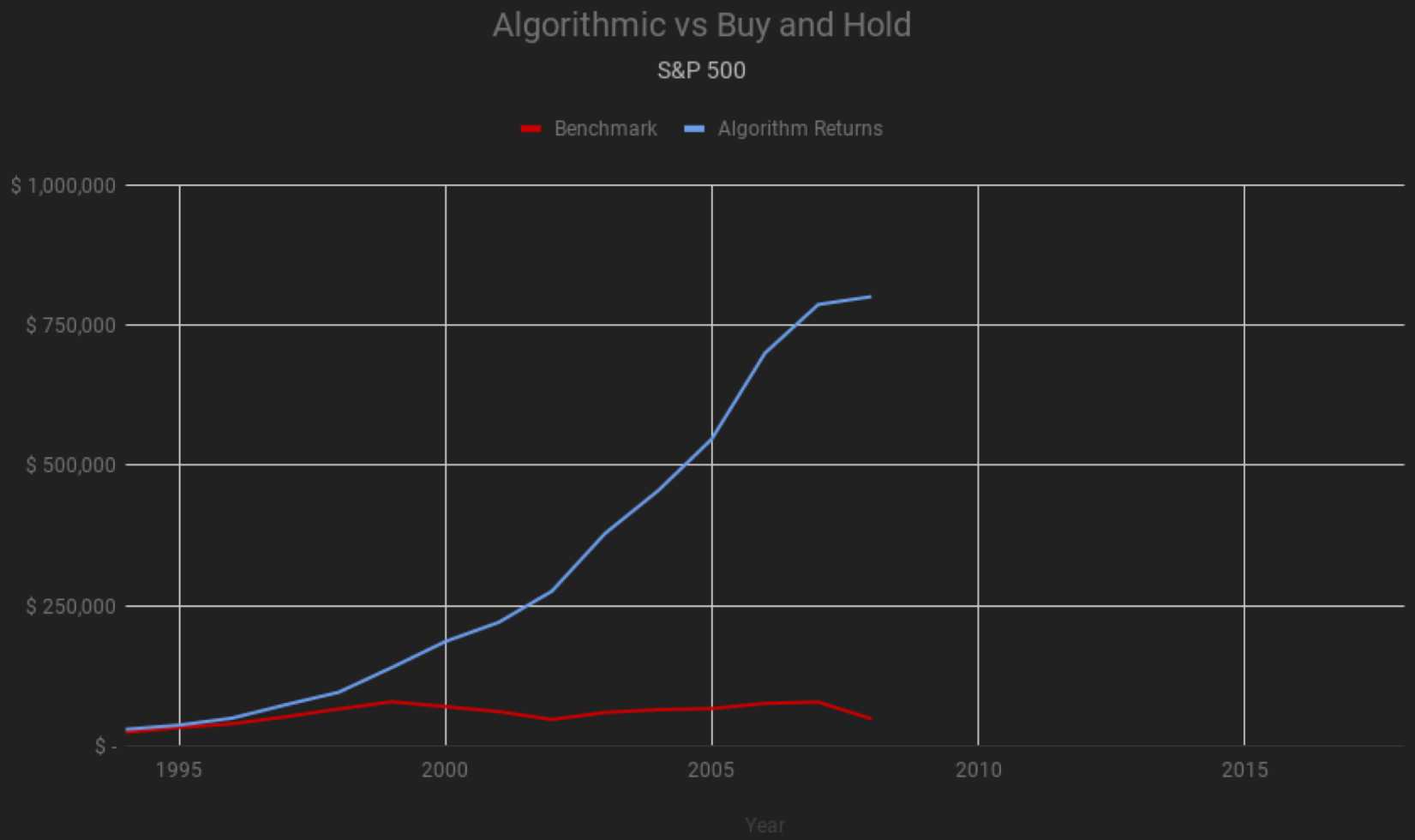

Algorithmic vs Buy and Hold
S&P 500
$ 150,000
$ 100,000
$ 50,000
$ -
1995
2000
2005
2010
2015
Year

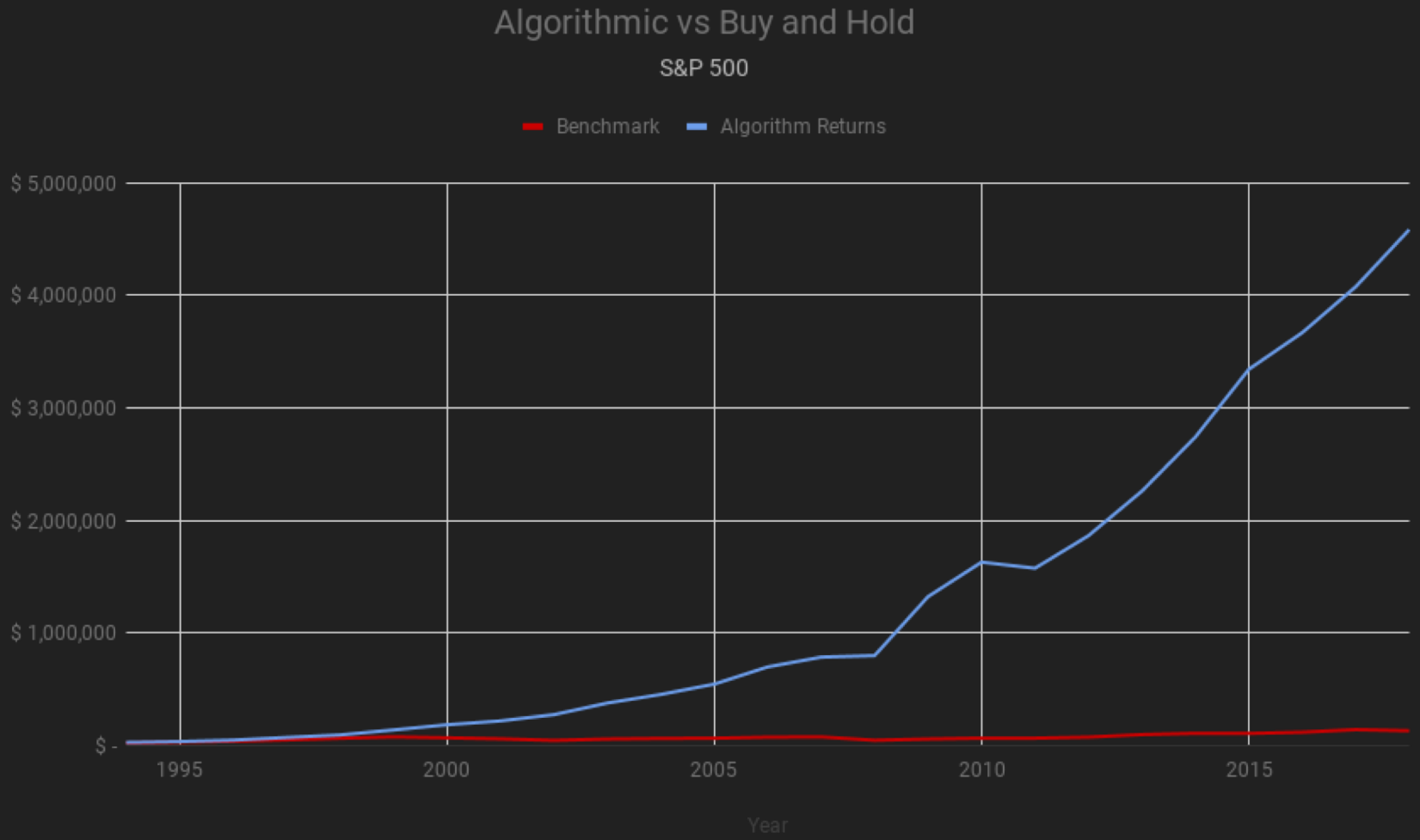

Algorithmic vs Buy and Hold
S&P 500
Benchmark
Algorithm Returns
$ 150,000
$ 100,000
$ 50,000
$ -
1995
2000
2005
2010
2015
Year

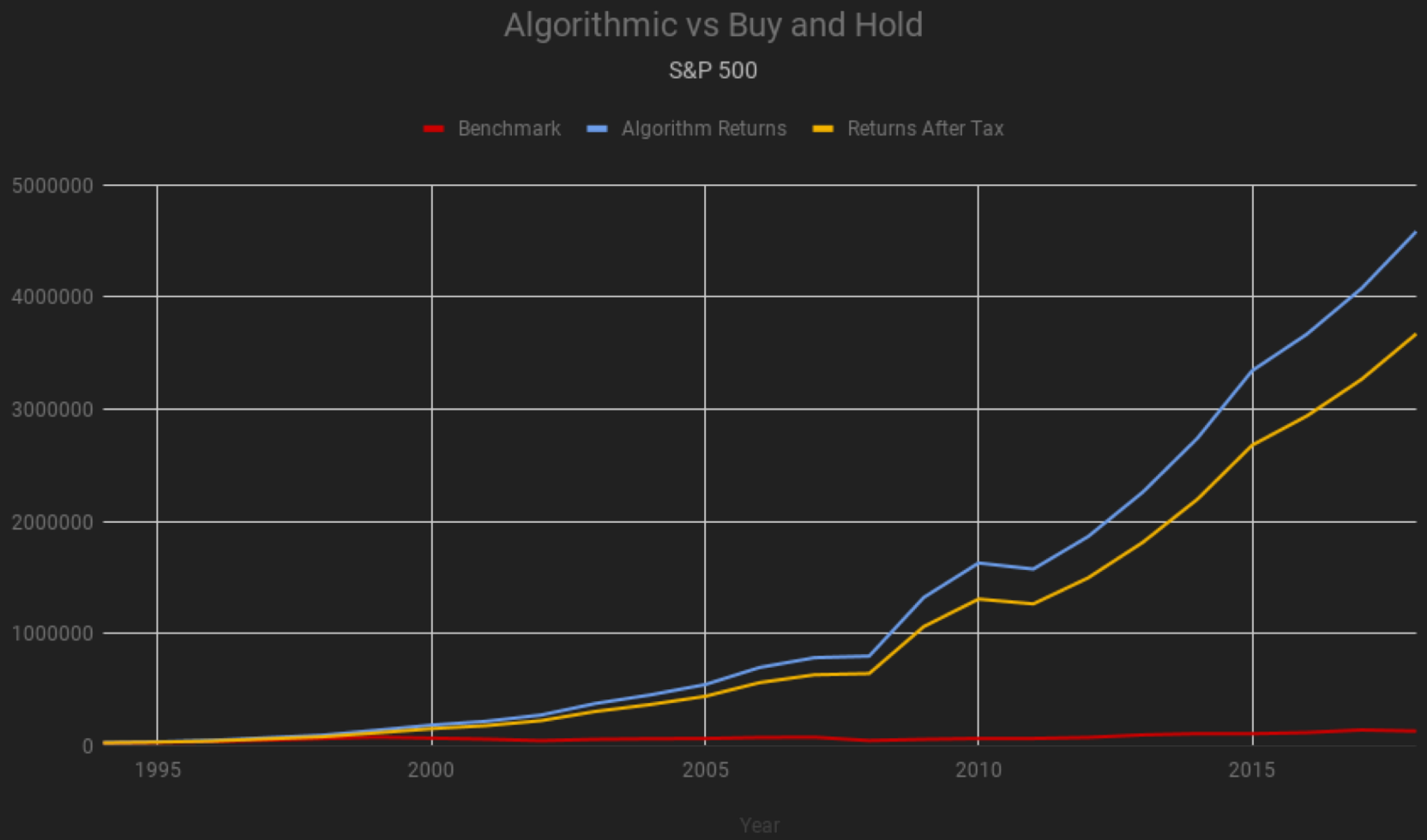

Algorithmic vs Buy and Hold
S&P 500
Benchmark
Algorithm Returns
$ 300,000
$ 200,000
$ 100,000
$ -
1995
2000
2005
2010
2015
Year

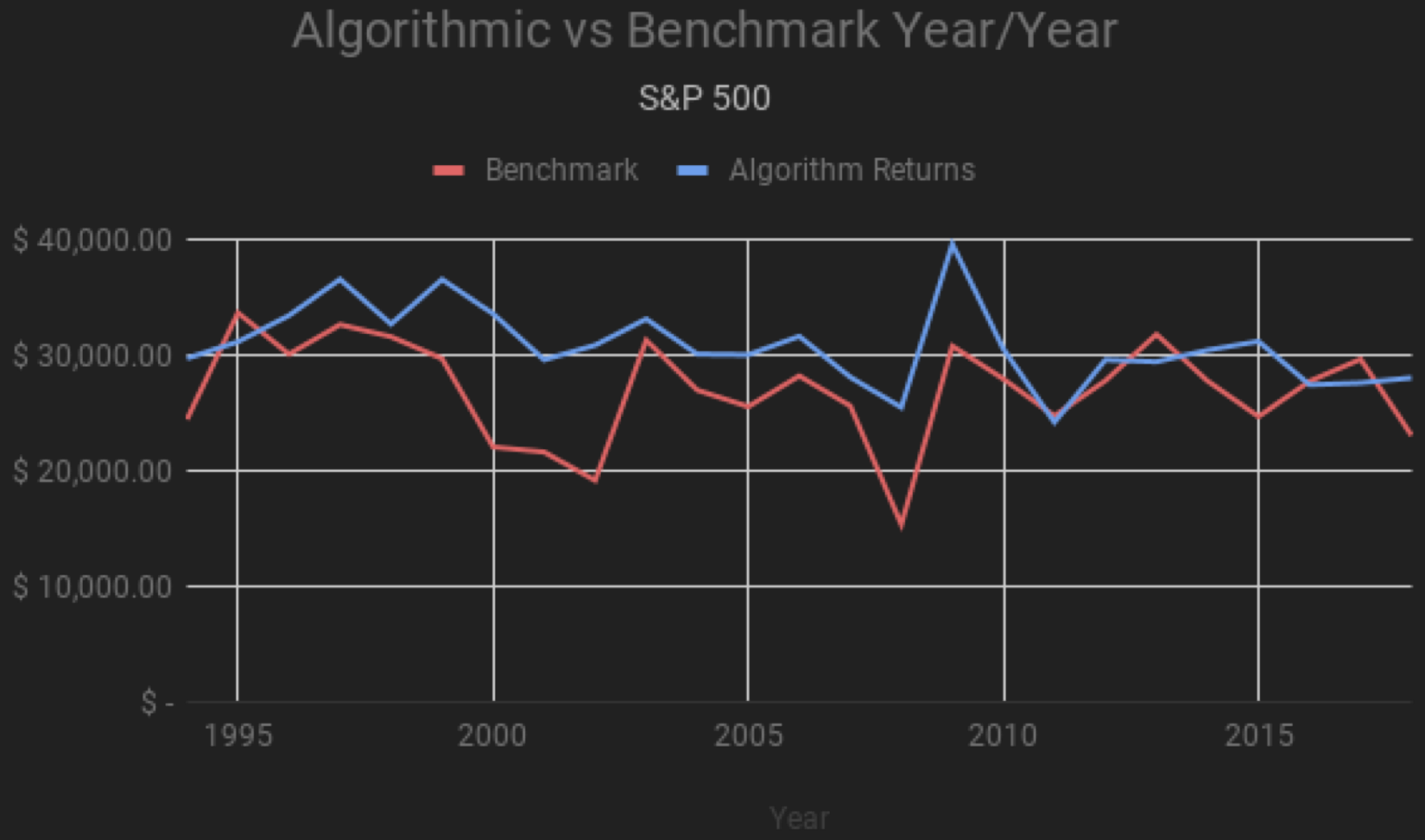

Algorithmic vs Buy and Hold
S&P 500
Benchmark
Algorithm Returns
$ 300,000
$ 200,000
$ 100,000
$ -
1995
2000
2005
2010
2015
Year

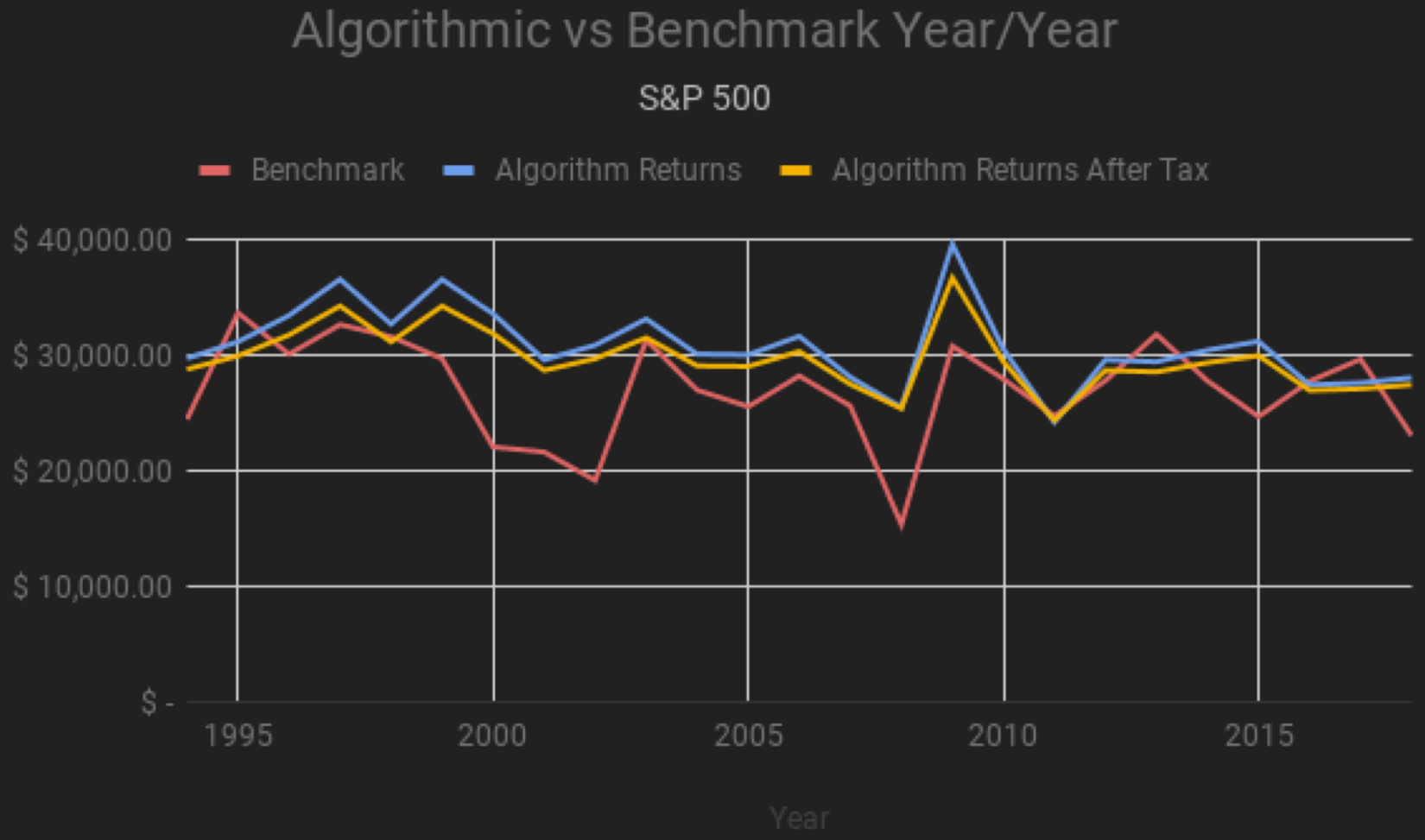

Algorithmic vs Buy and Hold
S&P 500
Benchmark
Algorithm Returns
$ 300,000
$ 200,000
$ 100,000
$ -
1995
2000
2005
2010
2015
Year


Algorithmic vs Buy and Hold
S&P 500
Benchmark
Algorithm Returns
$ 1,000,000
$ 750,000
$ 500,000
$ 250,000
$ -
1995
2000
2005
2010
2015
Year

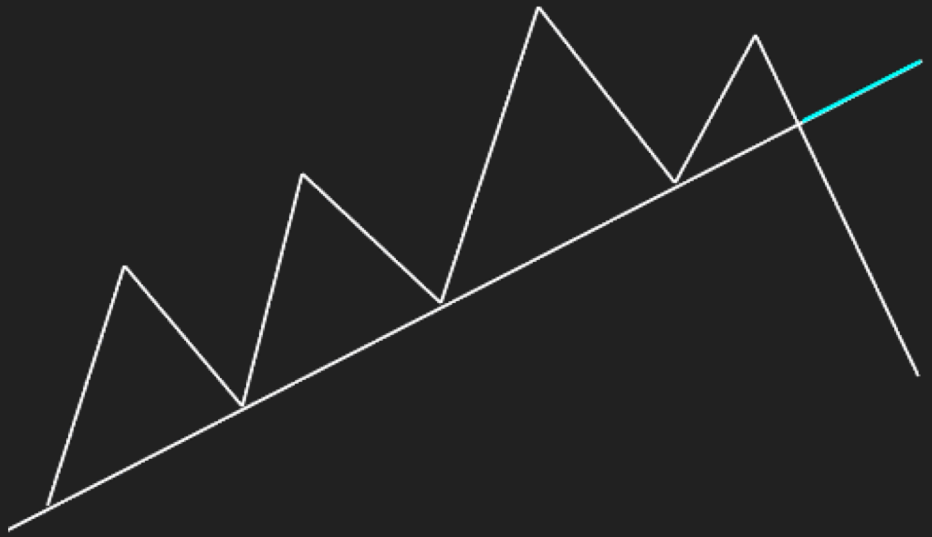

Algorithmic vs Buy and Hold
S&P 500
Benchmark
Algorithm Returns
$ 5,000,000
$ 4,000,000
$ 3,000,000
$ 2,000,000
$ 1,000,000
$ -
1995
2000
2005
2010
2015
Year


Algorithmic vs Buy and Hold
S&P 500
Benchmark
Algorithm Returns
Returns After Tax
5000000
4000000
3000000
2000000
1000000
0
1995
2000
2005
2010
2015
Year


Algorithmic vs Benchmark Year/Year
S&P 500
Benchmark
Algorithm Returns
$ 40,000.00
$ 30,000.00
$ 20,000.00
$ 10,000.00
$ -
1995
2000
2005
2010
2015
Year


Algorithmic vs Benchmark Year/Year
S&P 500
Benchmark
Algorithm Returns
Algorithm Returns After Tax
$ 40,000.00
$ 30,000.00
$ 20,000.00
$ 10,000.00
$ -
1995
2000
2005
2010
2015
Year

The Algorithm



```
elif (signal['action'] == "sell" and positionPrice > 0.0):
    cash += todayCandle['close']*positionSize
    positionSize = 0
    positionPrice = 0.0
    if (lastTradePrice < todayCandle['close']):
        winningTrades += 1
    else:
        losingTrades += 1
    lastTradePrice = todayCandle['close']
elif (signal['action'] == "buy" and positionPrice == 0.0):
    positionPrice = todayCandle['close']
    positionSize = int(cash/positionPrice)
    cash -= positionSize*positionPrice
    if (lastTradePrice > todayCandle['close']):
        winningTrades += 1
    else:
        losingTrades += 1
    lastTradePrice = todayCandle['close']
```

```python
class Event(object):
    pass

class MarketEvent(Event):
    ''' Handles the event of recieving a
    new market order update with corresponding bars
    '''
    def __init__(self):
        self.type = 'MARKET'

class SignalEvent(Event):
    def __init__(self, symbol, datetime, signalType, signalStrength):
        self.type = 'SIGNAL'
        self.symbol = symbol
        self.datetime = datetime
        self.signalType = signalType
        self.signalStrength = signalStrength

class OrderEvent(Event):
    def __init__(self, symbol, orderType, quantity, direction, stop = None, stop_type = None):
        self.type = 'ORDER'
        self.symbol = symbol
        self.orderType = orderType
        self.quantity = quantity
        self.direction = direction
        self.stop = stop
        self.stop_type = stop_type

    def printOrder(self):
        print("Order : Symbol= %s, Type= %s, Quantity= %s, Direction= %s" % (self.symbol, self.orderType, self.quantity, self.direction))

class FillEvent(Event):
    def __init__(self, timeindex, symbol, exchange, quantity, direction, fill_cost, commission=None):
        self.type = 'FILL'
        self.timeindex = timeindex
        self.symbol = symbol
        self.exchange = exchange
        self.quantity = quantity
        self.direction = direction
        self.fill_cost = fill_cost #holding value in dollars
        self.commission = commission #should never be anything if using alpaca
```

The Road Ahead

Questions?

Appendix

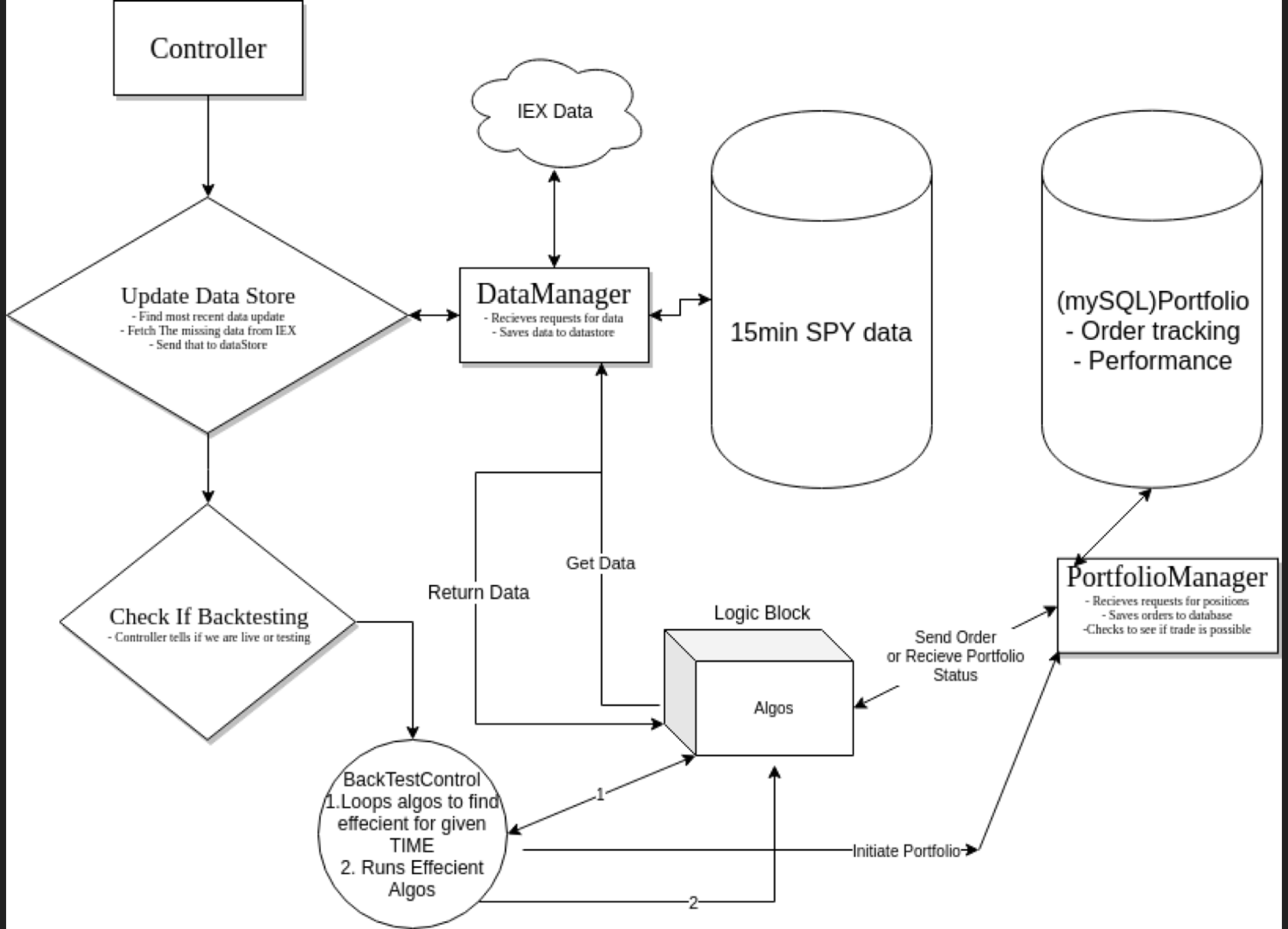
Year	Benchmark	Algorithm Returns	Algorithm Returns A	Benchmark Returns	AlgoReturns	Algo vs Benchmark	Winning Trades	Losing Trades	Total Trades	Max DrawDown	Accuracy
1994	$ 24,421.50	$ 29,691.09	$ 28,752.88	-2.31%	18.76%	21.08%	50	42	92	-2.55%	54.35%
1995	$ 33,631.91	$ 31,130.68	$ 29,904.54	34.53%	24.52%	-10.00%	49	57	106	-0.85%	46.23%
1996	$ 30,054.38	$ 33,398.58	$ 31,718.86	20.22%	33.59%	13.38%	51	51	102	-3.04%	50.00%
1997	$ 32,613.00	$ 36,533.01	$ 34,226.41	30.45%	46.13%	15.68%	52	52	104	-3.56%	50.00%
1998	$ 31,568.00	$ 32,654.69	$ 31,123.75	26.27%	30.62%	4.35%	52	66	118	-5.02%	44.07%
1999	$ 29,668.75	$ 36,521.03	$ 34,216.82	18.68%	46.08%	27.41%	55	48	103	-2.85%	53.40%
2000	$ 22,039.50	$ 33,549.30	$ 31,839.44	-11.84%	34.20%	46.04%	54	50	104	-3.53%	51.92%
2001	$ 21,602.70	$ 29,570.68	$ 28,656.55	-13.59%	18.28%	31.87%	51	57	108	-6.76%	47.22%
2002	$ 19,145.91	$ 30,834.24	$ 29,667.39	-23.42%	23.34%	46.75%	62	57	119	-5.04%	52.10%
2003	$ 31,269.68	$ 33,096.47	$ 31,477.17	25.08%	32.39%	7.31%	53	42	95	-2.67%	55.79%
2004	$ 26,954.01	$ 30,057.79	$ 29,046.23	7.82%	20.23%	12.42%	61	49	110	-1.19%	55.45%
2005	$ 25,524.55	$ 30,002.17	$ 29,001.73	2.10%	20.01%	17.91%	54	50	104	-1.84%	51.92%
2006	$ 28,182.38	$ 31,603.53	$ 30,282.82	12.73%	26.41%	13.68%	52	34	86	-3.01%	60.47%
2007	$ 25,586.75	$ 28,073.50	$ 27,458.80	2.35%	12.29%	9.95%	53	61	114	-3.41%	46.49%
2008	$ 15,340.80	$ 25,457.67	$ 25,366.13	-38.64%	1.83%	40.47%	52	63	115	-4.48%	45.22%
2009	$ 30,757.44	$ 39,559.73	$ 36,647.78	23.03%	58.24%	35.21%	50	56	106	-5.02%	47.17%
2010	$ 27,916.50	$ 30,502.17	$ 29,401.73	11.67%	22.01%	10.34%	45	51	96	-3.27%	46.88%
2011	$ 24,723.50	$ 24,176.64	$ 24,341.31	-1.11%	-3.29%	-2.19%	48	64	112	-7.79%	42.86%
2012	$ 27,769.95	$ 29,572.61	$ 28,658.09	11.08%	18.29%	7.21%	48	59	107	-1.35%	44.86%
2013	$ 31,766.68	$ 29,402.36	$ 28,521.89	27.07%	17.61%	-9.46%	43	76	119	-2.11%	36.13%
2014	$ 27,747.90	$ 30,402.88	$ 29,322.31	10.99%	21.61%	10.62%	56	48	104	-2.30%	53.85%
2015	$ 24,668.27	$ 31,163.54	$ 29,930.83	-1.33%	24.65%	25.98%	60	48	108	-3.17%	55.56%
2016	$ 27,717.72	$ 27,415.24	$ 26,932.20	10.87%	9.66%	-1.21%	44	62	106	-3.65%	41.51%
2017	$ 29,621.46	$ 27,580.33	$ 27,064.27	18.49%	10.32%	-8.16%	37	63	100	-1.05%	37.00%
2018	$ 23,040.75	$ 28,019.26	$ 27,415.41	-7.84%	12.08%	19.91%	51	45	96	-4.27%	53.13%
Total	$ 48,334.00	$ 144,969.19	$ 115,975.35	193.34%	579.88%	386.54%	1283	1351	2634	-	-
Average	$ 26,933.36	$ 30,798.77	$ 29,639.01	7.73%	23.20%	15.46%	51.32	54.04	105.36	-3.35%	48.94%
Std Dev	$ 4,411.27	$ 3,435.58	$ 2,748.46	17.65%	13.74%	16.18%	5.55	9.16	8.35	1.69%	6.00%

Year	Benchmark	Algorithm Returns	Returns After Tax	Benchmark Returns	AlgoReturns	Algo vs Benchmark	Winning Trades	Losing Trades	Total Trades	Max DrawDown	Accuracy
1994	$ 24,422	$ 29,691	$ 28,752.88	-2.31%	18.76%	21.08%	50	42	92	-2.55%	54.35%
1995	$ 32,956	$ 37,214	$ 34,771	34.94%	25.34%	-9.61%	100	98	198	-2.55%	50.51%
1996	$ 39,580	$ 49,714	$ 44,771	20.10%	33.59%	13.49%	151	149	300	-3.04%	50.33%
1997	$ 52,026	$ 73,365	$ 63,692	31.44%	47.57%	16.13%	205	199	404	-3.55%	50.74%
1998	$ 66,096	$ 95,847	$ 81,678	27.04%	30.64%	3.60%	257	265	522	-5.03%	49.23%
1999	$ 78,725	$ 140,085	$ 117,068	19.11%	46.15%	27.05%	312	313	625	-5.03%	49.92%
2000	$ 70,317	$ 186,166	$ 153,933	-10.68%	32.89%	43.58%	367	363	730	-5.03%	50.27%
2001	$ 61,265	$ 220,267	$ 181,214	-12.87%	18.32%	31.19%	419	419	838	-6.76%	50.00%
2002	$ 47,291	$ 275,841	$ 225,673	-22.81%	25.23%	48.04%	483	474	957	-6.76%	50.47%
2003	$ 59,646	$ 378,925	$ 308,140	26.12%	37.37%	11.25%	536	515	1051	-6.76%	51.00%
2004	$ 64,786	$ 455,463	$ 369,371	8.62%	20.20%	11.58%	597	565	1162	-6.76%	51.38%
2005	$ 66,737	$ 546,910	$ 442,528	3.01%	20.08%	17.07%	652	614	1266	-6.76%	51.50%
2006	$ 75,908	$ 700,103	$ 565,082	13.74%	28.01%	14.27%	704	648	1352	-6.76%	52.07%
2007	$ 78,369	$ 786,689	$ 634,351	3.24%	12.37%	9.13%	758	708	1466	-6.76%	51.71%
2008	$ 48,369	$ 800,811	$ 645,649	-38.28%	1.80%	40.08%	811	770	1581	-6.76%	51.30%
2009	$ 59,732	$ 1,324,473	$ 1,064,578	23.49%	65.39%	41.90%	862	824	1686	-6.76%	51.13%
2010	$ 67,402	$ 1,630,391	$ 1,309,313	12.84%	23.10%	10.26%	908	874	1782	-6.76%	50.95%
2011	$ 67,268	$ 1,576,460	$ 1,266,168	-0.20%	-3.31%	-3.11%	956	938	1894	-7.80%	50.48%
2012	$ 76,332	$ 1,865,715	$ 1,497,572	13.47%	18.35%	4.87%	1004	997	2001	-7.80%	50.17%
2013	$ 98,994	$ 2,261,516	$ 1,814,213	29.69%	21.21%	-8.47%	1048	1071	2119	-7.80%	49.46%
2014	$ 110,169	$ 2,742,438	$ 2,198,950	11.29%	21.27%	9.98%	1105	1119	2224	-7.80%	49.69%
2015	$ 109,274	$ 3,343,912	$ 2,680,129	-0.81%	21.93%	22.74%	1167	1167	2334	-7.80%	50.00%
2016	$ 119,812	$ 3,669,460	$ 2,940,568	9.64%	9.74%	0.09%	1212	1228	2440	-7.80%	49.67%
2017	$ 143,037	$ 4,077,434	$ 3,266,947	19.38%	11.12%	-8.27%	1250	1290	2540	-7.80%	49.21%
2018	$ 132,794	$ 4,584,300	$ 3,672,440	-7.16%	12.43%	19.59%	1301	1335	2636	-7.80%	49.36%
		TOTAL		212.06%	599.55%						
		AVERAGE		16.31%	46.12%						
		STD DEV		17.77%	14.71%						

```
<generator object DataFrame.iterrows at 0x7f5152bf3a40>
THATS THE REINDEXED DATA
...updateBars called
...getNewBar called
('2014-04-07', logID                    3775
open                  168.368
high                  168.694
low                   166.611
close                 166.956
volume              1.40803e+08
unadjustedClose            None
unadjustedVolume    1.40803e+08
changed                -1.8657
changePercent           -1.105
vwap                  167.536
label                 Apr 7, 14
changeOverTime                0
Name: 2014-04-07, dtype: object)
0
```

```
'''
    Okay so this is where it gets tricky. The method update_timeindex
    handles the new holdings tracking. First it optains the latest prices
    from the market data handler and creates a new dictionary of symbols to
    represent the current_positions. these are changed when a FillEvent is
    given. WHich is handled in a later function. Th emethod appends this set of
    current_positions to the all_positions list. Next the holdings are updated
    in a simalar manner.  with the exemption that the market value is
    recalculated by multiplying the current_positions count with the closing
    price of the last bar

    ie: self.current_positions[s] * bars[s][0][5]

    then the new holdings are appended to all_holdings
'''
def update_timeindex(self, event):
    ###
    # Adds a new record to the positions matrix/table for the current market
    # data bar. This relects the previous bar.
    # this means all current market data is "Known"
    #
    # This is the function that acually  uses the MarketEvent from the queue
    ###
    print('inside Portfolio.update_timeindex')
    bars = {}
    for sym in self.symbolList:
        bars[sym] = self.bars.getLatestBars(sym, N=1)

    ###
    # Update positions
    ###
    index = 0
    dp = dict( (x,y) for x, y in [(s, 0) for s in self.symbolList] )
    dp['date'] = bars[0][0][1]

    for x in self.symbolList:
        dp[x] = self.current_positions[x]

    #Append current positions
    self.all_positions.append(dp)

    #update holdings
    dh = dict( (x,y) for x, y in [(s, 0) for s in self.symbolList] )
    dh['date'] = bars[0][0][1]
    dh['cash'] = float(self.current_holdings['cash'])
    dh['total'] = float(self.current_holdings['cash'])

    for s in self.symbolList:
        #the real value of portfolio.
        market_value = float(float(self.current_positions[s]) * float(bars[s][0][5]))
        print(market_value)

        dh[s] = market_value
        dh['total'] += market_value

    self.all_holdings.append(dh)
```

```
from DataHandler import HistoricSQLDataHandler

from ExecutionDriver import SimulatedExecutionHandler
import PortfolioDriver
import Strategy
import sys
is_py2 = sys.version[2] == '2'
if is_py2:
    import Queue as queue
else:
    from multiprocessing import Queue
import time
import queue

events = queue.Queue()
symbolList = {}
symbolList[0] = 'SPY_Data'
bars = HistoricSQLDataHandler(events, 'SELECT * FROM', symbolList)
#strategy = Strategy.SterlingBeatsBuffet(bars, events) #Trend Strat
strategy = Strategy.BuyAndHoldStrategy(bars, events) #Benchmark
port = PortfolioDriver.BasicPortfolio(bars, events, '2014-04-07')
broker = SimulatedExecutionHandler(events)

while True:
    if bars.continueBacktest == True:
        bars.updateBars()
    else:
        break

    while True:
        try:
            event = events.get(False)
        except queue.Empty:
            print('The Queue Is Currently Empty')

            break
        else:
            if event is not None:
                print(event)
                if event.type == 'MARKET':
                    print('MARKET EVENT RECIEVED')
                    print(event)
                    strategy.calculate_signals(event)
                    port.update_timeindex(event)
                elif event.type == 'SIGNAL':
                    print('SIGNAL EVENT RECIEVED')
                    #input('Press Enter To Continue')
                    port.updateSignal(event)
                elif event.type == 'ORDER':
                    print('ORDER EVENT RECIEVED')
                    #input('Press Enter To Continue')
                    broker.execute_order(event)
                elif event.type == 'FILL':
                    print('FILL EVENT RECIEVED')
                    #input('Press Enter To Continue')
                    port.updateFill(event)
print('Analysis Complete')
```


Controller
IEX Data
Update Data Store
- Find most recent data update
- Fetch The missing data from IEX
- Send that to dataStore
DataManager
- Recieves requests for data
- Saves data to datastore
15min SPY data
(mySQL)Portfolio
- Order tracking
- Performance
Check If Backtesting
- Controller tells if we are live or testing
Return Data
Get Data
Logic Block
Algos
Send Order
or Recieve Portfolio
Status
PortfolioManager
- Recieves requests for positions
- Saves orders to database
-Checks to see if trade is possible
BackTestControl
1.Loops algos to find effecient for given TIME
2. Runs Effecient Algos
1
2
Initiate Portfolio