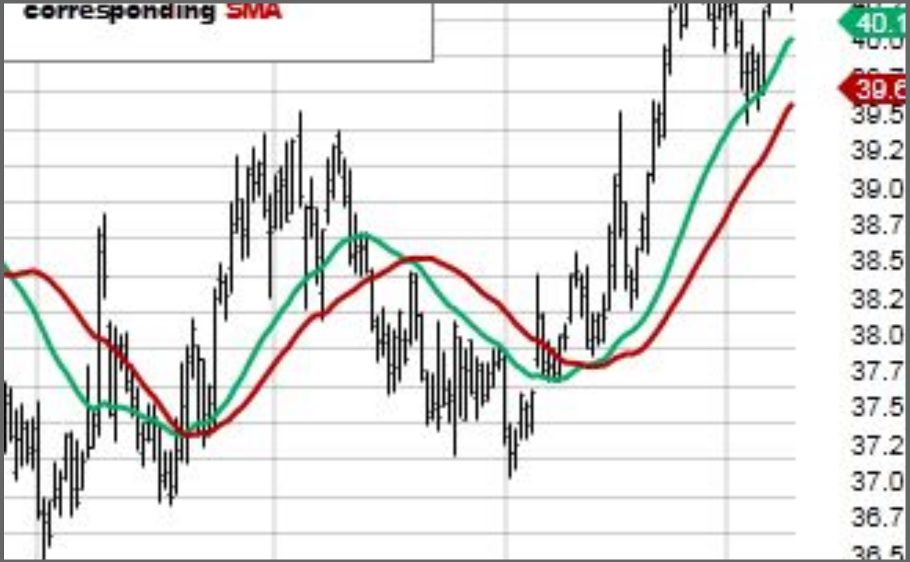
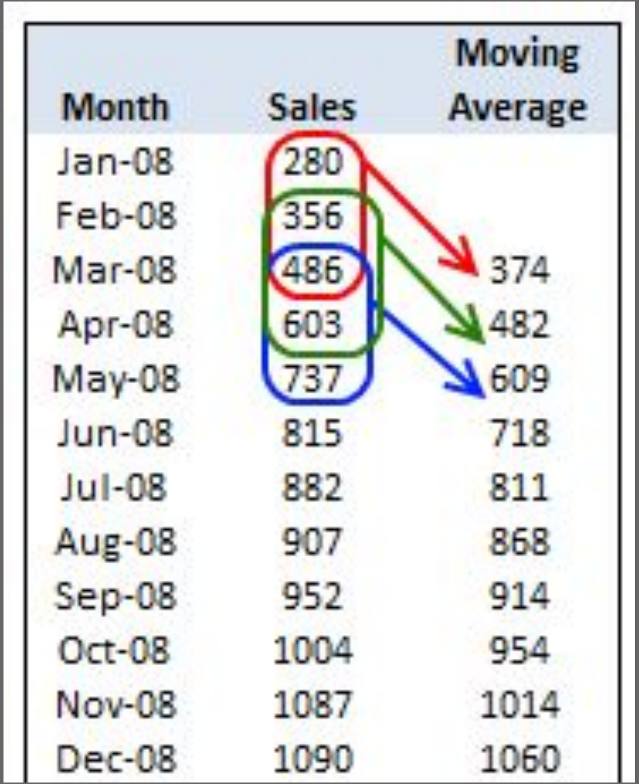
Simple MAC strategy

Delish

What is MAC (Moving Average Crossover)?

Moving Average: (right pic is 3 days)
Use different number of days to create different MA lines, and bull/sell on crossovers





Month	Sales	Moving Average
Jan-08	280	
Feb-08	356	
Mar-08	486	374
Apr-08	603	482
May-08	737	609
Jun-08	815	718
Jul-08	882	811
Aug-08	907	868
Sep-08	952	914
Oct-08	1004	954
Nov-08	1087	1014
Dec-08	1090	1060

Libraries/software needed

- Python 3
- Python IDE (unless Chad notepad user?)
- PyPi packages
  - Yahoo_fin
  - Trade-stat-logger
  - Recommended learning for future: numpy, pandas, matplotlib

Let’s get to coding!

Switch to IDE: (here I will live code everything you will see in future slides

Explanations

Why use a bunch of function params instead of hardcoding?

Allows us to dynamically assign MA values

Why

```
if ndays_momentum > ndays_resistance:
   raise ValueError('momentum should be short term, thus fewer days')
```

Momentum should be computed with fewer days than the resistance because we want momentum to be more responsive to recent changes

```
df = get_data(ticker=ticker, start_date=start_date, end_date=end_date)
df = df['open']
daily_df = df[ndays_resistance:]
# removes NaNs and aligns the two moving averages and the daily
moving_average_momentum = df.rolling(window=ndays_momentum).mean()
moving_average_momentum = moving_average_momentum[ndays_resistance:]
moving_average_resistance = df.rolling(window=ndays_resistance).mean()
moving_average_resistance = moving_average_resistance[ndays_resistance
```

- get_data() is from the yahoo_fin package, returns a DF in OHLC with index as the date.
- We only want "open" to avoid look ahead-bias
  - say our algo trades in the middle of the day, if we use data like "close", then we rely on data not yet available

```
df = get_data(ticker=ticker, start_date=start_date, end_date=end_date)
df = df['open']
daily_df = df[ndays_resistance:]
# removes NaNs and aligns the two moving averages and the daily
moving_average_momentum = df.rolling(window=ndays_momentum).mean()
moving_average_momentum = moving_average_momentum[ndays_resistance:]
moving_average_resistance = df.rolling(window=ndays_resistance).mean()
moving_average_resistance = moving_average_resistance[ndays_resistance
```

- We create MAs by using using rolling().mean(), illustrated by (window=2) [1, 2, 3, 4] -> [NaN, 1.5, 2.5, 3.5].
- Since ndays_resistance is the largest value, we need to have all MAs start at ndays_resistance so day by day comparisons don’t include any NaN (Not a Number) values

```
prev_below_resistance = False
```

We want momentum to be below resistance first, then buy when momentum surpasses resistance

```
for x in range(len(moving_average_momentum)):
```

Allows us to iterate over 3 DFs. (len(any df) works b/c of slicing from b4)

```
    momentum = moving_average_momentum.iloc[x]
    resistance = moving_average_resistance.iloc[x]
    shares, _ = logger.get_position(ticker)
```

- iloc[] is for integer indexing for a Series (each DF column is a Series)
- We will use shares later to avoid over exposure (say you have $100,000, would you want to put that all on AMD for your portfolio?)

Buy Signal

```
if (momentum * bandwidth > resistance) and
              prev_below_resistance and shares < 100:
   logger.log(security=ticker, shares=100, share_price=daily_df.iloc[x])
   prev_below_resistance = False
```

- bandwidth = 1.05, used to make sure rapid buying and selling doesn’t occur near crossovers.
- Shares < 100 to make sure we don’t overexpose our position
- We want previous momentum to be below resistance (prev_below_resistance) because we want a scenario where is below resistance, but then exceeds resistance
- log() logs trades with given info

Sell Signals

```
elif momentum * threshold_ratio > resistance and shares > 0:
    logger.log(security=ticker, shares=-100, share_price=daily_df.iloc[x])
    prev_below_resistance = False
```

- Take profit once momentum reaches our threshold value multiple

```
elif momentum < resistance:
    logger.log(security=ticker, shares=-100, share_price=daily_df.iloc[x])
    prev_below_resistance = True
```

- If momentum goes against us, close position (like stop loss)
- Since momentum is below resistance again, we set p_b_r to True

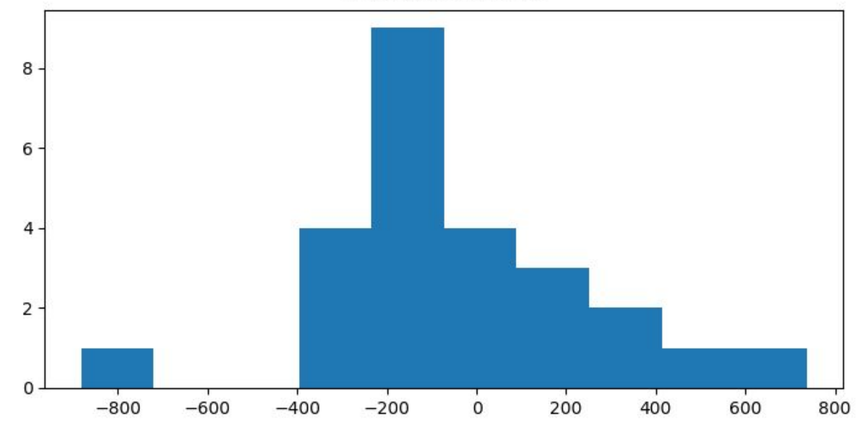
Putting it all together

```
logger_10_100 = compute_performance_MA(ticker='AAPL',
ndays_momentum=10, ndays_resistance=100, start_date='01/01/2017',
end_date='01/01/2019')
print(logger_10_100.get_summary_statistics())
```

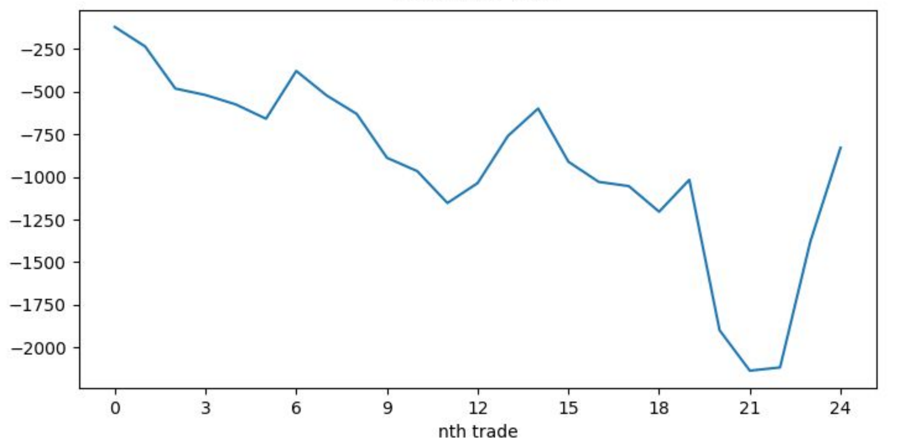
- Call function developed earlier
- Get statistics of our strategy!
- Results: {'profit': -828.9993286132812, 'drawdown': 2014.996337890625, 'std_dev': 306.65410507256234, 'fisher_kurtosis': 2.782634212064931, 'win_ratio': 0.32, 'average_win': 290.9996032714844, 'average_loss': -185.7056561638327, 'kelly_criterion': 0.7539519531014448}
- Hey, a losing strategy is winning strategy once shorted

Statistics

Return distribution

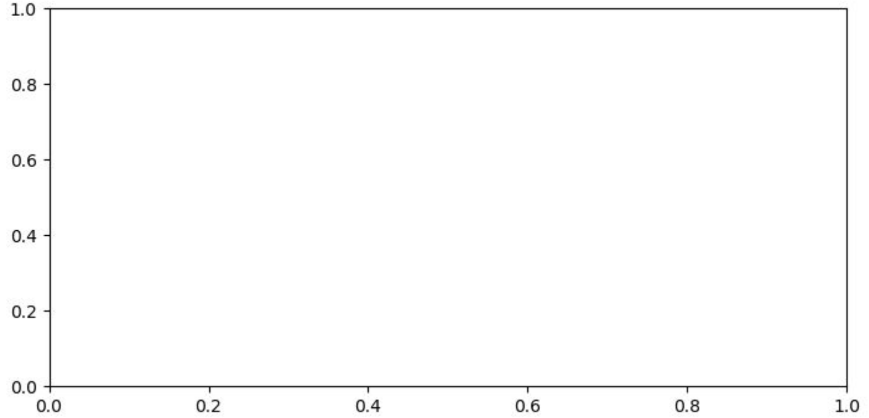


Profit over time



Statistics summary

profit	-829.0
drawdown	2015.0
std_dev	306.65
fisher_kurtosis	2.78
win_ratio	0.32
average_win	291.0
average_loss	-185.71
kelly_criterion	0.75



Poll

https://www.strawpoll.me/19817714