Sarit Maitra

1.4K Followers About

TEST HARNESS IS IMPORTANT TO EVALUATE TRADING STRATEGY

Simple Way of Evaluating Algorithmic Trading Strategy

Designing a financial trading strategy

Sarit Maitra 6 days ago · 7 min read ★

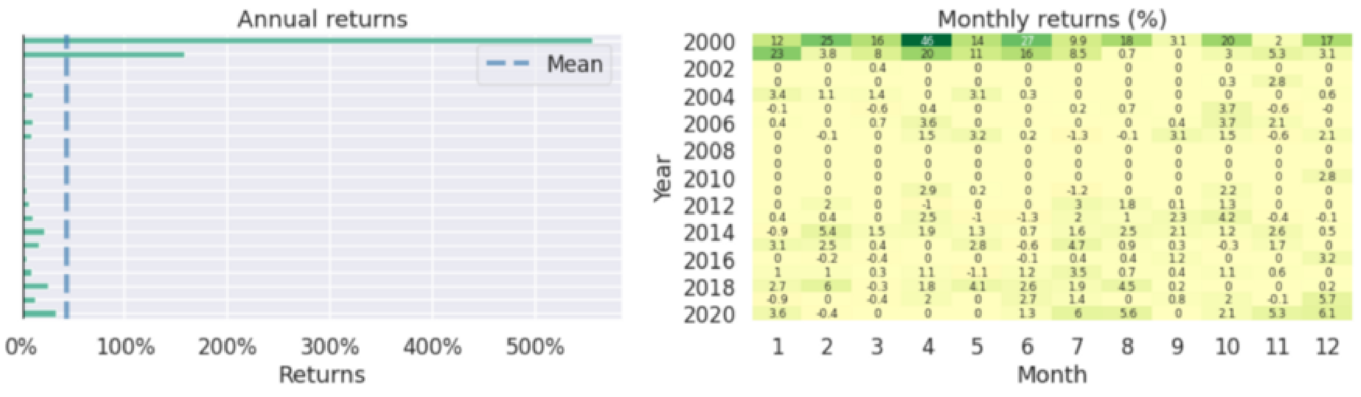
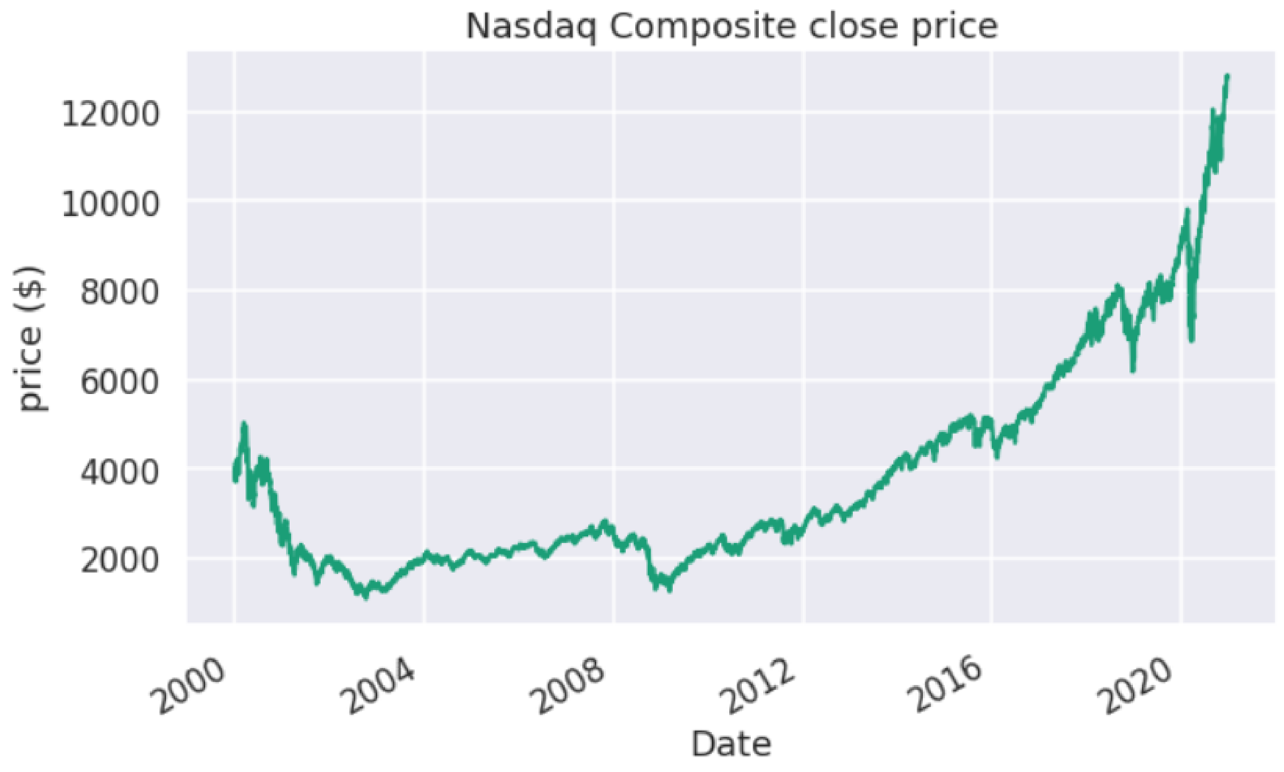


Image by author

Backtesting is an important step to get the statistics to ensure effective trading strategy. It comes with some of the key points such as profit and loss, net profit and loss, invested capital, number of trades/orders return, sharpe ratio etc. Here, we will discuss as how to design a financial trading strategy using open source Python tools and we'll review the results of the backtest by going through some plots generated by pyfolio.

```
dataset = web.DataReader('^IXIC', data_source = 'yahoo', start =
'2000-01-01')
dataset = dataset.sort_index(ascending=True)
# display
print(dataset.head()); print(dataset.shape)
# Plot the adjusted closing prices
dataset['Adj Close'].plot(grid=True, figsize=(10, 6))
plt.title('Nasdaq Composite close price')
plt.ylabel('price ($)')
# Show the plot
plt.show()
```

```
                   High          Low         Open        Close      Volume    Adj Close
Date
2000-01-03  4192.189941  3989.709961  4186.189941  4131.149902  1510070000  4131.149902
2000-01-04  4073.250000  3898.229980  4020.000000  3901.689941  1511840000  3901.689941
2000-01-05  3924.209961  3734.870117  3854.350098  3877.540039  1735670000  3877.540039
2000-01-06  3868.760010  3715.620117  3834.439941  3727.129883  1598320000  3727.129883
2000-01-07  3882.669922  3711.090088  3711.090088  3882.620117  1634930000  3882.620117
(5280, 6)
```

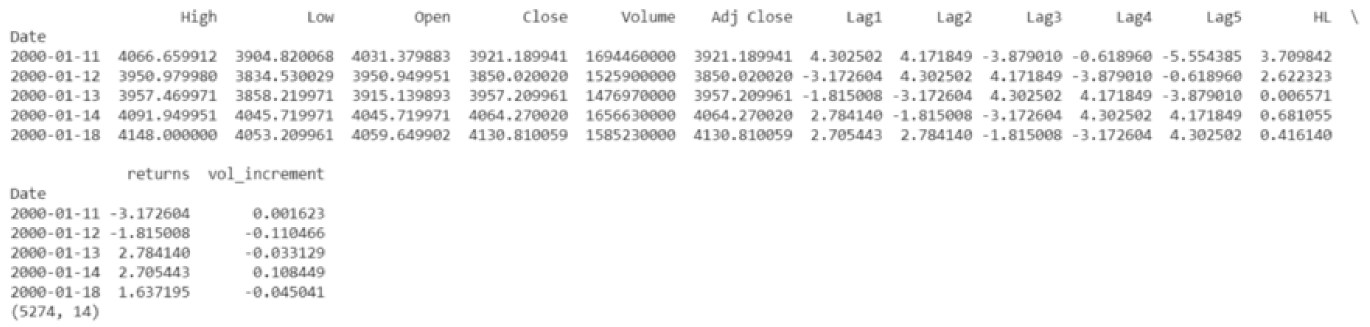


It can be noticed that data is on daily frequency.

> (2009) suggest that it is really difficult to make money as a day trader. Instead daily charts gives a much better chance. Shorter timeframes are unreliable, because the market hasn't had enough time to digest the information. Chague et al. (2019) found that it is "virtually impossible" for day traders to make money in the long run.

Let's start preparing the data for the analysis. I have created some simple features as shown below.

```
lags = 5
# Shifted lag series of prior period Adj close values
for i in range(0, lags):
    dataset["Lag%s" % str(i+1)] = dataset["Adj
Close"].shift(i+1).pct_change()*100.0
dataset['HL'] = (dataset['High'] - dataset['Close'])/
dataset['Close'] * 100
# creating more features
dataset['returns'] = dataset['Adj Close'].pct_change() * 100
dataset['vol_increment'] = dataset.Volume.diff() / dataset.Volume
dataset.dropna(inplace=True)
print(dataset.head()); print(dataset.shape)
```



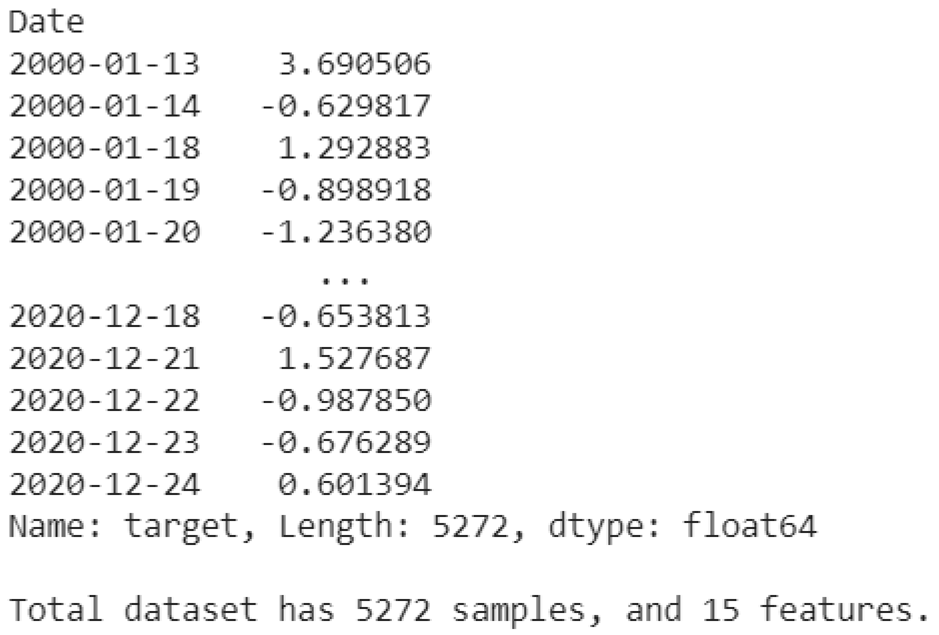
```
                   High          Low         Open        Close      Volume    Adj Close     Lag1      Lag2      Lag3      Lag4      Lag5        HL  \
Date
2000-01-11  4066.659912  3904.820068  4031.379883  3921.189941  1694460000  3921.189941  4.302502  4.171849 -3.879010 -0.618960 -5.554385  3.709842
2000-01-12  3950.979980  3834.530029  3950.949951  3850.020020  1525900000  3850.020020 -3.172604  4.302502  4.171849 -3.879010 -0.618960  2.622323
2000-01-13  3957.469971  3858.219971  3915.139893  3957.209961  1476970000  3957.209961 -1.815008 -3.172604  4.302502  4.171849 -3.879010  0.006571
2000-01-14  4091.949951  4045.719971  4045.719971  4064.270020  1656630000  4064.270020  2.784140 -1.815008 -3.172604  4.302502  4.171849  0.681055
2000-01-18  4148.000000  4053.209961  4059.649902  4130.810059  1585230000  4130.810059  2.705443  2.784140 -1.815008 -3.172604  4.302502  0.416140

             returns  vol_increment
Date
2000-01-11 -3.172604       0.001623
2000-01-12 -1.815008      -0.110466
2000-01-13  2.784140      -0.033129
2000-01-14  2.705443       0.108449
2000-01-18  1.637195      -0.045041
(5274, 14)
```

Target Variable

alternate for -ve values, we buy (-1). This will be a classification variable, if the average price will go either up or down the next day.

```
dataset['target'] = dataset['Close'].pct_change()*100.0 -
dataset['Open'].pct_change()*100.0
dataset.dropna(inplace=True)
print(dataset.target)
print('Total dataset has {} samples, and {}
features.'.format(dataset.shape[0], dataset.shape[1]))
```



```
Date
2000-01-13    3.690506
2000-01-14   -0.629817
2000-01-18    1.292883
2000-01-19   -0.898918
2000-01-20   -1.236380
                ...
2020-12-18   -0.653813
2020-12-21    1.527687
2020-12-22   -0.987850
2020-12-23   -0.676289
2020-12-24    0.601394
Name: target, Length: 5272, dtype: float64

Total dataset has 5272 samples, and 15 features.
```

We see here both +ve and -ve values in target; we could turn this into classification task (+1 & -1 later).

Train/Test split:

I have introduced gaps between the training set and the test set; objective is to mitigate the temporal dependence. Moreover, data is split into Kfolds. The test sets are untouched, while the training sets get the gaps removed.

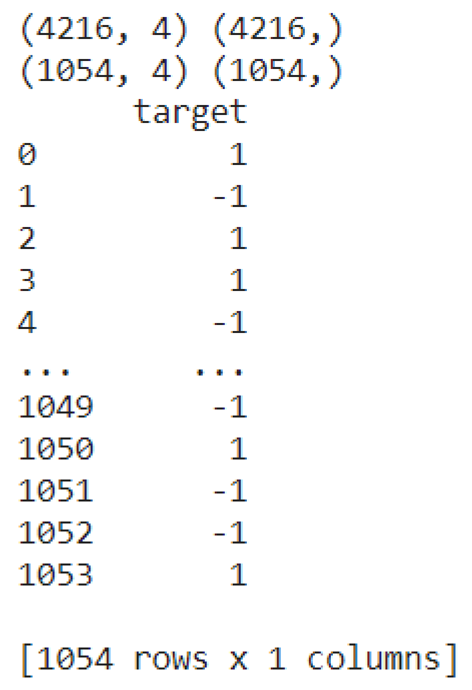
Overfitting is a common phenomenon on machine learning and thereby, it is important to use cross-validation technique to improve the accuracy of model.

```
for train_samples, test_samples in gkcv.split(X,y):
    XTrain, XTest = X.values[train_samples], X.values[test_samples
    yTrain, yTest = y.values[train_samples], y.values[test_samples]

feature_names = ['Lag1','Lag2','HL', 'returns']
XTrain = pd.DataFrame(data=XTrain, columns=feature_names)
XTest = pd.DataFrame(data=XTest, columns=feature_names)
print(XTrain.shape, yTrain.shape); print(XTest.shape, yTest.shape)

yTrain = pd.DataFrame(yTrain);
yTrain.rename(columns = {0: 'target'}, inplace=True)
yTest = pd.DataFrame(yTest)
yTest.rename(columns = {0: 'target'}, inplace=True)

def getBinary(val):
    if val>0:
        return 1
    else:
        return -1
yTestBinary = pd.DataFrame(yTest["target"].apply(getBinary))
print(yTestBinary)
```

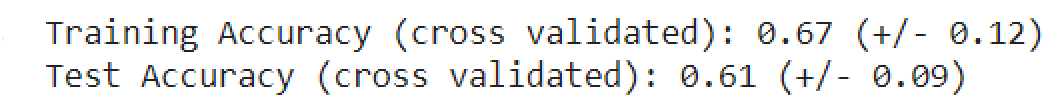


```
(4216, 4) (4216,)
(1054, 4) (1054,)
      target
0          1
1         -1
2          1
3          1
4         -1
...      ...
1049      -1
1050       1
1051      -1
1052      -1
1053       1

[1054 rows x 1 columns]
```

The target variable here transformed for classification. A positive change (>0) in the value of prices classified as 1 and a non-positive change as -1.

```
regressor = xgb.XGBRegressor(objective ='reg:squarederror',
gamma=0.0, n_estimators=200, base_score=0.7, colsample_bytree=1,
learning_rate=0.01).fit(XTrain, yTrain)
scores = cross_val_score(regressor, XTrain.values, yTrain.values,
cv=gkcv)
print("Training Accuracy (cross validated): %0.2f (+/- %0.2f)" %
(scores.mean(), scores.std() * 2))
scores = cross_val_score(regressor, XTest.values, yTest.values,
cv=gkcv)
print("Test Accuracy (cross validated): %0.2f (+/- %0.2f)" %
(scores.mean(), scores.std() * 2))
```



```
Training Accuracy (cross validated): 0.67 (+/- 0.12)
Test Accuracy (cross validated): 0.61 (+/- 0.09)
```

ML is not the focus here, so, I will avoid getting into the details of it. Our objective is to showcase how to run test harness to validate the performance of a trading strategy. The idea here is to understand that given historical data, what kind of performance would a specific trading strategy have. Having said so, the first step of algorithmic trading is to develop the ML model with necessary features.

We have to remember that, once the model is goes into production, the next step is to test and ensure that the algorithm behaves as expected in the trading infrastructure.

```
model =  xgb.XGBRegressor(objective ='reg:squarederror',
gamma=0.0, n_estimators=200, base_score=0.7, colsample_bytree=1,
learning_rate=0.01).fit(X, y)
Predict = model.predict(X)
PredictBinary = [1 if yp > 0 else -1 for yp in Predict]
BinaryPredict = pd.DataFrame(PredictBinary)
BinaryPredict.rename({0 : 'Prediction'}, axis=1, inplace=True)
BinaryPredict.index = dataset.index
print(BinaryPredict) ; print()

# number of trades over time for highest and second highest return
strategy
print('Number of trades = ',
(BinaryPredict.Prediction.diff()!=0).sum())
```

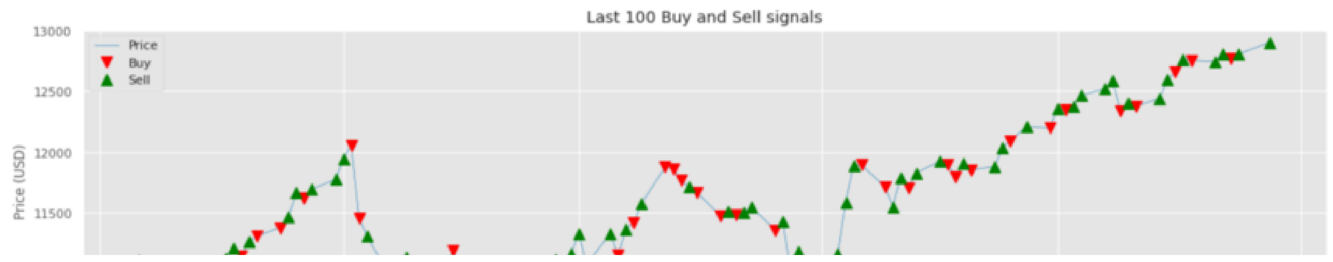
```
2000-01-13          1
2000-01-14         -1
2000-01-18          1
2000-01-19         -1
2000-01-20         -1
...               ...
2020-12-18         -1
2020-12-21          1
2020-12-22          1
2020-12-23         -1
2020-12-24          1

[5272 rows x 1 columns]

Number of trades =  3471
```

Visualization buy/sell signals:

```
# Buy/Sell signals plot
sells = prices.loc[prices['predicted'] == 1];
buys = prices.loc[prices['predicted'] == -1];
# Plot
fig = plt.figure(figsize=(20, 5));
plt.plot(prices.index[-100:], prices['close'][-100:], lw=2.,
label='Price');
# Plot buy and sell signals
# up arrow when we buy one share
plt.plot(buys.index[-100:], prices.loc[buys.index]['close'][-100:],
'v', markersize=10, color='red', lw=2., label='Buy');
# down arrow when we sell one share
plt.plot(sells.index[-100:], prices.loc[sells.index]['close'][-100:],
'^', markersize = 10, color='green', lw=2., label='Sell');
plt.ylabel('Price (USD)'); plt.xlabel('Date');
plt.title('Last 100 Buy and Sell signals'); plt.legend(loc='best');
plt.show()
```



Strategy evaluation:

Here, I have calculated the signals on day t's close price, but calculated the number of shares to buy based on day t+1's open price. Backtesting has been conducted to assess the viability of this strategy by discovering how it would play out using historical data. This is the logical combination of the strategy with the data set. The data reflects a variety of market conditions since 2000 and thereby can better judge the results of the backtest.

In Backtrader, strategy class needs to inherit from bt.Strategy. It has input parameters as assets and rebalance_months. In the notify_order function, we need to check if the order is completed. If the order is completed then notify the trade with the notify_trade function which completes the Strategy class.

Finally, need to add the Strategy to cerebro with input as the list of tuples of stock and weight that we created in the previous section. We add returns and pyfolio analyzers that we will use to extract the performance of our strategy and run the strategy and capture the output in the results variable.

```
OHLCV = ['open', 'high', 'low', 'close', 'volume']
class SignalData(PandasData):
    cols = OHLCV + ['predicted']
    lines = tuple(cols)
    # parameters
    params = {c: -1 for c in cols}
    params.update({'Date': None})
    params = tuple(params.items())

class MLStrategy(bt.Strategy):
    params = dict()
    def __init__(self):
        self.data_predicted = self.datas[0].predicted
        self.data_open = self.datas[0].open
        self.data_close = self.datas[0].close
        self.order = None
        self.price = None
        self.comm = None
```

Open in app

```
    def notify_order(self, order):
        if order.status in [order.Submitted, order.Accepted]:
            return
        if order.status in [order.Completed]:
            if order.isbuy():
                self.log(f'BUY EXECUTED — — Price:
{order.executed.price:.2f}, Cost:
{order.executed.value:.2f},Commission: {order.executed.comm:.2f}')
                self.price = order.executed.price
                self.comm = order.executed.comm
            else:
                self.log(f'SELL EXECUTED — — Price:
{order.executed.price:.2f}, Cost:
{order.executed.value:.2f},Commission: {order.executed.comm:.2f}')
            elif order.status in [order.Canceled, order.Margin,
order.Rejected]:
                self.log('Order Failed')
                self.order = None
    def notify_trade(self, trade):
        if not trade.isclosed:
            return
            self.log(f'OPERATION RESULT — — Gross: {trade.pnl:.2f},
Net: {trade.pnlcomm:.2f}')
    def next_open(self):
        if not self.position:
            if self.data_predicted > 0:
                size = int(self.broker.getcash() /
self.datas[0].open)
                self.log(f'BUY CREATED — — Size: {size}, Cash:
{self.broker.getcash():.2f}, Open: {self.data_open[0]}, Close:
{self.data_close[0]}')
                self.buy(size=size)
            else:
                if self.data_predicted < 0:
                    self.log(f'SELL CREATED — — Size:
{self.position.size}')
                    self.sell(size=self.position.size)

data = SignalData(dataname=prices)
cerebro = bt.Cerebro(stdstats = False, cheat_on_open=True)
cerebro.addstrategy(MLStrategy)
cerebro.adddata(data)
cerebro.broker.setcash(100000.0)
cerebro.broker.setcommission(commission=0.001)
cerebro.addanalyzer(bt.analyzers.PyFolio, _name='pyfolio')

# run the backtest
print('Starting Portfolio Value: %.2f' %
cerebro.broker.getvalue())backtest_result = cerebro.run()
print('Final Portfolio Value: %.2f' % cerebro.broker.getvalue())
```

```
2020-12-02, SELL CREATED --- Size: 749
2020-12-02, SELL EXECUTED --- Price: 12285.75, Cost: 9222706.90,Commission: 9202.03
2020-12-02, OPERATION RESULT --- Gross: -20680.15, Net: -39104.89
2020-12-03, BUY CREATED --- Size: 743, Cash: 9193448.59, Open: 12369.259765625, Close: 12377.1796875
2020-12-03, Order Failed
2020-12-04, BUY CREATED --- Size: 741, Cash: 9193448.59, Open: 12399.3203125, Close: 12464.23046875
2020-12-04, Order Failed
2020-12-07, BUY CREATED --- Size: 737, Cash: 9193448.59, Open: 12461.0, Close: 12519.9501953125
2020-12-07, BUY EXECUTED --- Price: 12461.00, Cost: 9183757.00,Commission: 9183.76
2020-12-09, SELL CREATED --- Size: 737
2020-12-09, SELL EXECUTED --- Price: 12591.69, Cost: 9183757.00,Commission: 9280.08
2020-12-09, OPERATION RESULT --- Gross: 96318.85, Net: 77855.01
2020-12-10, BUY CREATED --- Size: 756, Cash: 9271303.61, Open: 12247.5498046875, Close: 12405.8095703125
2020-12-10, BUY EXECUTED --- Price: 12247.55, Cost: 9259147.65,Commission: 9259.15
2020-12-11, SELL CREATED --- Size: 756
2020-12-11, SELL EXECUTED --- Price: 12336.79, Cost: 9259147.65,Commission: 9326.61
2020-12-11, OPERATION RESULT --- Gross: 67465.62, Net: 48879.86
2020-12-14, BUY CREATED --- Size: 748, Cash: 9320183.46, Open: 12447.4404296875, Close: 12440.0400390625
2020-12-14, BUY EXECUTED --- Price: 12447.44, Cost: 9310685.44,Commission: 9310.69
2020-12-16, SELL CREATED --- Size: 748
2020-12-16, SELL EXECUTED --- Price: 12611.04, Cost: 9310685.44,Commission: 9433.06
2020-12-16, OPERATION RESULT --- Gross: 122372.51, Net: 103628.76
2020-12-17, BUY CREATED --- Size: 740, Cash: 9423812.23, Open: 12730.7802734375, Close: 12764.75
2020-12-17, Order Failed
2020-12-21, BUY CREATED --- Size: 748, Cash: 9423812.23, Open: 12596.1396484375, Close: 12742.51953125
2020-12-21, Order Failed
2020-12-22, BUY CREATED --- Size: 737, Cash: 9423812.23, Open: 12785.2197265625, Close: 12807.919921875
2020-12-22, Order Failed
2020-12-24, BUY CREATED --- Size: 736, Cash: 9423812.23, Open: 12791.5400390625, Close: 12804.73046875
2020-12-24, Order Failed
Final Portfolio Value: 9423812.23
```

We see here that out portfolio value has turned into 9.4m over a period of 20 years. Well this is an example of how we can turn a regression model into classification use case and run a simple back-test to evaluate the trading strategy taken.

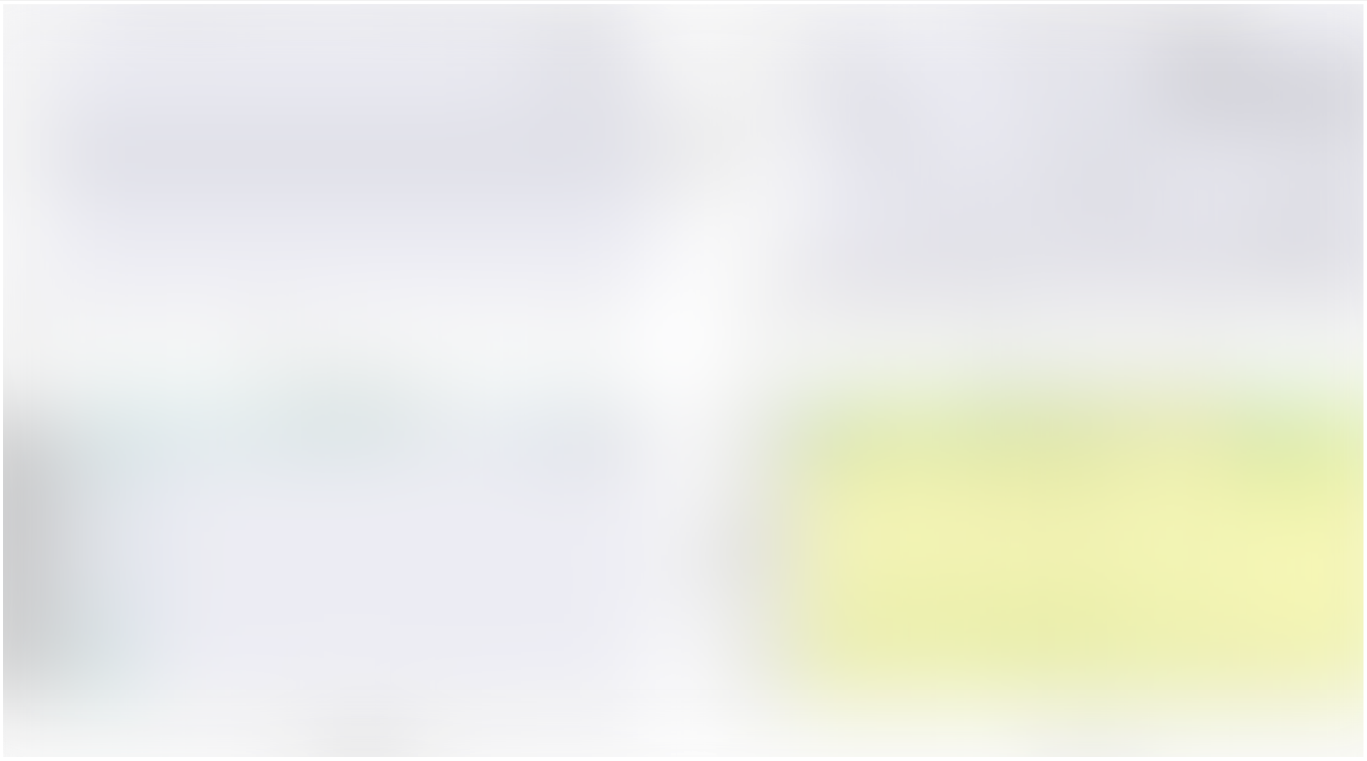
Summarize report:

```python
# Extract inputs for pyfolio
strat = backtest_result[0]
pyfoliozer = strat.analyzers.getbyname('pyfolio')
returns, positions, transactions, gross_lev = pyfoliozer.get_pf_items()
returns.name = 'Strategy'
#returns.tail()

# benchmark returns
benchmark_rets = dataset['returns']
benchmark_rets.index = benchmark_rets.index.tz_localize('UTC')
benchmark_rets = benchmark_rets.filter(returns.index)
benchmark_rets.name = 'Nasdaq Composite'
#benchmark_rets.tail()
```

Open in app

Open in app



Conclusion:

I have used backtrader of pyfolio (python open source library). We can also test the performance statistics against benchmark statistics. It might be a coincidence that overall the strategy performance looks good; however, 20 years is a long time and also, I didn't use a scientific approach to define my view and confidence in it. Moreover, I have used free pricing data and there might be data quality issues.

I can be reached here.

Disclaimer: The programs described here are experimental and should be used with caution for any commercial purpose. All such use at your own risk.

Trading Strategy Backtesting Algorithmic Trading Machine Learning