Algorithmic Bankruptcy

or: An Unofficial Companion Guide to the Georgia Institute of Technology's **CS 7646**: *Machine Learning for Trading*

George Kudrayvtsev
george.k@gatech.edu

Last Updated: July 22, 2019

THIS work is a culmination of hours of effort to create a lasting reference that follows along with Georgia Tech's graduate course on machine learning algorithms for trading. All of the explanations and algorithms are in my own words; the majority of the content (and many of the images) are based on Dr. Tucker Balch's lectures for the course.

This is still a work-in-progress; once this sentence is gone, you can consider it to be v1.0. There may be errors, typos, or entirely incorrect or misleading information, though I have done my best to ensure there isn't.

Much of this companion guide was fueled by caffeine. ☕ If you found it useful and are in a generous mood, feel free to buy me another cup of coffee! Shoot me a donation on Venmo `@george_k_btw` with whatever this guide was worth to you.

Happy studying! 😇

I Manipulating Financial Data 6

1 Python for Finance 7
1.1 Global Statistics . . . 7
1.1.1 Bollinger Bands® . . . 8
1.1.2 Daily Returns . . . 8
1.1.3 Cumulative Returns . . . 9
1.2 Fixing Bad Data . . . 10
1.3 Graphing Financial Data . . . 10
1.3.1 Histograms . . . 11
1.3.2 Scatter Plots . . . 13
1.4 Portfolios . . . 15
1.5 Optimizers . . . 17
1.5.1 Using an Optimizer . . . 18
1.5.2 Building Parameterized Models . . . 18
Error Metrics . . . 19
n Dimensions . . . 20
1.5.3 Portfolio Optimization . . . 20

II Computational Investing 22

2 Hedge Funds 23
2.1 Types of Managed Funds . . . 23
2.2 Compensation . . . 24
2.3 Attracting Investors . . . 26
2.3.1 Goals . . . 26
2.3.2 Metrics . . . 26
2.4 Hedge Fund Computing Architecture . . . 27

3 The Order Book 28
3.1 Making Orders . . . 28
3.1.1 The Order Book . . . 29
3.1.2 Filling Orders . . . 32
3.1.3 Exploiting the Order Book . . . 34
Exploit 1: Faster-Than-Light Travel . . . 34
Exploit 2: Geographic Arbitrage . . . 34
3.1.4 Selling Short . . . 35
What Can Go Wrong? . . . 36

4 Evaluating a Company 37
4.1 Metrics for Evaluation . . . 37
4.1.1 Intrinsic Value . . . 38
4.1.2 Book Value . . . 39
4.1.3 Market Capitalization . . . 40

4.1.4 Knowledge is Power . . . 40
4.2 Capital Asset Pricing Model . . . 41
4.2.1 Portfolio Management Under CAPM . . . 42
4.2.2 CAPM for Portfolios . . . 42
4.2.3 Arbitrage Pricing Theory . . . 43
4.2.4 CAPM for Hedge Funds . . . 43
4.3 Technical Analysis . . . 45
4.3.1 Indicators . . . 46
Momentum . . . 46
Simple Moving Average . . . 47
Bollinger Bands® . . . 48
Normalization . . . 49
4.4 Anomalous Price Changes . . . 49
4.4.1 Data Aggregation . . . 50
4.4.2 Stock Splits . . . 50
4.4.3 Dividends . . . 51

5 Beating the Market 52
5.1 The Efficient Markets Hypothesis . . . 52
5.1.1 Three Forms . . . 52
Weak Form . . . 52
Semi-Strong Form . . . 53
Strong Form . . . 53
5.1.2 EMH Validity . . . 53
5.2 The Importance of Diversification . . . 53
5.2.1 Coin Flipping Casino . . . 54

III Learning Algorithms for Trading 57

6 Supervised Regression Learning 58
6.1 Regression . . . 59
6.1.1 Linear Regression . . . 59
6.1.2 k-Nearest Neighbor . . . 60
6.1.3 Training vs. Testing . . . 61
6.1.4 Problems with Regression . . . 61
6.2 Decision Trees . . . 62
6.2.1 Representing a Decision Tree . . . 63
6.2.2 Learning a Decision Tree . . . 64
6.3 Evaluating a Learning Algorithm . . . 64
6.3.1 Metrics . . . 65
RMS Error . . . 65
Correlation . . . 65
6.3.2 Overfitting . . . 66
6.3.3 Cross-Validation . . . 67

6.4 Ensemble Learners . . . 67
6.4.1 Bootstrap Aggregating . . . 68
6.4.2 Boosting . . . 69

7 Reinforcement Learning **70**
7.1 Q-Learning . . . 72
7.1.1 Training a Q-Learner . . . 73
A Learning Step . . . 73
Exploration . . . 74
7.1.2 Trading as an MDP . . . 74
Creating the State . . . 75
7.1.3 Dyna-Q . . . 76
Hallucination . . . 76

Index of Terms **78**

Code Snippets

1.1 Normalizing prices . . . 7
1.2 Calculating daily returns . . . 8
1.3 Calculating cumulative returns . . . 10
1.4 Calculating portfolio statistics . . . 16
1.5 2D model fitting via vertical error. . . . 19
1.6 Polynomial function fitting via vertical error. . . . 20
4.1 Calculating momentum. . . . 47
4.2 Calculating the simple moving average. . . . 48
4.3 Calculating Bollinger Bands®. . . . 48
7.1 Discretizing learning factors . . . 75

PART I

Manipulating Financial Data

Python for Finance

Python has emerged as a dominant language in financial analysis in recent years due to its accessibility and the development powerful libraries such as **pandas** and **numpy** which enable highly-optimized manipulation of stock market data. Many of the statistics we'll talk about here have some accompanying snippets that describe how calculating that statistic would look in **pandas**. This is especially useful for various projects in *CS 7646* as there are many equally-correct ways to calculate certain values, but a particular calculation is expected.

Snippet 1.1: A convenient way to compare price data for stocks is to normalize them to all start at 1.0. This is trivial in **pandas**: just divide a dataframe by its first price.

```
def normalize(df):
  return df / df[0]
```

1.1 Global Statistics

Given a selection of stock market data, there are a number of global statistics we can calculate that can give us insights to trends in a particular stock over time and allow us to compare performance of various stocks in a portfolio.

These include things like the **rolling mean**, the **rolling standard deviation**, and more.

Rolling Statistics in pandas

There are a number of functions in the **pandas** library that give us global statistics. These aren't members of the **DataFrame** class; rather, we pass in a **DataFrame** and some additional parameters that specify a few details about the computation we want.

- **stats.moments.rolling_mean** — notable parameters include the window size and the number of minimum necessary observations

1.1.1 Bollinger Bands®

Invented by John Bollinger in the 1980s, **Bollinger bands** are a measurement of a stock's volatility. By leveraging the rolling standard deviation—specifically, 2σ above and below the global mean—we can get a sense of the volatility based on when the current stock intersects the bands.

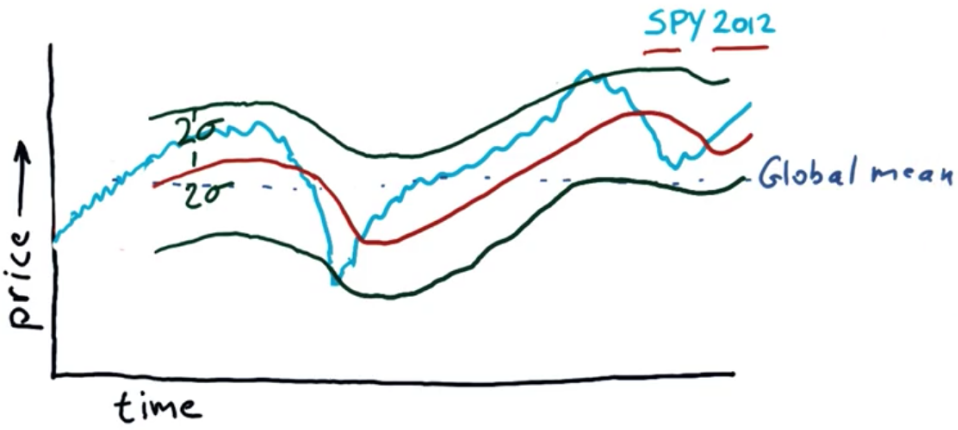


Figure 1.1: A visual demonstration of Bollinger bands® applied to $SPY. In this particular example, buying at the dip and selling at the peak would've rendered great profits, but there are plenty of examples in which this isn't an effective strategy.

A point of intersection, such as the dip in Figure 1.1, can indicate a trading opportunity. A double intersection (as in, it dips through the lower band and back up through it) with -2σ suggests a buy, whereas a double intersection with $+2\sigma$ suggest a sell. We won't discuss the effectiveness of this method as an actual trading strategy until a later chapter, but it's an important demonstration of the power of these simple global statistics.

1.1.2 Daily Returns

One of the most important statistics used in financial analysis is **daily returns**. It's very straightforward: how much did the price go *up* or *down* on a particular day? The calculation is very simple: the daily return at time t is based on the price that day divided by the prices on the previous day.

$$\text{return}_t = \frac{\text{price}_t}{\text{price}_{t-1}} - 1 \tag{1.1}$$

Snippet 1.2: Given the portfolio value (either as a `DataFrame`, `Series`, or other `pandas`data structure) in `port_val`, we can find the daily returns (as a percentage) rather easily by dividing the portfolio value by itself, offset by one day.

```
daily = (port_val[1:] / port_val[:-1].values) - 1
```

For example, if a stock was at \$100 on Monday and \$110 on Tuesday, the daily return would simply be: $^{110}/_{100} - 1 = 10\%$. We can plot the daily return for a stock over time to get a sense of its growth trend; it will generally zig-zag around zero, though its mean will be above zero if the stock price is increasing over time and below zero otherwise.

Figure 1.2: A potential daily return graph for an arbitrary stock.

Daily returns aren't very useful on their own relative to other global statistics like the rolling mean. Instead, they shine when *comparing* stocks: we can compare the daily returns of, for example, \$AAPL[1] against the S&P 500.

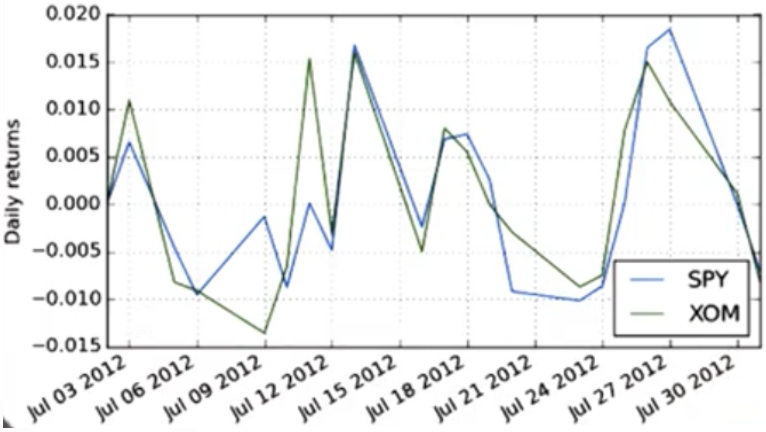


Figure 1.3: A comparison of the daily returns of Exxon Mobile (\$XOM) to the daily returns of the S&P 500 (\$SPY). We can see that it matches the ups and downs of the stock market in general quite closely.

1.1.3 Cumulative Returns

Another important statistic is **cumulative returns**. This is a measure of the stock price over a large period of time starting from some t_0. For example, you might say that the S&P has grown 10% in 2012; that would be its cumulative return (for t_0 = January 1$^{\text{st}}$ 2012). The calculation is almost identical to (1.1), except it uses a fixed initial time:

$$\text{return}_t = \frac{\text{price}_t}{\text{price}_{t_0}} - 1 \tag{1.2}$$

[1] Generally speaking, we use the notation \$SYM to indicate a stock symbol.

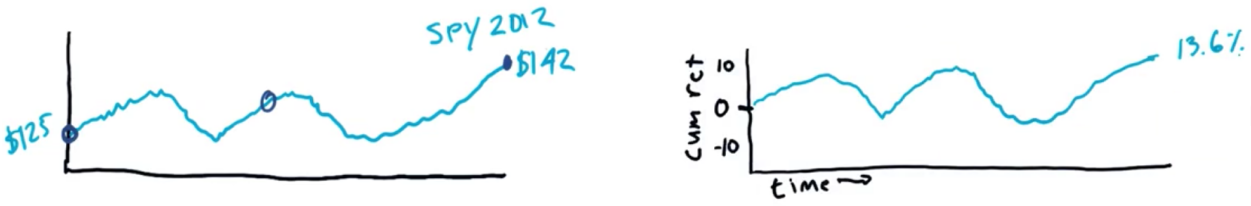


Figure 1.4: An example of calculating and subsequently charting cumulative returns for $SPY in 2012.

Snippet 1.3: As usual, we assume an existing portfolio value calculation in `port_val`. Note that the syntax `.iloc[idx]` is a way of directly accessing a particular row in a **`pandas DataFrame`** or **`Series`** object.

```
cumulative = (port_val.iloc[-1] / port_val_value.iloc[0]) - 1
```

1.2 Fixing Bad Data

We need to work around a number of (invalid) assumptions about financial data in order to manipulate it successfully. Many people assume stock ticker data is perfectly recorded minute-by-minute and has no gaps or missing data. The harsh reality is that the data is actually an amalgamation from multiple sources. A particular stock can trade at different (though similar) prices on different exchanges (like the NYSE or the NASDAQ).

Naturally, data can also be missing. Low-volume stocks that don't have a lot of trading activity might not have any data for a particular day; similarly, stocks come in and out of existence as the company (d)evolves.

The best way to work around gaps in a stock's price data is to **fill forward** then **fill backward**. To fill forward is to fill the price data from the last known price until trading activity resumes. This is far better than **interpolation** because interpolation is a "prediction" about the stock's price in the future which is a big no-no when it comes to analysis. After filling forward, there may still be an initial gap; for this, we fill backwards.

Fixing Bad Data in **pandas**

The key to fixing bad data in **`pandas`** is **`DataFrame.fillna`**, typically called with the parameter `method='ffill'` (or `'bfill'` for backwards filling).

1.3 Graphing Financial Data

We'll be discussing **histograms** and **scatter plots** in this section, and we'll start by taking a closer look at daily returns.

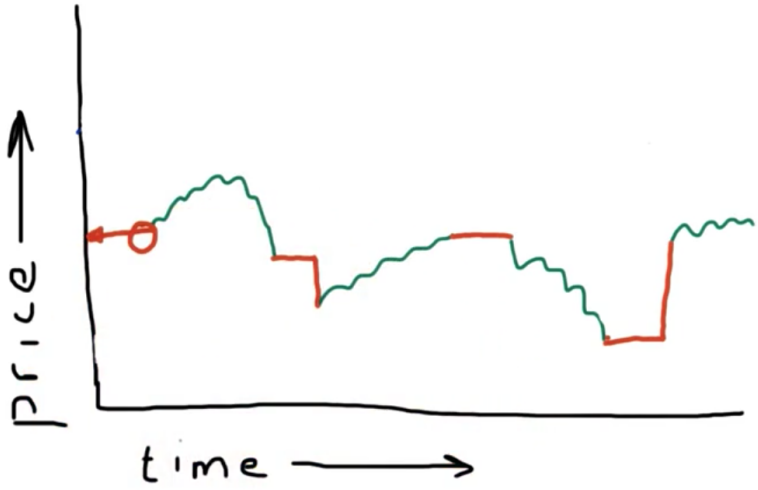


Figure 1.5

1.3.1 Histograms

On their own, daily return graphs don't offer us much. What we can do to get a better insight on a stock's behavior is to turn it into a histogram, which is essentially a bar graph that relates values to the frequency of their occurrences. For example, if there were 3 days in which you had 1% daily returns, you would have a 3-unit tall bar for $x = 1\%$. An important factor is choosing the granularity of the histogram (i.e. how many "buckets" of values we want to track).

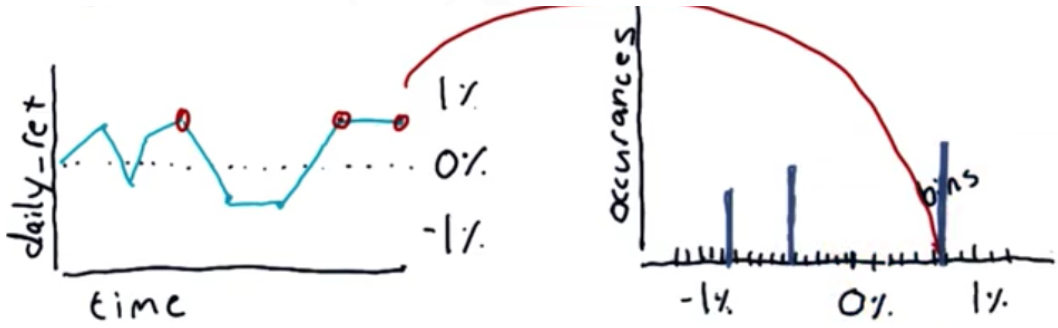
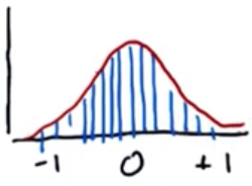


Figure 1.6: An example demonstrating how a daily return graph can be turned into a discretized histogram.

Suppose we looked at $SPY over many years and created a histogram, normalizing it to $[-1, 1]$. What would it look like? Interestingly enough, it results in a **normal distribution** (also called a Gaussian distribution).



With our neat little histogram, we can now compute interesting values like the mean (in the case of a normal distribution, this is its peak) and the standard deviation (which gives us a sense of how often and how far things deviate from the mean). Another important thing to note is the

measure of **kurtosis**. Kurtosis shows us how much our histogram would deviate from an actual perfect normal distribution.

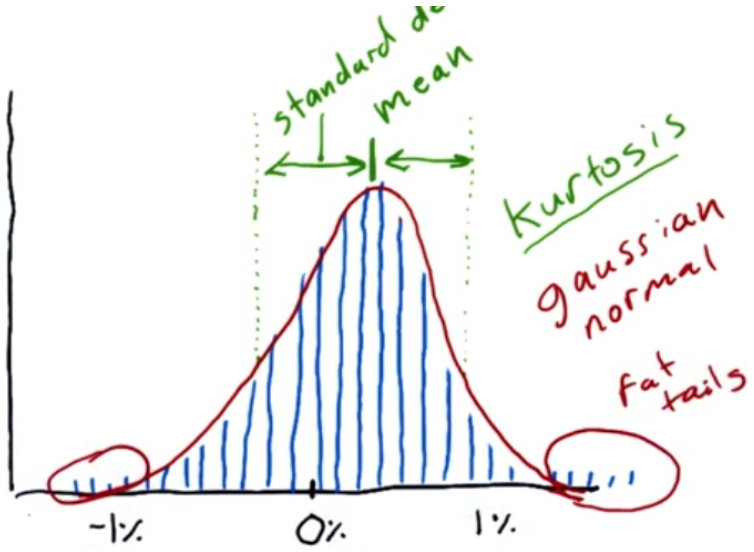


Figure 1.7: A visualization of kurtosis. Tails of the histogram that don't conform to the typical normal distribution (in the above example, the tails are too fat) speak to the deviation of outliers relative to the norm.

This is especially notable at the tails of the distribution; we can see "fat tails" in the histogram in Figure 1.7. This tells us there are frequently large outliers relative to a perfect normal distribution, indicated by a positive kurtosis.

Kurtosis has real-world consequences. If we assume that our models are perfect normal distributions, we mathematically disregard inconsistencies in the data. This is a big mistake that is amplified with the amount of outliers actually present in the data. Kurtosis contributed to the Great Recession of 2008: banks built bonds based on mortgages whose returns they assumed were normally distributed. Thus, they claimed that the bonds had a very low probability of default. This proved to be false as massive numbers of homeowners defaulted on their loans, precipitating the economic collapse.

Comparing the histograms of stocks is an important part of financial analysis. For example, consider the two histograms for $SPY and $XYZ:

We can see that $XYZ has a lower return and a higher volatility than $SPY: its mean is

lower and the standard deviation is larger. There is a higher spread of data, so there is more variation in daily returns, which is the definition of volatility.

1.3.2 Scatter Plots

A scatter plot lets us visualize differences between stocks at particular points in time by plotting them against each other. For example, in Figure 1.8 we can see the darkly circled area, which corresponds to both $SPY and $XYZ having a positive daily return, but $XYZ's is higher.

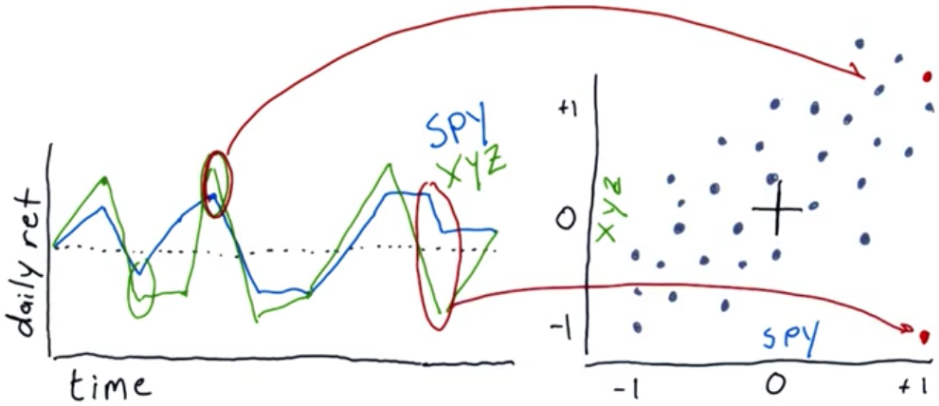


Figure 1.8: Turning the daily return plots of two stocks into a scatter plot.

Given enough data, we can typically see a trend. In Figure 1.8, we can see that there appears to be a linear relationship, however its quite loose as the dots are relatively spread apart. We can use linear regression[2] to fit a line to the plot. The **slope** of that line—referred to in the financial world as β when comparing to a general market-tracking stock like $SPY—describes how the stock reacts to the market. If $\beta = 1$, it means that when the market goes up 1%, that stock will also go up 1%.

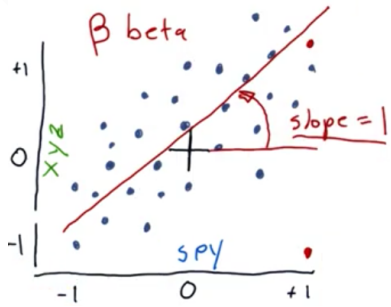


Figure 1.9: The slope of a line indicates a relationship between market trends and a specific stock (when comparing to $SPY).

[2] We won't cover linear regression for now. It should be a familiar concept, but if not, you can think of it as the mathematical way to find the "line of best fit" for some points.

Notice that the line of best fit in Figure 1.9 is *above* the origin. This value (the y-intercept of the line) known as α describes how much better a stock performs relative to the market on average. In this case, $XYZ often outperforms the market slightly.

Another important metric on the scatter plot is **correlation**. This is a measure of how tight the points are to the line of best fit, in the range $[0, 1]$. In Figure 1.9, the dots are typically fairly far from the line,[3] which means there is a **low** correlation. It's critical to keep in mind that slope $\neq$ correlation: it's just a measure of how tightly the dots fit a line of a particular slope.

Understanding Check: Correlation vs. Slope

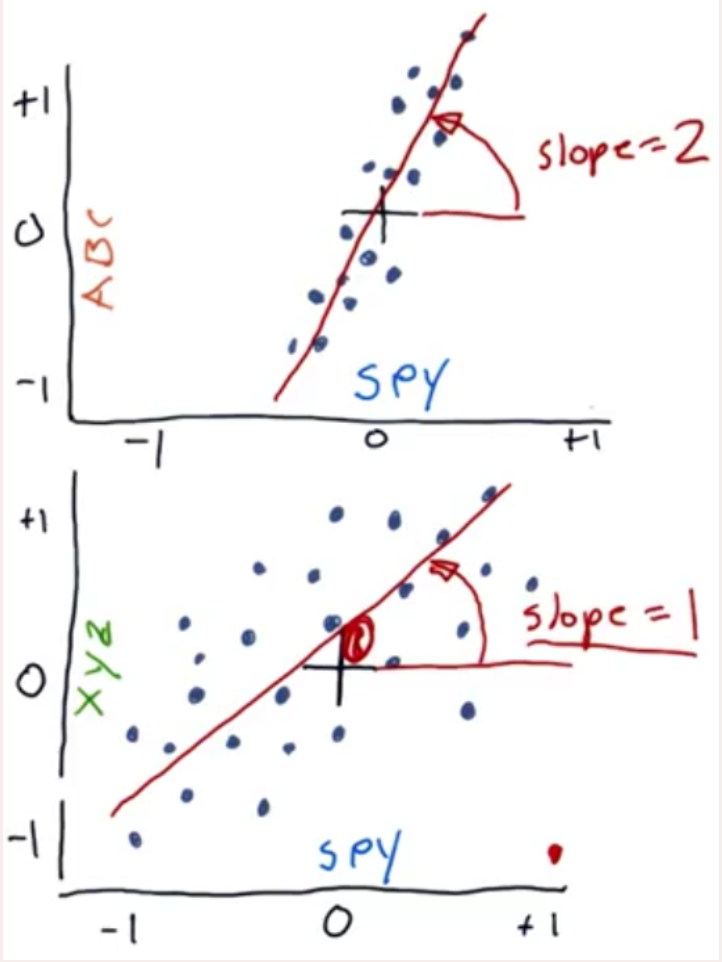
Below are scatter plots for two stocks in relation to the S&P 500: $ABC and $XYZ. Which of the following is true about $ABC?

(a) $ABC has a higher β (slope) and a lower correlation.

(b) $ABC has a lower β and a higher correlation.

(c) $ABC has a higher β and a higher correlation.

[3] Mathematically speaking, this would be a measure of the total error using something like the distance.



Answer: (c)

1.4 Portfolios

Let's expand beyond the realm of single-stock-analysis. We'll start out with a "buy-and-hold" strategy for our **portfolio** in which we have some initial principal investment spread out over a number of stocks; we will observe its behavior and performance over time. A portfolio is just a weighted set of assets. For example, your portfolio might be composed of \$AAPL, \$GOOG, and \$AMZN; the weight of each asset is the portion of total funds allocated to it.

Suppose our portfolio has the following setup:

- Principal starting value: \$1,000,000 *(I wish...)*
- Investment period: Jan. 1st 2009 to Dec. 31st 2011
- Stocks: \$SPY, \$XOM, \$GOOG, \$GLD

- Allocation ratios: $[0.4, 0.4, 0.1, 0.1]$

To calculate our daily portfolio value, we can use the following formula (assuming an initial data frame `prices` which has daily prices for our stocks):

$$\text{portfolio} = \sum \underbrace{\underbrace{\text{ratio} * \text{start}}_{\text{per-stock principal}} * \underbrace{\frac{\text{prices}}{\text{prices}[0]}}_{\text{normalized prices}}}_{\text{daily position}} \tag{1.3}$$

Let's assume now we have a data frame `port_val` which is the list of portfolio values over time and the data frame `daily_rets` which is its daily returns. We can now calculate some statistics that everyone wants to know about a portfolio: the cumulative and average daily return, the standard deviation of the daily return, and the **Sharpe ratio**.

The cumulative return is simply the percentage of total portfolio growth, so its final value minus the initial value. The next two we have already discussed. The Sharpe ratio gives us an insight about our returns in the context of risk. Finance folk often associate risk with the standard deviation as that represents a stock's volatility (hence why cryptocurrency investment is risky: your coin could be worth $100 tomorrow or $1000 or $3).

Snippet 1.4: In this snippet, we assume the existence of a `DataFrame` of stock data called `prices` that's formed as we've discussed: each row is labeled by a date and each column corresponds to a $SYMBOL. We'll also assume that `allocs` is our ratio of allocations and that `principal` is our initial (total) investment.

```
normalized = prices / prices.iloc[0]     # normalized prices
allocated = normalized_prices * allocs   # prices scaled by allocation ratio
positions = allocated * principal        # per-stock investment
portfolio = positions.sum(axis=1)        # portfolio value every day
```

Naturally, given two stocks with similar returns, we'd choose the one with less volatility; similarly, given similar volatility, we'd choose higher returns. What about a stock that is more volatile but has higher returns than another? This typically involves a bit of a "gut feeling," but we want to do better than that.

That's where the **Sharpe ratio** comes in: it gives us a **risk-adjusted** return. All else being equal, it considers low levels of risk and high returns to be good. The Sharpe ratio additionally considers the risk-free rate of return which is critical when comparing financial strategies. A risk-free investment is something like an FDIC-insured bank account or a short-term treasury, which are (more-or-less) guaranteed to grow at a particular percentage. If your investment strategy doesn't outperform a risk-free return rate, you might want to reconsider it.

Given the portfolio return R_p, the risk-free return rate R_f, and the standard deviation of R_p (i.e. the volatility or risk), we can formulate the Sharpe ratio as such:

$$\frac{R_p - R_f}{\sigma_p}$$

Notice that as volatility goes up, the ratio shrinks. Similarly, as the return goes up (sans the risk-free return), the ratio grows. The actual formula is a little more complicated, but describes the same pattern (here E is the expectation):

$$S = \frac{\mathrm{E}[R_p - R_f]}{\mathrm{std}[R_p - R_f]} \tag{1.4}$$

The "risk-free rate" can be taken from LIBOR, the 3-month treasury bill, or just set as 0% which has been an accurate approximation lately. Instead of having to constantly check and update the aforementioned values every day, we can approximate the daily risk-free rate by taking the 252^{th} root of the annual rate (remember, there are 252 trading days in a year): $R_f = \sqrt[252]{1.0 + R_{f_{\text{annual}}}} - 1$.

The Sharpe ratio can vary wildly depending on how frequently its sampled. It was originally envisioned as an **annual** measure. If we want to sample more frequently, we need to add an adjustment factor k to make it all work out, where k is the square root of the number of samples per year. For an R_p that represents daily portfolio returns, then, $k = \sqrt{252}$. Similarly, $k = \sqrt{52}$ for weekly samples, etc.

EXAMPLE 1.1: **Sharpe ratio**

Suppose we've had our portfolio for 60 trading days. Our average daily return is 1/10% per day (0.001%, or 10 **basis points**[a]). Our daily risk-free return is 2 basis points, and our volatility (σ) is 10 basis points.

What's the Sharpe ratio of this strategy?

Since we are operating in days, we know $k = \sqrt{252}$. Then, we just use (1.4) directly:

$$S = \sqrt{252} \cdot \frac{10 - 2}{10} = 12.7$$

[a] Basis points are often called "bips," and their unit is bps.

1.5 Optimizers

Optimization problems are a class of mathematical problems that require us to solve for a value or values that maximize (or minimize) an equation. In this section, we'll be discussing optimizers in code that can solve these types of problems in a financial context.

An optimizer can:

- find the minimum values of a function,
- build parameterized models from data, and
- refine stock allocations in a portfolio

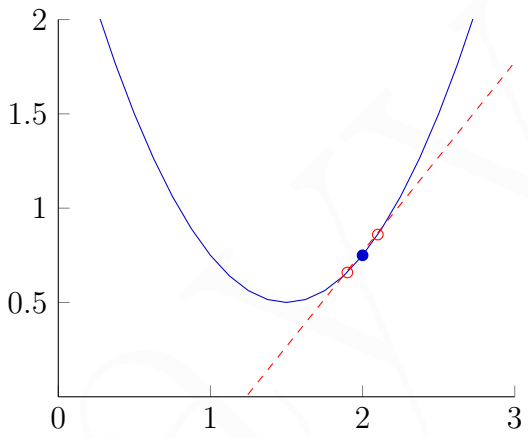
Using a pre-built optimizer is fairly simple. All we need to do is provide a function to minimize (such as $f(x) = x^2 + 5$) and an initial guess.

1.5.1 Using an Optimizer

Let's walk through a simple minimzation example. Suppose we're given the function (plotted below):

$$f(x) = (x - 1.5)^2 + 0.5$$

How does the minimizer work? Suppose our initial guess was $x = 2$. At that value, it uses the neighbors (like $x = 1.9$ and $x = 2.1$) to approximate a local slope and "marches downhill" (this is gradient descent). It repeats the process with another value that is down this slope, eventually converging to the minimum value $x = 1.5$.



Gradient descent is just one of a myriad of ways to minimize a function; it's an easy configurable parameter to `scipy.optimize.minimize`.

Unfortunately, this isn't a bulletproof method. Functions with many **local** minima and flat areas are tough for minimizers to solve. Specifically, **convex functions** are the easiest for optimizers to solve. Formally-speaking, a convex function is one in which you can draw a line segment between *any* two points and that line would not intersect the function. A convex function must have just one minimum and not have any problematic flat areas.

1.5.2 Building Parameterized Models

Our goal in this section is to build a parameterized model from data.

Those of you who have taken a course on computer vision may have covered a similar topic: instead of treating a line as an equation $y = mx + b$ in which m and b are known, and we can plug in arbitrary x values to get their corresponding ys, we instead already *have* a set of $\{(x_0, y_0), (x_1, y_1), \ldots\}$ data points and wish to solve for m and b.

For convenience and generalization in code, we'll use c_i to refer to our parameters, so $c_1 = m$ and $c_2 = b$ in our aforementioned parameterized model of a line.

Error Metrics

If the data fit our model perfectly, solving the parameters would be trivial. It's unlikely, though, that all of our points would be perfectly colinear, for example. Thus, we need a way to evaluate the quality of a "candidate model."

Given our understanding of minimizers, we want to reframe the "model of best fit" problem into something a minimizer can process. A common go-to error metric is simply the total vertical error from the points to the line.

Our error function in this case is the vertical distance between point (x_i, y_i) and the line, for all points. $mx_i + b$ is the "real y" at that x value via the true equation of the line:

$$E = \sum_{i=1}^{n} (y_i - (mx_i + b))^2$$

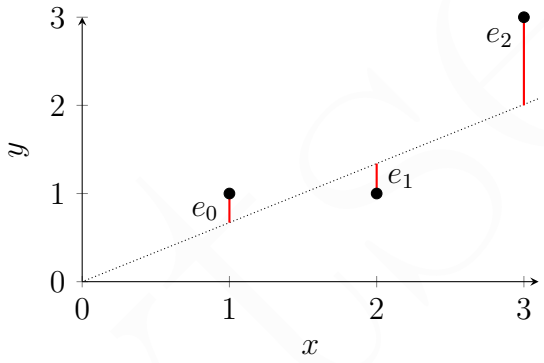


Figure 1.10: Fitting a line to a set of points, with the errors in red.

We can feed the minimizer the above error function applied to our simple line function: $f(x) = c_1 x + c_2$, resulting in the parameters that best fit our data.

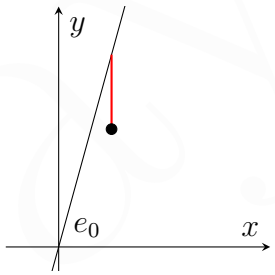


Figure 1.11: Measuring vertical error over-emphasizes errors in steep lines, but don't let that distract you from the fact that the Warriors blew a 3-1 lead in the 2016 Finals.

This error function isn't perfect: it over-emphasizes the error of steep lines because it only considers the y component, as seen in Figure 1.11. A better function would be distance, or *perpendicular* error, but we won't get into that here because apparently it's too advanced of a topic for a graduate-level course on financial algorithms.

Snippet 1.5: We can find the best-fitting model to a set of observed data by feeding the model function into SciPy's optimization toolkit, specifically `scipy.optimize.minimize`.

```
import numpy as np

def error_function(line, data):
  """ Computes the error between a line (m, b) and observed data.
```

```
    line:   a 2-tuple (c0, c1) where c0 is the slope and c1 is the y-intercept
    data:   a 2D array where each row is an (x, y) point
    """
    return np.sum((data[:, 1] - (line[0] * data[:, 0] + line[1])) ** 2)
```

n Dimensions

This process is extensible to n-dimensional data fairly simply. It does rely on the key assumption that the function can be modeled as an n^{th}-degree polynomial. That's okay for our current purposes, though, because we'll be applying this technique to optimize a portfolio which can be viewed as a simple function of allocation ratios applied to prices.

Snippet 1.6: We can find the best-fitting polynomial function to a set of observed data much like in Snippet 1.5.

```
import numpy as np

def error_function(C, data):
    """ Computes the error between a line (m, b) and observed data.

    C:      np.poly1d object (or 1D array) representing polynomial coefficients
    data:   2D array where each row is an (x, y) point
    """
    return np.sum((data[:, 1] - np.polyval(C, data[:, 0] + line[1])) ** 2)
```

1.5.3 Portfolio Optimization

Given a set of funds (i.e. stock symbols), assets (i.e. an initial amount of money to invest), and a time period (e.g. 1 year), find the ratio of assets to funds that maximizes performance. "Performance" here is the variable that depends on your investment strategy. If all you care about is growth, for example, you'd want to maximize cumulative returns (see Snippet 1.3).

In our running example, we'll optimize the portfolio's Sharpe ratio instead. The set of variables we wish to optimize is the allocation ratios across our chosen stock symbols. For example, we might want 20% of our funds to \$AAPL, 10% to \$GLD, and the rest to \$XOM for a nice, diversified portfolio. Of course, this might not be the optimal ratio. Framing this problem is a straightforward process:

- Provide a function for `minimize()` to work with.

  We're trying to find the optimal allocations across a fixed set of stocks. Thus, in this function we'd need to calculate the Sharpe ratio for a given "best allocation" guess. Because we're using a *minimizer*, though, we need to multiply our result by -1.

- Provide an initial guess for our allocation ratios.

  For this parameter, many values are likely to result in the same resulting optimal ratio. It might make sense to pick your true best guess, your ideal resulting ratio, or just a

uniform distribution (like 33% across our above 3 stocks).

- Set up our ranges and constraints.

  Obviously, we don't want to try any input values that result in allocations beyond 100% of our funds. In other words, our constraint must require that the sum of the ratios is 1. Similarly, we can restrict the possible inputs to our function to fall in $[0, 1]$.

- Call the optimizer. *(I don't think this bears more explanation.)*

PART II

Computational Investing

Hedge Funds

In this part of the course, we'll open up by discussing the different types of hedge funds. Following that, we'll look at how the market works, then talk about evaluating a company's worth and how that can be reflected in its stock price.

Let's cover a few vocabulary words here before we dive into details.

- **Liquidity** is a measurement of how easy it is to buy or sell shares in a fund.

  As we'll see soon, ETFs, or **e**xchange-**t**raded **f**unds are the most liquid of funds. They can be bought and sold easily and near-instantly during the trading day just like individual stocks; ETFs, though, represent some distribution of stocks. The **volume** of an ETF is just as important to its liquidity: because there are often millions of people trading it, it's easy to get your buy / sell order filled.

- A **large-cap stock** like Apple refers to a stock with a large **market capitalization**.

  Market cap is a metric of a stock's total shares times its price. It's worth noting that the price of a stock has no relation to the value of a company; it only describes the cost of owning a single share *in* that company.

  If you can afford the market cap of a company, you can afford to buy the company in its entirety and take over its ownership.

- A **bull market** or a **bullish** position on a stock is an optimistic viewpoint that implies that things will continue to grow. On the other hand, a **bear market** or a **bearish** position is pessimistic (or cautionary, or realistic, depending on how you see the glass) about the future of an asset.

2.1 Types of Managed Funds

With that in mind, managed funds can be broken down into three categories:

ETFs These funds function almost identically to stocks. Though they represent buckets of various ratios of stocks rather than shares in a single company, they can still be bought and sold just like regular stocks during the trading day. They are (usually) very liquid, especially if they have a high market capitalization. They often track either an entire

market (like $SPY is an ETF tracking the S&P 500) or a specific sector. Furthermore, they're completely transparent: their structure, strategy, and distribution of allocations is publicly available.

For example, $SMH, the VanEck Vectors semiconductor ETF, tracks various semiconductor producers. Their holdings are publicly available (with their highest allocation ratio falling onto Intel $INTC at 12.3%) and are tradeable via most brokers and trading platforms.

Mutual Funds These funds are typically less transparent and less liquid than ETFs. Their allocations are disclosed quarterly, and they can only be traded at the end of a trading day. In line with that, their strategies are often much less transparent as well. Often, you have to sign an NDA or some other "hush-hush" agreement to buy shares in a mutual fund; generally-speaking, there's a much higher barrier to entry relative to an ETF.

Hedge Funds These funds, the biggest kahuna of them all, are highly secretive about their strategy and don't have to disclose their allocations or plans at all. They entice people to invest in them by demonstrating results—if they can show you a 30% gain in value over the last quarter, do you *really* care what their strategy is? Yeah, me neither.

Of course, there is evidence that hedge funds generally don't outperform mutual funds, but that's beside the point here. ☺

Table 2.1 summarizes the three categories. Now besides the difference in liquidity, composition, and transparency, why would you want to be the fund manager of one of these or another? Let's talk about the only thing that matters—**compensation**.

ETFs	Mutual Funds	Hedge Funds
Bought/sold like stocks	Bought/sold end of day	Bought/sold by agreement
Are backed by "baskets" of stocks	Disclosed quarterly	No disclosure
Transparent and liquid	Less transparent	Not transparent

Table 2.1: A brief summarization of the differences between the three types of managed funds.

2.2 Compensation

Let's discuss how the managers of each of these funds is compensated for their hard work of deciding how to spend other people's money. First let's introduce the idea of **assets under management** or AUM, which is the amount of other people's money the fund manager is responsible for.

In ETFs, managers are compensated by an **expense ratio** which is often simply a percentage of the AUM. They're generally pretty low: anything from 0.01% (1 "bip") to 1% is typical (though the higher end is more unusual).

In mutual funds, the managers are also compensated by an expense ratio, though it's much higher than that of an ETF. They typically range from about 0.5% to 3%. If you ask a mutual fund manager why they get compensated more, they're tell you it's because their job requires more skill than an ETF's manager. An ETF is typically designed to track an index; the manager of $SPY, for example, just needs to make sure s/he allocates the stocks in the S&P 500 as they are represented. Because a mutual fund has more discretion about the ratios in its portfolio, it can incorporate research, deeper insights, etc. into its strategy to get better returns than an index.

Hedge funds follow an "old school" model known as **two and twenty**: they get 2% of the AUM as well as 20% of the profits. Now that's quite a big chunk of the profits; however, if you're an investor and you're making more money than you would be at a mutual fund (even after the manager's cut), why do you care?

> Example 2.1: **A Hedge Fund Manager's Salary**
>
> Imagine you're a hedge fund manager managing 100 million dollars. Why are you taking this class again? Oh right, we're just imagining... Okay, so suppose you had a 15% return this year. How much would you be compensated?
>
> Well, the "two"—two percent of the AUM—comes out to $\$100\text{M} \times 0.02 = \$2\text{M}$, plus another "twenty"—twenty percent of the profits–or $\$15\text{M} \times 0.20 = \$3\text{M}$, meaning you made $5M this year.
>
> Now stop imagining and snap back to cold reality.
>
> A common follow-up to this might be, "Is the 'two' calculated on the initial principal of $100M or the end fund balance of $115M?" Naturally, this depends on the hedge fund's compensation policy. In reality it'll probably be somewhere between the two as they take snapshots of the value of the fund throughout the year.

Modern-day hedge funds typically don't offer this classic 2/20 rate anymore. More common is things like 1/10 or somewhere inbetween, but the split compensation structure usually remains the same.

With the compensation model in mind, what motivates each respective fund manager? Well with just AUM factoring into the expense ratio, increasing that number is basically the only motivation for managers of expense-ratio-compensated funds—ETFs and mutual funds. Hedge funds, on the other hand, are additionally-motivated by profit, and so are consequently also motivated by risk-taking strategies.

Let's move forward assuming we want to be hedge fund managers since it seems like the most glamorous option and look deeper into what that entails.

2.3 Attracting Investors

A hedge fund with no investors is meaningless. How can we make money without risking someone else's? *Invest our own*? Unbelievable. So let's talk about how we can ~~con~~ attract some people to give us their money to invest.

For whatever reason, hedge funds typically don't have more than 100 investors. As a result, they generally want each one to offer them a fairly large principal investment. So **who** are they targeting? Investors can be broken down into three categories: (very wealthy) **individuals**, (very large) **institutions**, and **funds of funds**.

Now **why** would anyone trust someone like you with their wealth? At the very least, you need to offer them a **track record** of success in the past, a convincing **simulation** and **story** for your investment strategy, and a **portfolio fit** that aligns with their investment goals. If you want to go all-in on the technology startup sector whereas Georgia Tech's foundation needs a stable place to keep their money, it might not be a good fit.

In addition to strategy, variety might be important to keep in mind. Diversity is key in a good portfolio that can endure market dips and general volatility. If an investor already has a particular area covered (like small-cap growth stocks, for instance), if you can only offer them a similar sector, things might not work out.

2.3.1 Goals

Obvious your goal as a hedge fund manager is to make money. Things can be a little more nuanced than that, though. Some common goals that are helpful in attracting investors include:

- **Beating a benchmark**: if our portfolio can consistently beat the S&P 500 by choosing the best-performing stocks of the bunch, that can be a good and convincing benchmark.

- **Absolute return**: no matter what happens, we want a way to ensure a positive return overall. This is typically accomplished by "going long" (investing long-term) on good stocks and "selling short" (betting on declines on bad stocks in the short-term) on bad stocks. We'll discuss shorting later.

Your choice of goal as a hedge fund manager should depend on your experience and expertise. For example, if you're really good at picking stocks in emerging markets overseas, your best bet is targeting beating an index that tracks a similar investment strategy.

2.3.2 Metrics

Given these goals, how can we measure our fund to see if our portfolio meets them? Well we already discussed quite a few metrics in the previous minicourse. Cumulative returns, volatility, and a measurement of the risk / reward (like the Sharpe ratio) are all reliable and convincing metrics.

2.4 Hedge Fund Computing Architecture

Hedge funds and high-frequency trading firms are some of the most computationally-demanding environments out there. There is a massive interconnected infrastructure that connects real-time market data, massive databases of historical pricing data, and state-of-the-art machine learning models to create trading algorithms. Figure 2.1 maps out a (very) high-level architecture for a typical firm.

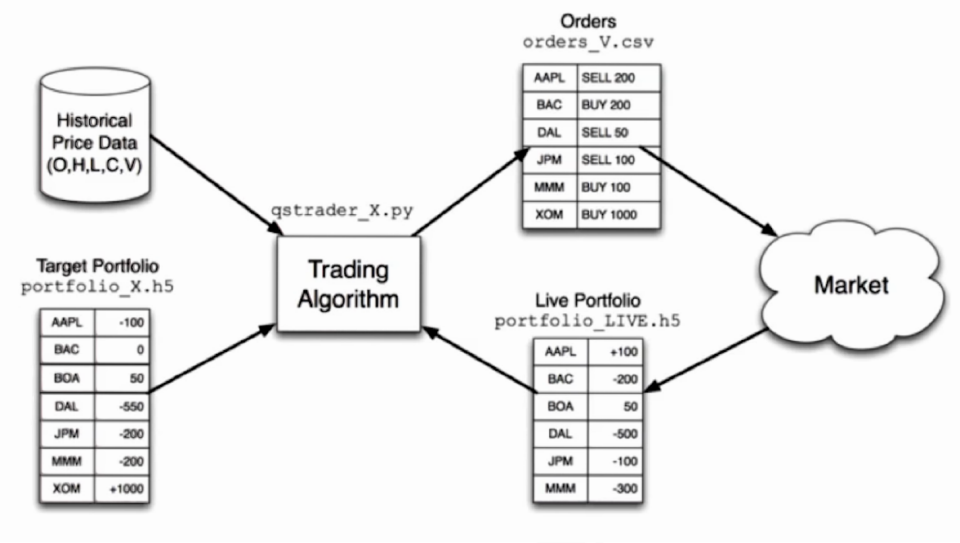


Figure 2.1: The architectural structure of a hedge fund.

The current prices constitute a real portfolio. The trading algorithm uses that as well as real-time data about the market (pending orders, etc. which we'll get to shortly) to send out orders that make progress towards matching the *target* portfolio. There are complexities in this "progress-making" process: if your target portfolio wants 10,000 shares of $AAPL, buying those 10,000 shares outright would be detrimental to our portfolio and probably make us change our target portfolio midway. This is because buying such a large quantity of shares will make the price rise as the market price goes up to meet demand. Hence there is a financial benefit to incrementally reaching the target portfolio with minimal market effecets.

We'll work through this architecture backwards, starting from the market itself and eventually getting into the nitty-gritty inner workings of a machine learning model applied to market data.

The Order Book

What happens when you click "buy" on a share of stock through an online broker?[1] Probably a lot more than you might imagine. Let's dive into the details and discuss orders and the order book.

3.1 Making Orders

An **order** is a well-defined structure that contains very specific instructions about the stock to buy. It is made up of:

- **Action**: this is either a **BUY** or **SELL** instruction.
- **Symbol**: this is the unique stock symbol (like $AAPL or $SPY) to execute the order on.
- **Quantity**: this is the number of shares to purchase.
- **Order Type**: this is either a **market** or **limit** order. There are significant differences between these two order types.

  A market order does not specify a price for the order. It generally indicates that you are willing to accept a "good price" for a stock. In reality, though, it's filled at whatever price someone is willing to accept the order.[2] This can vary wildly during volatile market times. For example, if $TSLA stock is plummeting because of Elon's latest Twitter rant, your market sell order might get filled at far below the "current" price you saw in your broker's stock ticker.

  A limit order, on the other hand, *does* specify a price for the order. It guarantees that when the order is executed, it will be executed at that price. For buy orders, it can also

[1] By the way, if this course has motivated you to start investing, RobinHood is a great way to get started. It has no trading fees (!) and is extremely mobile-friendly. If you use my invite link, we'll each get a free stock! https://share.robinhood.com/georgek84

[2] This is typically where high-frequency trading firms will take advantage of their literal physical proximity to the exchanges' order books to fill market orders for slightly more than the current price and make a few pennies in profit. Well, a few pennies per share for a few million or billion orders a day... Check out *Is the stock market rigged?* for details on this surprisingly-legal phenomenon.

be filled *below* that price; similarly, for sell orders, it can be filled *above* your specified sale price. The outcome is guaranteed to be what you specify or better, however, it's entirely possible that your order will *not be filled* in the reasonable time frame you expect. During the same aforementioned period of volatility, if you ask for price x on your stock, but the market has fallen far below that point, it may take months or years for the share price to recover to x and thus for your order to be filled.

- **Price**: for limit orders, this specifies the precise minimum value you're willing to pay or receive for your order.

Let's take a look at how a typical market order will be filled.

> Fun Fact: **Other Order Types**
>
> Your online broker may offer you some other order types such as **stop loss** or **stop limit**. These are actually not directly supported by the market but are special orders that your broker automatically converts to one of the above types when the condition is fulfilled.
>
> For example, a stop-loss sell order specifies that once a stock drops below a particular user-defined price, it should be executed as a market sell. It is generally used to, as the name suggests, *stop losse*s as a stock drops in value. You might, for example, never want to risk losing money on an investment, so you could set a stop-loss order to be at (or slightly below) the price you purchased the stock at; your broker would convert and execute the sale any (!) time the stock dipped that low.
>
> The most impactful and well-known of these is the ability to sell short, which we'll discuss in further detail soon.

3.1.1 The Order Book

Let's suppose we executed the following order on the exchange, "Buy 100 shares of $IBM at a limit price of $99.95":

```
BUY, IBM, 100, LIMIT, 99.95
```

Suppose this is actually the first order of the day. This action adds a `BID` entry to the *public* **order book** for the exchange:

Type	Price	Quantity
BID	99.95	100

The order itself is anonymized, but it becomes public knowledge that there is interest in 100 shares of $IBM. It's important to note that identical orders are lumped together. If another person comes along asking for another 200 shares at that same price, the order book would look like this:

Type	Price	Quantity
BID	99.95	300

NOT like this:

Type	Price	Quantity
BID	99.95	200
BID	99.95	100

Only your broker maintains the relationship between identities and orders in the exchange's order book.

Let's see what happens when a sale, or ASK, entry comes in. Suppose you submit an order to buy 1000 shares of $IBM with a limit price of $99.95. There's nobody selling $IBM at that price, so it just gets added to the order book.

Type	Price	Quantity
ASK	100.00	200
BID	99.95	1000

Suppose that after some time, the order book looks like this:

Type	Price	Quantity
ASK	100.10	100
ASK	100.05	500
ASK	100.00	1000
BID	99.95	1000
BID	99.90	50
BID	99.85	50

Then a market order arrives to buy 100 shares of $IBM. Remember, market orders don't specify a price, so they can always be filled (well, provided that there *are* sellers). The order is filled by the cheapest "ask" order, resulting in the client getting their shares for $100 each.

Type	Price	Quantity
ASK	100.10	100
ASK	100.05	500
ASK	100.00	~~1000~~ 900
BID	99.95	1000
BID	99.90	50
BID	99.85	50

The gap in price between the asks and bids is called the **bid-ask spread**, representing the supply and demand for the asset. The "current value" of a stock is somewhere in this spread.

Example 3.1: **How Does the Spread Reflect Stock Value?**

Now that you know how an order book works, let's consider how we can use information about it to learn about the state of the market. Given the following order book, do you think the price of the stock is going to go *up* or *down*?

Type	Price	Quantity
ASK	100.10	100
ASK	100.05	500
ASK	100.00	1000
BID	99.95	100
BID	99.90	50
BID	99.85	50

The stock is much more likely to go down in price because of the selling pressure. Consider the difference between someone placing a market-buy vs. a market-sell order for 500 shares. For the buy, you would immediately get rid of all of the bids in the order book, meaning the price goes down. On the other hand, for the sell, the price wouldn't budge at all; you would only get rid of 500 shares in the cheapest ask.

Let's walk through the changes that occur to the order book through a handful of order examples. We will start with the order book from the previous example.

Type	Price	Quantity
ASK	100.10	100
ASK	100.05	500
ASK	100.00	1000
BID	99.95	100
BID	99.90	50
BID	99.85	50

The following orders come in (oldest first):

```
BUY,  100, Market
BUY,  100, Limit,  100.02
SELL, 175, Market
```

The resulting order book is as follows:

Type	Price	Quantity
ASK	100.10	100
ASK	100.05	500
ASK	100.00	800
BID	99.85	25

Do you follow the flow? The first order is filled by the cheapest ask, dropping its quantity to 900. Then, the second order is likewise filled by the cheapest ask—remember, a limit-sell

guarantees the specified price *or better*—dropping it to 800. Finally, the third order fills out the first bid completely, then the second bid completely, and even 25 shares in the third bid. All together, the client would probably have their market purchase "cost" be the average across the three orders it used:

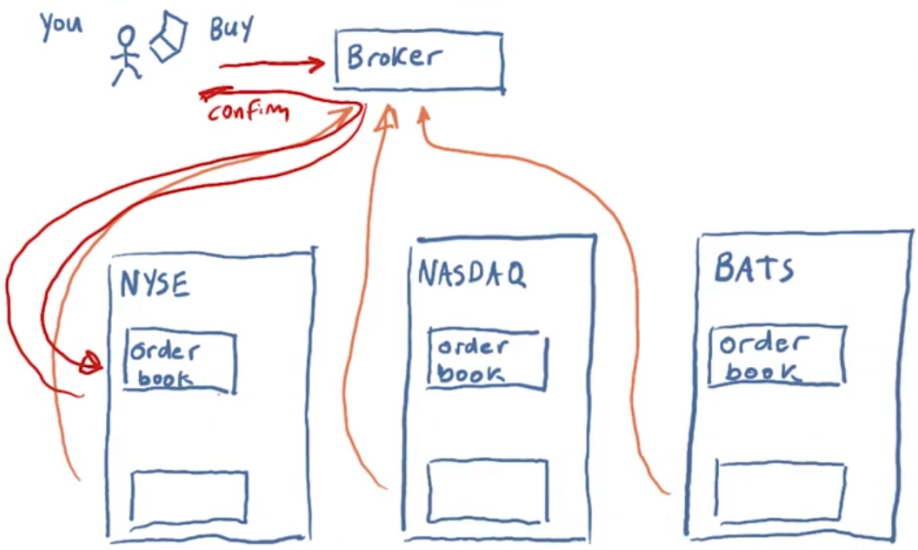
$$\begin{aligned} \text{cost} &= \frac{(99.95 \times 100) + (99.90 \times 50) + (99.85 \times 25)}{175} \\ &\approx 99.92 \end{aligned}$$

We can concretely see why the stock was more likely to decrease in price prior to this transaction, as explained in the previous example. At the end of these orders, the price is somewhere between \$100 and \$99.85 rather than between \$100 and \$99.95 as it was previously.

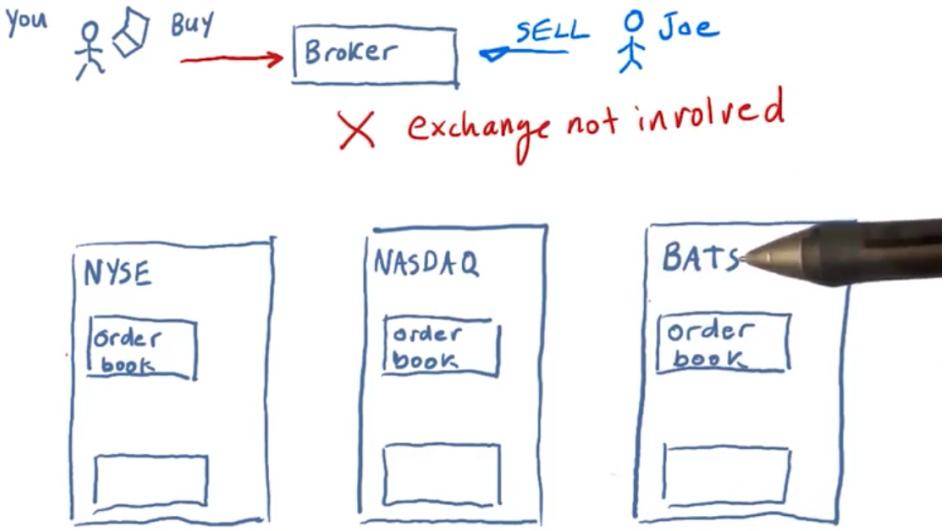
Figure 3.1: The order book of a real stock using a real trading platform.

3.1.2 Filling Orders

The path your order takes varies wildly depending on a number of factors. There is always an intermediary between you and a filled order: your broker. Your broker is connected to a number of exchanges (like the NASDAQ or NYSE). Certain stocks might be listed at one or more exchanges. Your broker has devices at each exchange that query the order book and report back results to the broker's "home base" where it gathers and analyses that data to give you the best price. It executes your order on the best exchange and gives you a confirmation.



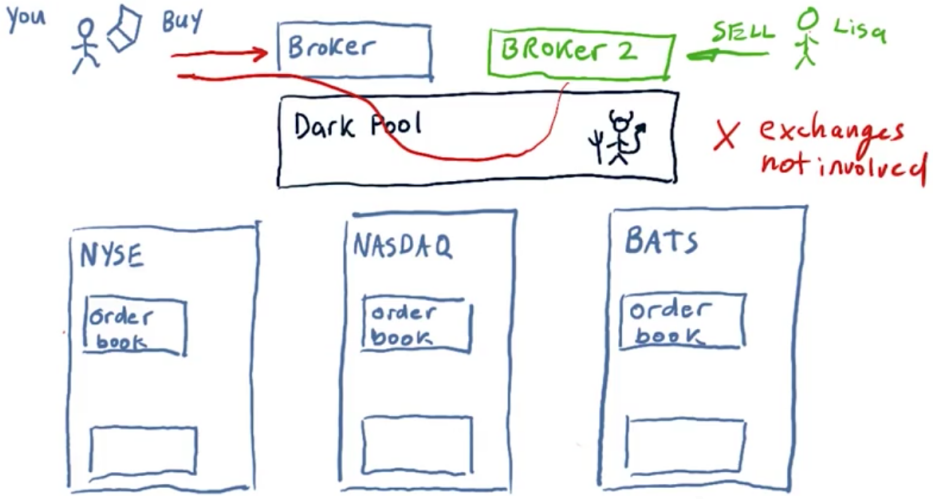
This is the simplest, clearest scenario. What if there are multiple clients using the same broker to make orders, and you want to buy the same stock that Joe Shmoe is selling? The trade can be made within the broker without having to contact (or pay fees to) the exchanges for their order information.[3]



Can we extrapolate this scenario across brokers? Turns out there's an entity out there called the **dark pool**, which is an intermediary between brokerages and exchanges. The dark pool often makes predictions about price movement; they pay the brokerages for the privilege of seeing orders before they're executed, and will actually execute advantageous ones without involving exchanges. Both of these entities—the brokerages and the dark pool members—claim this is kosher, legally-speaking, because they are still guaranteeing at least as good

[3] By law, the broker *must* give you a price as good as you would get on the exchange.

of a price as is available on the exchanges themselves.[4] Apparently around 80-90% of the trades made by "retail traders" never actually make it to the exchanges.



3.1.3 Exploiting the Order Book

There are far more avenues for exploitation and (legal) skimming available than the simplistic happenings between the dark pool and brokerages we talked about in Filling Orders. Let's talk about how *hedge funds* can exploit inefficiencies in this system.

Exploit 1: Faster-Than-Light Travel

We're located in Seattle and use e-Trade as our broker. That means our view of the order book, as well as any orders we make, are out-of-date by the amount of time it takes for that information to get to (or from) us. If we traveled at the speed of light, this would be almost 13ms one way, but since we have to deal with traveling through thousands of routers, switches, NSA taps, etc. through inefficient copper or even fiber optic cables, it's more like 76ms on average.[1, 2]

Hedge funds exploit their geographical advantage by having a device at the exchange itself, giving them instant access to order book changes.

With this massive advantage in reaction time, the hedge fund can make predictions about the price of a stock and place orders ahead of *your* prediction. For example, it might buy the stock as you're buying it, then sell it to you (12ms later) for a few pennies more.

Exploit 2: Geographic Arbitrage

Suppose there is a stock listen in two exchanges: the NYSE and the London Stock Exchange. They operate independently, so there may occasionally be a small difference in the price of a

[4] Of course, by pooling together orders they get a better view of the market as a whole than the exchanges or any independent trader ever would, but I guess they ~~bribe~~ lobby the government enough to turn a blind eye to that competitive, monopolistic advantage.

Figure 3.2: Demonstrating the geographical advantage that hedge funds have. Your actions take at least 12ms to reach the market as they travel from Seattle to New York (even ignoring the pit stop at your broker in Atlanta), but the machine(s) that hedge funds have co-located on the exchange can beat you to any action every time, making a small profit every few milliseconds.

stock from one exchange to another. A hedge fund can have servers at both exchanges wired with an ultra-high-speed, dedicated connection. When any of these differences arise (such as NYSE's price going below London's), it immediately buys shares on the NYSE exchange and sells shares on the London exchange (either the same existing ones, depending on whether or not it already owns some).

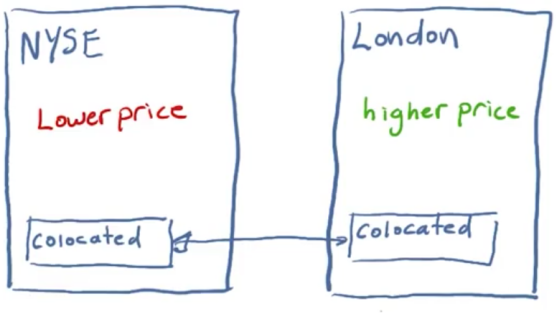


Figure 3.3: Demonstrating geographic arbitrage. A hedge fund uses its dedicated connection to take advantage of price differences between exchanges.

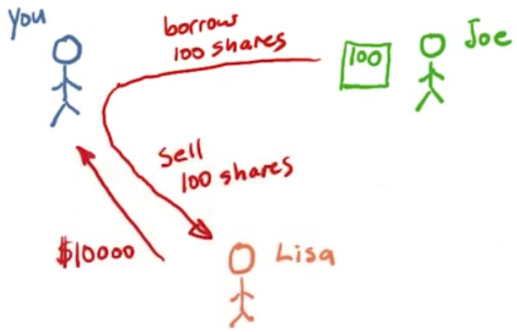
Because this process occurs and is well-understood, such price differentials rarely arise, but when they do, exchanges are there to pick the pennies up off the ground.

3.1.4 Selling Short

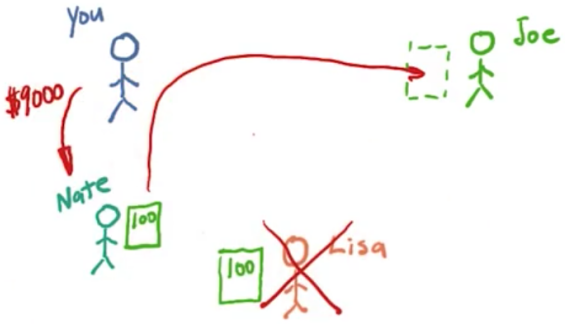
The most important and impactful order type "added" to the stock market by brokers (in addition to those we described previously) enables **selling short**—the bane of /r/wallstreetbets. Shorting a stock involves taking a *negative* position; you bet that its price will go down rather than up.

Suppose $IBM is selling at $100. You think $IBM is overhyped and want to short it. Lisa, on the other hand, thinks its still got plenty of growth ahead and wants to buy in. Joe holds 100 shares of $IBM, and Joe is a great guy. Even though he wants to hold onto his shares, he's willing to lend them to you (well, his broker is).

You borrow Joe's shares and sell them to Lisa; she gives you $10,000 in exchange. At the end of the day, you now have $10,000 in your account and owe Joe 100 shares of $IBM.



Joe is eventually going to want his shares back. Suppose $IBM is now selling at $90 and you want to cash out. Lisa doesn't want to get rid of her 100 shares; she's still bullish on $IBM. Thankfully, there are plenty of people out there buying or selling shares at any given time; Nate is willing to sell his 100 shares. You give those 100 shares back to Joe, fulfilling your obligation. But you only paid $9,000 for those shares, so you're left with a nice $1,000 profit in your account!



What Can Go Wrong?

What if the price didn't drop down to $90? If you bought them from Nate at $150 you'd be out $5,000. This is worse than the situation with regular trades. If your (owned) stock goes down in value after you buy and you decide to sell, you still have money, just less of it. In this case, you actually owe your brokerage money once you exit your short position.

Evaluating a Company

Investing in a company can be a gamble. To properly evaluate a stock and make an informed decision about whether it should be bought, held, or sold, we need to evaluate the company that issues it. There are a number of different factors worth taking into consideration that we'll cover in this chapter.

4.1 Metrics for Evaluation

Why does company's value matter in the first place? We want to make money by purchasing a company's stock when it's undervalued and sell it when it's overvalued, right? Generally-speaking, a company has a "true value" that increases monotonically over time; it's actual stock price, though, varies from that wildly. If we can identify deviations correctly, we can make good trades and steady profits.

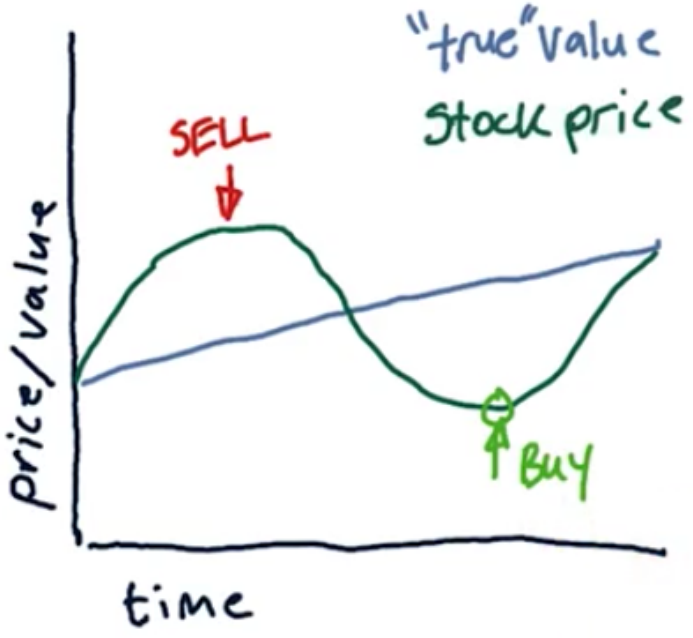


Figure 4.1: Determining a company's value helps you identify times at which a trade would be appropriate.

Suppose *Acme Corporation* can output $1 every year. How much would you say that company is worth? There are a few potential answers. Is it worth:

- $1,
- $70, because after that you'll be dead,
- $\$\infty$ because it will output $1 forever, or
- between $10 and $25 depending on interest rates?

Turns out all of these are valid choices depending on your valuation approach. The last one is probably the "most correct" given a traditional point of view of valuation.

A company is often evaluated on one or more of the following metrics:

- Intrinsic value: this is a measurement based on the **dividends** that a company puts out. A dividend is a cash payment to owners of that stock every quarter. If you owned one share of $AAPL, for example, it would pay out around $1 per year. Note that this payout is in addition to (or in spite of) changes to the share price itself. Intrisic value assigns a valuation to a company based on this payout.
- Book value: this is a valuation based on the assets that a company owns. Assets include things like factories, stock (in the "this item is out of stock" sense, not a financial sense), property, etc.
- Market cap: this assigns the value of the company to the value of a share times the number of shares there are, as previously defined.

4.1.1 Intrinsic Value

If someone told you, "Hey, I can offer you a guarantee that I'll give you a dollar a year from now." How much would you be willing to pay for that guarantee? Would you rather have:

- that guarantee from the U.S. government (this is called a **bond**),
- that guarantee from your professor, or
- $1 right now?

Obviously, the best thing to have is a dollar right now. There is no risk or need to wait. Furthermore, you probably trust the government more than your professor (well, maybe not...), so you'd take that guarantee over the other.

It should be obvious that the value of a dollar in the future is worth less than a dollar right now. There's a risk that that dollar will never be delivered. Furthermore, the value of a dollar often decreases over time due to inflation. How do we estimate the value of these promises to deliver money in the future? The calculation of this **intrinsic value** comes

down to the **interest rate**:

$$\text{present value} = \frac{\text{future value}}{(1 + \text{interest rate})^i}$$

Here, i is the amount of time[1] before you receive your future value. If the U.S. government offered you a 1% annual interest rate on your \$1, the present value comes out to:

$$\frac{1.00}{1 + 0.01} \approx \$0.99$$

In other words, it's worth it to pay $\leq$ \$0.99 today for a government bond of \$1 a year from now. What if your professor is offering you a 5% interest rate in order to entice you and offset the fact that he's less trustworthy? You should be willing to pay even less for that:

$$\frac{1.00}{1 + 0.05} \approx \$0.95$$

There is an obvious exponential decay as the amount of time you need to wait grows. The higher the interest rate, the lower you should be willing to pay and the faster the value drops.

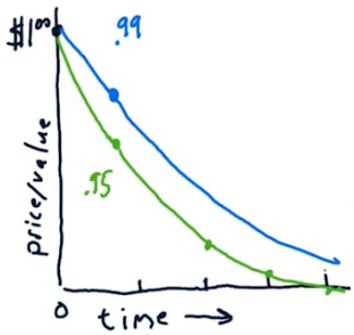


Figure 4.2: The exponential decay of the worth of future promises.

This problem is analogous to dividends in real companies. If \$AAPL pays out \$1/share in dividends every year, what is it worth paying for a single share? The "interest rate" in this case is now a metric of risk; we instead refer to it as a **discount rate**. Given that value, the value of the company to you would be the sum of the value of all of its dividends, where d is the discount rate. This infinite sum converges to a simple calculation:

$$\sum_{i=1}^{\infty} \frac{1}{(1+d)^i} = \frac{\text{future value}}{\text{discount rate}}$$

If we assume a \$1 annual dividend as above and a discount rate of 5%, the intrinsic value of that company would be $1/0.05 = \$20$.

4.1.2 Book Value

The **book value** of a company is defined by the total assets that company owns excluding its intangible assets and minus its liabilities.

[1] i operates on the same time scale as the interest rate. For example, if it's an annual interest rate, i is a measure of years.

Suppose a company owns four factories valued at $10M each, three patents at $5M each, and a $10M loan. The book value of this company would be:

$$(4 \times 10) - 10 = \$30\text{M}$$

Of course, this is a overly-complicated way to think about this. The intangible assets are just irrelevant to the calculation, so it's just tangible assets minus liabilities.

4.1.3 Market Capitalization

We can let the market determine the value of our company for us. We've already defined the concept of market capitalization, but for completeness sake here it is again:

$$\text{market capitalization} = \text{share price} \times \text{total shares}$$

4.1.4 Knowledge is Power

Why does information about a company like news reports, leaks, and new products affect its stock price? The way investors express their opinions on the value of a company is through its stock price (and consequently its market cap). If they believe the news devalues the company, they sell their shares and drive the price down. Let's look at how information can influence the other two metrics—intrinsic and book value.

If news comes out that a CEO is ill, the probability of investors getting the same dividends from that company as before decreases. This is "company" news and it directly lowers the intrinsic value.

If news comes out that the soil in Silicon Valley releases a chemical that increases brain capacity, the value of the assets of all companies in that area increases. This is "sector" news and directly increases the book value. We could similar have "market" news that affects all sectors.

Example 4.1: Stock Evaluation

Let's bring it all together and check our understanding with an example.

The company owns 10 airplanes valued at $10M each, its brand name is worth $10M, and it has a $20M loan. It pays out $1M in dividends annually across 1 million shares and has a 5% discount rate. Its stock price is $75.

What is the company's **book value**, **intrinsic value**, and **market capitalization**? Finally, **would you buy this stock?**

Give yourself a moment to work it out before you continue reading.

Its book value is \$80M: $(10 \times \$10) - \$20$.

Its intrinsic value is \$20M: $\$1\text{M}/0.05$.

Its market capitalization is \$75M: 1M shares $\times$ \$75.

You should **absolutely buy** this entire company right now. For \$75M, you can turn around and immediately sell all of its assets and pay off its loans for a cool \$5M profit. The "low" intrinsic value here is completely irrelevant, and this is why the market cap for a company *very* rarely drops below its book value.

4.2 Capital Asset Pricing Model

Developed independently by researchers in the 1960's, the **capital asset pricing model** (CAPM for short) explains how the market impacts individual stock prices and provides a mathematical framework for hedge fund investing.

Let's briefly recall how portfolios work. The total return of a portfolio is the sum of each asset's returns weighed by their ratio w_i:

$$r_p(t) = \sum_i w_i r_i(t) \tag{4.1}$$

Example 4.2: Portfolio Return

As a quick example, assume Stock A goes up $+1\%$ and Stock B goes down -2%. 75% of your portfolio is made up of \$A and you have a short position on \$B so its weight is -25%.

What is the return on your portfolio?
Applying our formula from (4.1) is trivial:

$$r_p(t) = (0.75 \cdot 0.01) + (-0.25 \cdot -0.02) = 1.25\%$$

This leads us to the concept of a **market porfolio**, which is effectively an index of a particular market. For example, the S&P 500 (tracked by \$SPY) represents the 500 largest companies that are trading on U.S. exchanges; it changes each day based on the value of those companies. There are similar indices in other countries like \$FTA in the United Kingdom or \$TOPIX in Japan. The market portfolio is a combination of these stocks, often weighed by their market cap (called **cap-weighted**):

$$w_i = \frac{\text{cap}_i}{\sum_j \text{cap}_j}$$

Each of these markets are broken into sectors like energy, technology, manufacturing, finance, etc... As we've discussed, news affects sectors and individual companies more often than it

affects an entire market. Some stocks have large weights and impact the market portfolio more significantly than others, for example, Apple ($AAPL) and Exxon ($XON) are each about 5% of the S&P 500.

This leads us to the **CAPM equation**:

$$r_i(t) = \underbrace{\beta_i r_m(t)}_{\text{market}} + \underbrace{\alpha_i(t)}_{\text{residual}} \tag{4.2}$$

You might recognize this as the equation of a line: $y = mx + b$; that's no coincidence. We've seen α and β before in Python for Finance when we discussed scatter plots: β was the slope and α was the y-intercept of a stock plotted against an index.

The CAPM equation states that the return for an individual stock i on a particular day t is the market return r_m scaled by some β plus some residual α. It asserts something important: a significant part of the return of a stock is affected *just* by the market at large. Most stocks have a $\beta \approx 1$: when the market goes up, the stock goes up approximately the same amount. The CAPM further asserts that the residual $\alpha_i(t)$ is essentially a random variable; as $t \to \infty$, the factor becomes irrelevant because it continually cancels itself out. In other words, its expectation is 0: $E_\alpha = 0$.

4.2.1 Portfolio Management Under CAPM

The inception of CAPM led to a debate about whether **passive** or **active** investment is better. Passive investing involves simply buying an index and holding it, letting the growth of the market bring in profits. Active investing is (obviously) more involved: you pick and choose the individual stocks (or change the weights of an index) in hopes you can beat the market.

Let's examine the CAPM equation under these two philosophies:

$$r_i(t) = \beta_i r_m(t) + \alpha_i(t)$$

Both camps agree that the β term is correct: the return is most significantly affected by the market. The contention comes from the α term: CAPM says its unpredictable and 0 on average. Active managers, though, claim they can predict α better than a coin flip.

4.2.2 CAPM for Portfolios

To compute the return for an entire portfolio under the CAPM, we just slap the weighted summation on (4.2):

$$r_p(t) = \sum_i w_i \beta_i r_m(t) + \alpha_i(t) \tag{4.3}$$

We can pull out the β term from the summation: $\beta_p = \sum_i w_i \beta_i$. Here's where the management strategies we just discussed differ, though. CAPM says $E_{\alpha_i} = 0$ so we can lump them all into a single term:

$$r_p(t) = \beta_p r_m(t) + \alpha_p(t)$$

Active managers, though, claim they can adjust the weights to outperform randomness, so the summation is still necessary:

$$r_p(t) = \beta_p r_m(t) + \sum_i w_i \alpha_i(t)$$

The CAPM implies some things about a robust portfolio. In a downward market, we want $\beta \leq 1$, so we don't lose as much money as the market in general. Contrarily, we want $\beta \geq 1$ in an upward market to outperform just tracking the market.

4.2.3 Arbitrage Pricing Theory

The **arbitrage pricing theory**, developed in 1976 by Stephen Ross, digs deeper into the CAPM equation and notes that the β_i for a stock i is not representative of the sectors its composed of. For example, RobinHood could swing fairly significantly with both the financial and technology sectors, but less so with manufacturing. To properly model that, we'd want to break up its β into $\beta_F r_F$ and $\beta_T r_T$:

$$r_i = \beta_{iF} r_F + \beta_{iT} r_T + \beta_{im} r_m + \ldots + \alpha_i$$

This would, in theory, give us a more accurate pricing model.

4.2.4 CAPM for Hedge Funds

By picking stocks with appropriate β values, the CAPM pricing model virtually guarantees a positive return. The key, of course, still lies in picking the stocks.

Suppose we're geniuses (or time travelers) and make perfect predictions about two stocks: Stock A has a $\beta = 1.0$ and will be $+1\%$ over market, and Stock B has a $\beta = 2.0$ and will be -1% below market. The behaviour of the stocks is plotted in Figure 4.3 below; if we enter our positions at the dashed line below—long \$50 on Stock A and short $-\$50$ on Stock B—how much money will we make?

Well, the market (magically) had no change during our investment period, so $\beta_A r_m = \beta_B r_m = 0$. Our return, then, is:

$$\begin{aligned} r &= r_A + r_B \\ &= \beta_A r_m + \alpha_A + \beta_B r_m + \alpha_B \\ &= \alpha_A + \alpha_B \\ &= (0.01 \cdot \$50) + (-0.01 \cdot 0\$50) \\ &= \$0.50 + \$0.50 = \$1 \end{aligned}$$

What if the market went up 10% during our investment time frame? In that case, the

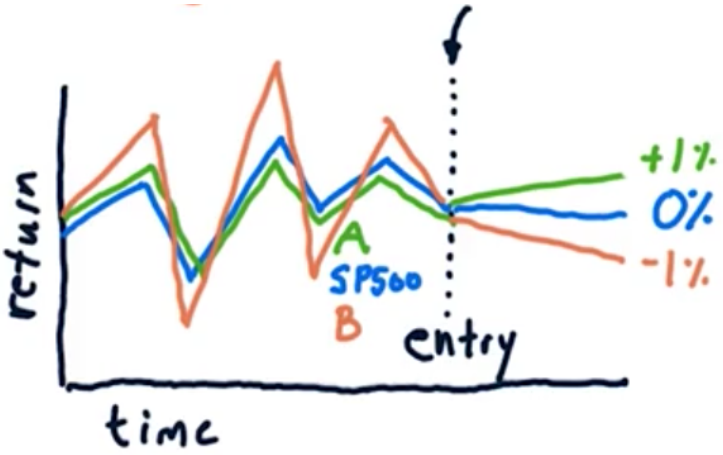


Figure 4.3

equation gets a little more complicated:

$$\begin{aligned} r &= \beta_A r_m + \alpha_A + \beta_B r_m + \alpha_B \\ &= (1 \cdot 0.10 + 0.01) \cdot \$50 + (2 \cdot 0.10 - 0.01) \cdot -\$50 \\ &= \$5.50 - \$9.50 = -\$4 \end{aligned}$$

Contrarily, what if the market went down 10%? Our short position on a $\beta = 2$ stock pulls us into the black:

$$\begin{aligned} r &= \beta_A r_m + \alpha_A + \beta_B r_m + \alpha_B \\ &= (1 \cdot -0.10 + 0.01) \cdot \$50 + (2 \cdot -0.10 - 0.01) \cdot -\$50 \\ &= -\$4.50 + \$10.50 = \$6 \end{aligned}$$

If we're not careful about how we allocate money, perfect α and β knowledge won't save us. Lets plug our assumptions into the CAPM and see how our stock choices affect our portfolio outcome, keeping the market return as our variable. From (4.3), we know that: $r_p = \sum_i w_i \beta_i r_m(t) + \alpha_i(t)$.

Plugging in our terms gives us:

$$\begin{aligned} r_p &= \sum_i w_i \beta_i r_m(t) + \alpha_i(t) \\ &= (w_A \beta_A + w_B \beta_B) r_m + w_A \alpha_A + w_B \alpha_B \\ &= (0.5 \cdot 1 - 0.5 \cdot 2) r_m \\ &= -0.5 \cdot r_m + 0.01 \end{aligned}$$

This corresponds to our calculations above. Though we have perfect predictions and information about our αs, we have no knowledge or control over the market returns, r_m. Is there

a way we can remove that component? In other words, can we make $\beta_p r_m = -0.5 r_m = 0$? Remember that

$$\beta_p = w_A \beta_A + w_B \beta_B$$

The question is, then: how can we choose weights for our portfolio such that they evaluate to zero? For what w_A, w_B is the following true?

$$w_A + 2w_B = 0$$

Don't forget we also have the constraints that $w_A > 0$, $w_B < 0$ (since we're shorting Stock B), and the absolute value of their sum is 1. We can arrange all of this information into a simple system of equations a high-schooler could solve:

$$\begin{cases} w_A = -2w_B \\ |w_A| + |w_B| = 1 \end{cases}$$

We can easily solve this system to get $w_B = -1/3, w_A = 2/3$. Plugging this back into the CAPM portfolio equation gives us:

$$\begin{aligned} r_p &= \underbrace{(w_A \beta_A + w_B \beta_B)}_{=0} r_m + w_A \alpha_A + w_B \alpha_B \\ &= w_A \alpha_A + w_B \alpha_B \\ &= 0.67 \cdot 1.0 + -0.33 \cdot -1.0 \\ &= 0.01 \end{aligned}$$

This means that regardless of the way the market moves, we will get a 1% return. Of course, this heavily relies on the fact that the αs we claimed to know are in fact true, and the βs accurately represent the stocks' reactions to market changes. The main purpose of this was to demonstrate that we can minimize the effects of the market on our portfolio with $\beta_p = 0$.

4.3 Technical Analysis

There are two broad categories of choosing stocks to buy or sell: **fundamental analysis** and **technical analysis**. In the first section of this chapter, we discussed Metrics for Evaluation that fall in the former category; fundamental investors analyze the aspects of a company that estimate its value and look for opportunities to trade when the stock price goes above or below that value. On the other hand, technical analysis disregards the true value of a company, instead looking for and leveraging trends in a stocks price.

Technical analysis looks at **only** at historical price and volume. It involves computing statistics called **indicators** that are heuristics that hint at a buy or sell opportunity. Since it doesn't consider the value of a company, it might be more correct to call this a *trading* strategy rather than an *investment* strategy.

Technical analysis is often effective in the short term: Buffet invests in the long-term after doing thorough due diligence about the true value of a company, while high-frequency traders use indicators to buy and sell stocks within seconds. Since there are so many traders looking at individual indicators, it's harder to leverage them as an individual. Consequently, well-crafted combinations of indicators can be much more unique and effective. Finding stocks that contrast heavily to the market can also be beneficial; if every stock follows the market, you simply can't do better than the market.

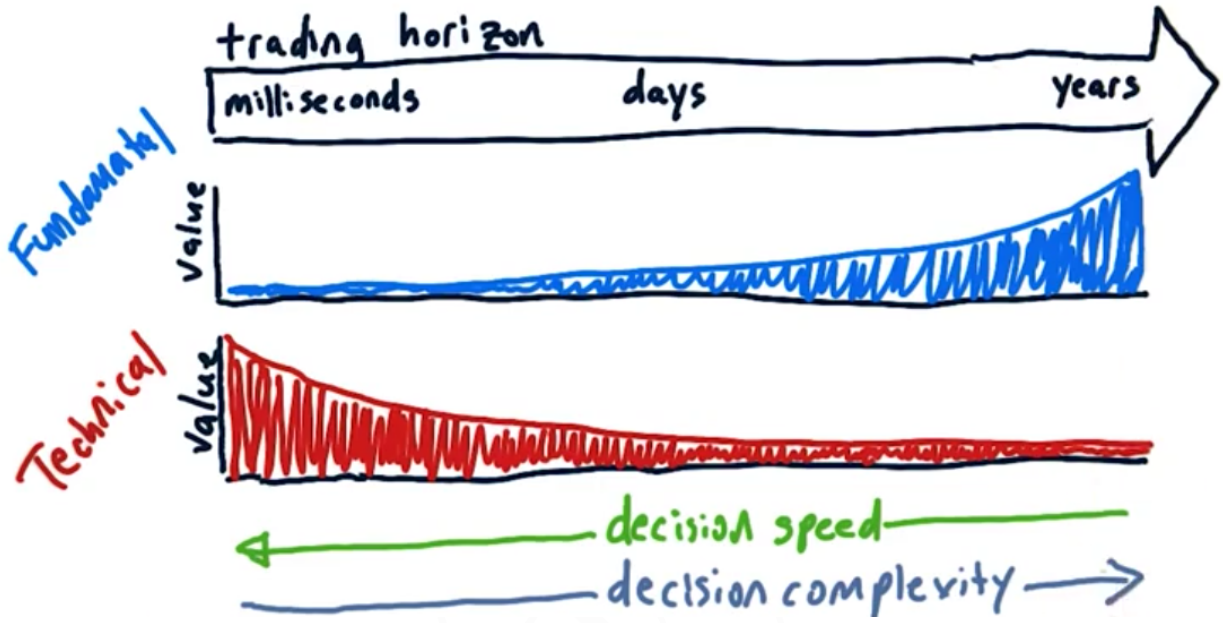


Figure 4.4: The contrast between fundamental and technical analysis. In the short term, technical analysis has a lot of value that quickly peters out; fundamental analysis, though, is most effective long-term as the true value of the company can shine. There are similar differences in decision speed vs. complexity, showing us how computers can shine in short-term trading, though they're less useful in long-term value-based investing.

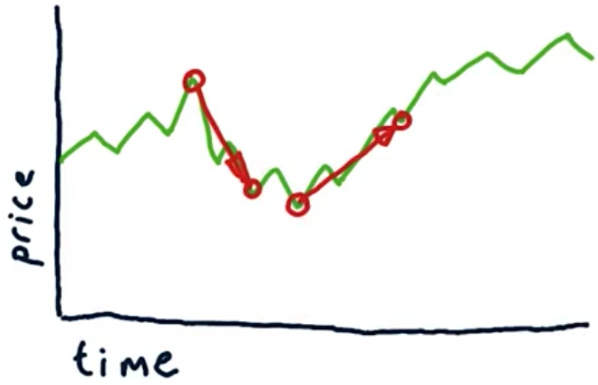
4.3.1 Indicators

There are thousands of technical indicators out there, but we'll cover a few solid, common ones you may see: **momentum**, simple moving average, and Bollinger bands.

Momentum

Momentum is one of the simplest indicators; it's the rate of change of a stock. Over a certain number of days, how much has the price changed?

The first red line in the price graph above has steep negative momentum, whereas the second red line has steady positive momentum. The common way some traders will use momentum as an individual indicator is by assuming that a stock's momentum will continue.



Snippet 4.1: Momentum is a simple calculation measured over a time window of n days. It's a percentage, so it's often $\in [-0.50, 0.50]$.

```
momentum[t] = (price[t] / price[t - n]) - 1
```

Simple Moving Average

Much like momentum, the **simple moving average** (or SMA) operates on an n-day window of time. The SMA is then just the average over those n days; it looks like a smoothed value of the price chart that lags behind.

The points of intersection between the simple moving average and the price graph tend to be important when using SMA as an indicator. Combined with momentum, this can indicate a trading opportunity. For example, in the above graph, the downward momentum and SMA intersection suggests that the stock may turn around its trend soon, indicating a buying opportunity.

Furthermore, gaps between a stock price and its moving average can be interpreted as a proxy for its true underlying value. The far-left downward excursion suggests a buying opportunity since the stock price has dipped significantly below its "true value" estimated by the SMA, and vice-versa for the two upward excursions on the right.

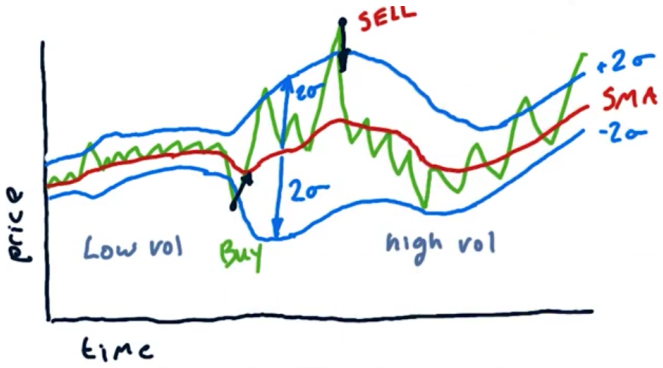
Snippet 4.2: The simple moving average is a ratio between the current price and the average price over a time window of n days. The SMA is a percentage, so it does not often deviate from being $\in [-0.50, 0.50]$.

```
sma[t] = (price[t] / price[t - n : t].mean()) - 1
```

Bollinger Bands®

We've seen Bollinger bands before: it's a $\pm 2\sigma$ deviation from the simple moving average. The bands are good thresholds for acting on price excursions, allowing short spikes and deviations in the stock price.

The reason why 2σ is a good value is because it naturally varies with the volatility of the stock. In periods of low volatility, we would want to act on larger spikes, whereas in periods of high volatility we would want to let those slide a little more.



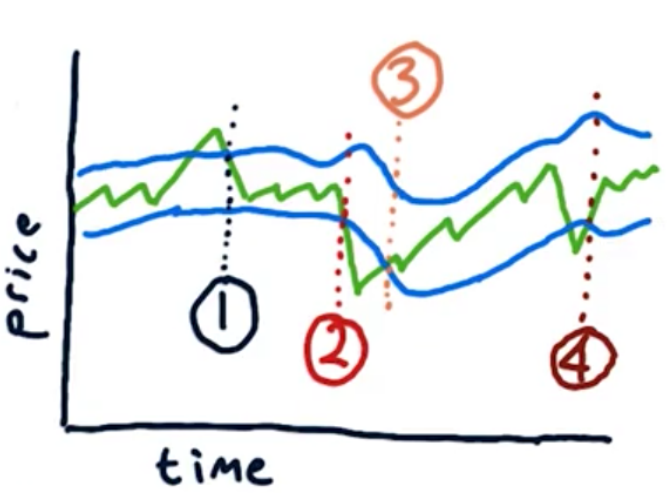
Points at which a stock crosses to the outside of its Bollinger bands *then re-enters* are often considered to be trading opportunities. For example, in the above graph when the stock spikes above the 2σ band and dips back in, that is a selling opportunity.

Snippet 4.3: Bollinger bands are $\mu \pm 2\sigma$. We typically expects to see BB values $\in [-1, 1]$; deviations from that often indicate trading opportunities.

```
bb[t] = (price[t] - sma[t]) / (2 * std[t])
```

Example 4.3: Bollinger Bands — Buy or Sell?

Given the following graph, do each of the circled points indicate a **buy** signal, a **sell** signal, or **neither**?



Answers

In ①, we see a spike outside of the upper band, then back in. That is a signal to **sell**.

In ②, we see a spike outside of the lower band, but no re-entering. This **isn't an indication** of anything and prevents us from a premature buy when the stock could be plummeting for a while.

In ③, we re-enter through the lower band, indicating a **buy** signal.

In ④, we spike below the lower band and re-enter, again signaling a **buying** opportunity like in ③.

Normalization

Notice that each of our indicators has a different range. This makes it difficult to plug them into plots or learning models, so we need to normalize them. This is straightforward and something we saw very early on in chapter 1.

Given an array of values, $\mathbf{v}$, the normalized array in the range $[-1, 1]$ is simply:

$$\hat{\mathbf{v}} = \frac{\mathbf{v} - \mu_{\mathbf{v}}}{\sigma_{\mathbf{v}}}$$

4.4 Anomalous Price Changes

Until this point, we've assumed that stock pricing data follows a sensible trend and that large leaps or discrepancies in price can be attributed to news or "market forces." That's not always true, though. Things like stock splits, dividends payouts, etc. can change the price but not indicate a change in company value or the result of a trade.

Before we get to that, though, we need to briefly discuss how stock data is aggregated.

4.4.1 Data Aggregation

The data we're working with—volume and price data—is discretized by a **tick**, or a single buy/sell transaction. For high-volume and highly-liquid stocks, there might be hundreds of thousands of ticks occurring every second over several exchanges, which is a massive amount of data. For ease-of-use, brokers often aggregate the tick data on a minute-by-minute or hour-by-hour basis, summarized by its **volume** as well as its **open**, **high**, **low**, and **close** values. These should all be fairly self-explanatory: over a given time period, open and close are the first and last transactions; high and low are the highest and lowest prices; and volume is the number of trades that occurred.

4.4.2 Stock Splits

If we look at this aggregated ticker data over a sufficient period of time, we may see areas with massive drops in value that are uncharacteristic for the company. Of course, they may be attributed to actual catastrophic events, but that's highly unlikely.

A **stock split** is a way for a company to lower its per-price share. This may occur to make the stock more accessible[2] (i.e. its price is too high), to make the shares more divisible, etc. If you own stock in company and it undergoes a split, you still come out with the same total value, just split across more shares.

In a price chart, this is dealt with my using an *adjusted* closing value. Going backwards in time, when encountering a split, the "actual" price is divided by the split factor to create the adjusted price. This adjusted price is easier to understand.

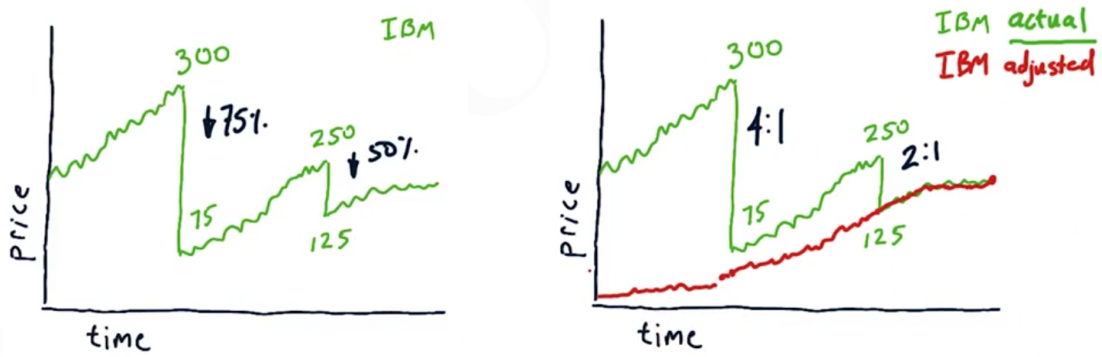


Figure 4.5: A fabricated stock chart for $IBM featuring a 4-way split and a 2-way split that (incorrectly) appears as a −75% and −50% drop in value when viewing raw price data (left). On the right is the adjusted price chart (in red) after accounting for the splits.

[2] Options trading—like shorting—often happens on multiples of 100 shares; given a high-enough stock price, this becomes relatively inaccessible.

4.4.3 Dividends

We discussed dividends previously when we talked about calculating the intrinsic value of a company. Dividends are issued periodically and they have an effect on the stock price.

Consider a company that pays out a $1 dividend whose "true worth" is $100 per share. What should its stock price look like the day before and the day of the dividend? Remember, that would mean we own the $1 dividend AND a share of the company that's worth $100.

What ends up happening is that as the payout day gets closer, the share price will grow towards $101, sharply dropping off back to $100 after the payout. This is because people know that they will get $101 worth of value on that day: the dividend and the share.

Just like with stock splits, we need to adjust the historical stock prices to take into account dividend payout behaviours.

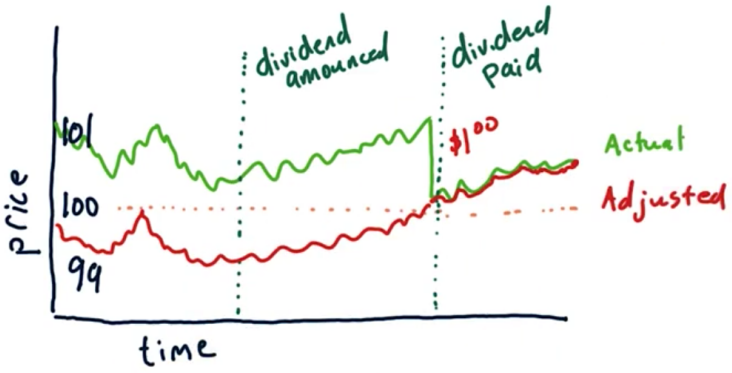


Figure 4.6: Dividend announcements affect share prices and the payout usually results in an uncharacteristic price drop. For an accurate evaluation of the historical price data, we need to adjust it based on the payouts by subtracting the dividend payout from the price.

Beating the Market

Is it possible to beat the market? If so, how do we do it?

5.1 The Efficient Markets Hypothesis

In the previous chapter we discussed the capital asset pricing model and how we could make money despite it. It hinged on some very important assumptions, though. We assumed that there was untapped information in historical pricing data that we could discover and leverage, giving us an edge over other traders; we also assumed that we could use new information about a company, like earnings data, to make well-informed long-term trades. The **efficient markets hypothesis** states that **neither of these are true**.

The EMH hinges on a couple of (fairly true) assumptions about the market. It assumes that the market has a large number of investors, and new information about the market arrives relatively randomly. This essentially guaranteed that prices will adjust quickly in response, and thus prices already reflect all available information.

The weakest of these assumptions, in my opinion, that may "prevent" a savvy investor from making money is that (a) information arrives randomly and (b) the market reacts quickly.

Even if (a) is the case, information will at some point *randomly* come to you early on enough to act. Where does information come from? Typically, traders look at the indicators involved in both fundamental (earnings, dividends, etc.) and technical (price, volume, etc.) analysis. Exogenous information as well as information from company insiders can also be incredibly impactful. Each of these "categories" of information has a relationship to the efficient markets hypothesis.

5.1.1 Three Forms

We will analyze each of the three forms of the EMH in turn.

Weak Form

The weak form of the EMH asserts that future prices cannot be predicted by analyzing historical prices. Because the current price already integrates all possible information from the past, such an analysis has no bearing due to the impact of future, unknown information.

This is called the "weak" form of the EMH because it still leaves room for fundamental analysis which evaluates the company as a whole rather than by searching for trading patterns.

Semi-Strong Form

The semi-strong form of the EMH asserts that prices adjust rapidly in response to new public information. This breaks down the fundamental analysis "loophole" in the weak form: when a company's quarterly reports become public, the market responds immediately, making it nearly impossible to act on these changes.

Strong Form

Of course, we can still rely on insider information, right? Wrong, according to the strong form of the efficient markets hypothesis. It (boldly) asserts that the price always reflects all public *and private* information. Secret information that might indicate a price increase will already be incorporated into the price.

5.1.2 EMH Validity

If the EMH in all of its forms is correct, this entire chapter (and arguably all of the chapters) is useless. We would be unable to ever make any money on the market aside from simply buying $SPY and holding it in the hopes that the U.S. economy doesn't go under.

Don't forget the most important word of the EMH: **hypothesis**. It makes a bold claim that's essentially impossible to prove, and situations absolutely arise that "violate" it. Thankfully, we've empirically demonstrated that certain forms of the EMH can be violated.

The "strong" version of the EMH is particularly suspicious: we've definitely seen and heard about insiders leveraging their internal knowledge about the company to make significant profit. Sometimes, they even go to jail for it.

Even the "semi-strong" version of the EMH is subject to scrunity. Figure 5.1 shows the 10-year price-to-earnings ratio (often abbreviated P/E) for stocks over a number of decades. We can see a clear trend regardless of the time period: companies with lower P/E ratios (meaning their quarterly earnings are not too-large of a multiple of their stock price) tend to have higher annualized returns over a 20-year period.

Because of this trend, we can argue that P/E ratios—which are a fundamental analysis indicator—can give meaningful insights about a stock, something that the semi-strong form of the EMH claims is impossible.

5.2 The Importance of Diversification

Why would you diversify instead of going all-in on $NVDA during the cryptocurrency boom? Miners are buying high-end graphics cards like candy; it's a no-brainer! Because when the crypto bubble pops, so does $NVDA's and you lose thousands of dollars. Thank you cryptocurrency for teaching me that lesson; my portfolio is still recovering.

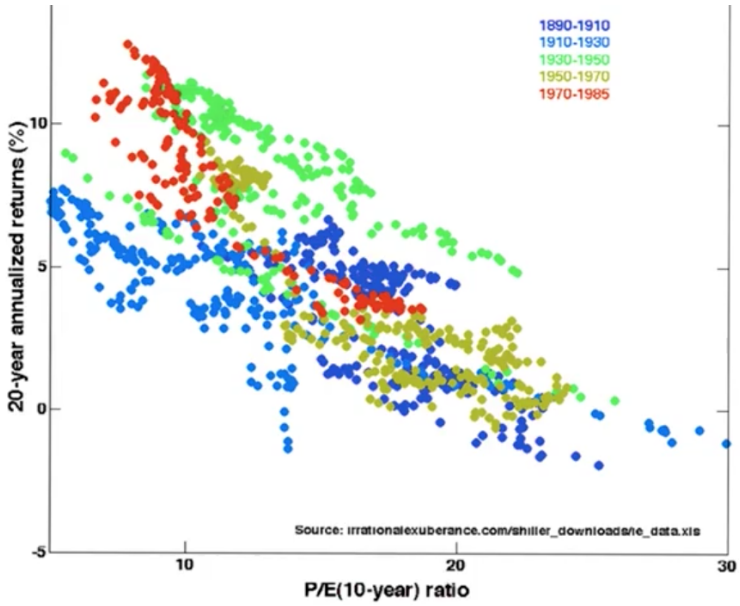


Figure 5.1: A plotting of the 10-year price-to-earnings ratio for a sample of companies from 1890–1985 versus their annualized returns. We can see that companies with lower P/Es tend to have higher returns.

Diversification is necessary, especially if you don't know what you're doing (e.g. most of us).

Richard Grinold was seeking to relate the following metrics as a guideline towards portfolio composition: *performance*, *skill*, and *breadth*. For example, you may be highly-skilled at picking stocks that do well, but you don't have enough breadth to find those stocks often. He came up with **Grinold's fundamental law** for active portfolio management:

$$\text{performance} = \text{skill} \cdot \sqrt{\text{breadth}}$$

This law implies that your portfolio's performance can improve by getting better, or by simply including more stocks, although this secondary growth method is far slower. This equation simplifies to an **information ratio**:

$$\text{IR} = \text{IC}\sqrt{\text{BR}}$$

Now that we've stated this "law", let's go through the intuition and rationale that makes it hold.

5.2.1 Coin Flipping Casino

We'll be exploring Grinold's law through a thought-experiment called the "coin-flipping casino." Instead of trading stocks, we'lll be flipped a biased coin. This bias is much like α; we want to do better than random chance. We'll say the coin has a 51% chance to come up heads. Similarly, the uncertainty of the outcome like β.

Betting works as follows: if you bet n coins and win, you have $2n$ coins. If you lose, you have zero and the game is over. The casino has 1000 tables, each with its own biased coin, and you have 1000 tokens (an indication of a bet) that you can distribute among the tables as you see fit.

How would you distribute your tokens? Is it better to go all-in on one coin flip, spread your tokens among all of the betting tables, or something in between? We need to consider both **risk** and **reward**.

The expected return of any bet is the chance of winning times the subsequent winnings plus the chance of losing times the resulting losses. If $\Pr[\text{win}] = 0.51$, then for a single bet:

$$0.51 \cdot 1000 + 0.49 \cdot -1000 = \$20$$

So we should expect to win (on average, over infinite trials) \$20 if we put 1000 coins down on one table. For multiple bets, the loss or gain is $\pm\$1$ but we have 1000 trials, so

$$1000 \cdot (0.51 \cdot \$1 + 0.49 \cdot -\$1) = \$20$$

The reward is the same for both scenarios! How can we prove, then, that the spread-out bet is better? We need to calculate risk, and there are a couple of ways to demonstrate it.

Losing It All What's the chance that we lose all of our money? For a single bet, we have a 49% chance we'll lose everything. For multiple bets, we have 1000 flips and each one has a 49% chance of being lost, but to lose *all* of them, it'd be $(0.49)^{1000}$ which is *astronomically* unlikely.

Standard Deviation Another way to analyze risk is by looking at the standard deviation of the individual bets. Assume we bet one token per table and we're now looking at the result after-the-fact. For example, we might lose one, gain one, lose one, lose another one, and so on, for our 1000 trials:

$$\underbrace{-1, 1, -1, -1, 1, 1, \ldots, 1}_{\text{1000 trials}}$$

The standard deviation is just 1; any given trial is either a win or a loss.[1]

What about for one 1000 token bet and 999 zero token bets? The standard deviation is a lot different. On the first table, we either win or lose 1000, and the rest of the tables are 0:

$$\begin{cases} +1000, 0, 0, \ldots, 0 & \text{if we win} \\ -1000, 0, 0, \ldots, 0 & \text{if we lose} \end{cases}$$

The standard deviation works out to be 31.62 for both of these cases, which is a far larger variance than in the previous case. Again, this signifies how much riskier the latter approach is.

[1] I wish he went into this more as my probability skills are trash and this explanation is not satisfying.

Clearly, 1 token on each of the 1000 tables is the way to go because the other extreme is *incredibly* risky. As we saw, expected return for both is actually exactly the same; it's just that one has a much lower risk.

If we calculate the risk-adjusted return (also known as the Sharpe ratio) for our two scenarios, we get:

$$\begin{aligned} \mathrm{SR}_{\text{single}} &= 0.63 \\ \mathrm{SR}_{\text{multiple}} &= 20 \end{aligned}$$

Fascinatingly, if we multiply our very safe single bet by our "breadth" (i.e. spread amongst 1000 tables) as in Grinold's Law, we get the same SR as the other case:

$$\mathrm{SR}_{\text{single}} = 0.63 \cdot \sqrt{1000} = 20 = \mathrm{SR}_{\text{multiple}}$$

PART III

Learning Algorithms for Trading

The third part of this topic will cover applying machine learning algorithms to financial data in order to create models that give us insights about stock behavior. We can use these insights to (try to) predict future price data and make trading decisions based on those predictions. Financial models have existed for a long time, but using machine learning lets us create "data-centric models" in which the evidence speaks for itself; there is no longer a need to subjectively interpret the data.

Let's talk about machine learning in broad terms at first, then get into specific techniques and algorithms. What problem does machine learning solve? Well, given an observation input X (like the last year's daily returns of a stock, for example), we can feed it into a **model** and get some output prediction Y (like the *next* set of daily returns). The model, of course, is created from massive amounts of data fed into a machine learning algorithm.

$$X \longrightarrow \boxed{\text{model}} \longrightarrow Y$$

Supervised Regression Learning

The first part of this chapter will focus on **supervised regression learning**: **supervised** by a sequence of example predictions $(\mathbf{x_i}, y_i)$, we will **learn** a numerical prediction (the **regression**, a poor naming choice). Some examples that fall under this category of machine learning algorithms that we'll be discussing soon include linear regression, k-nearest neighbor (or kNN), decision trees, and decision forests.

In terms of stock data, the most pertinent "prediction" is a future price. This allows us to make decisions on top of that price data rather than restricting our model to making strategy-specific decisions (like "Buy!") directly.

Remember, we can have many feature vectors $\mathbf{x}_i$ for a particular stock. Typically, we have a `pandas DataFrame` for each feature in which each column is the data for a particular stock and each row is the data at a specific point in time. When we extend this to many features, we can think of it as a third dimension to our data in which the "depth" is the feature number. What is our $\mathbf{y}$, then? Well, as we've already established, it will be price. Specifically, we can create an association between a particular feature and a "future" price. We say "future" because we actually have historical price data. So we could look at the Sharpe ratio for \$IBM on January 1st (this is our x_i) and correlate it with its price on January 5th (this is our y_i).

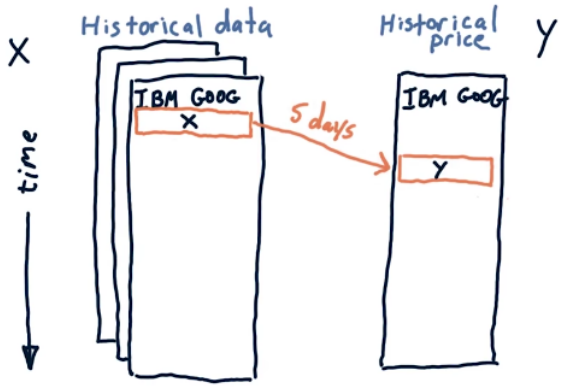


Figure 6.1: Our machine learning model associates a 3-dimensional array of feature values for a variety of stocks with their (historically) "future" price data.

The "art" of model creation is picking these predictive factors—the more accurate your resulting predictions, the more someone is probably willing to pay to use your model.

Backtesting How accurate can these models *really* be? The stock market is a fickle beast and some argue it runs more on emotion than reason. We need a way to test the effectiveness of our model without losing tons of money in the process. This is where **backtesting** comes in: we apply our model on a different period of time than the one it was trained on and see how accurately its predictions match the actual stock behavior.

This is somewhat akin to splitting the training data into a "training" and "test" set that we'll dive into further soon. Given a model based on a set of training data, we can place a set of orders for a future point in time. We then use the historical data to see how those orders would've performed, plotting the performance of our fake portfolio over time (in terms of any of the statistics we learned about in Part I). We can repeat this for as long as we have historical data for, seeing how our model would've "really" performed over that period of time.

Figure 6.2: Describing the process of backtesting our model over historical price data, plotting the results over time.

With a cursory overview out of the way, let's dive into our first set of machine learning methods that will make us teh big bux.

6.1 Regression

Don't let the spooky term "regression" scare you: it's just a numerical model. It outputs a number (its prediction) based on a bunch of input numbers.

6.1.1 Linear Regression

Let's take a look at a *parametric* method first. We've seen the concept of a parametric model before: in Building Parameterized Models, we chose parameters to our line (m, the slope, and b, the y-intercept from the familiar equation of a line, $y = mx + b$) that best fit our data set. We defined "best fit" as minimizing the total vertical error between the line and the points. This is called **linear regression**.

Once we've found our best-fitting model parameters, calculating predictions is just a matter of plugging our new input values into the model. We can throw the initial data away. The problem, as you can see in Figure 6.3, is that the real world seldom follows a linear model. Furthermore, it's hard to say what degree of a polynomial you'd need to best fit a model. It might be better to let the data speak for itself, which would lead us to...

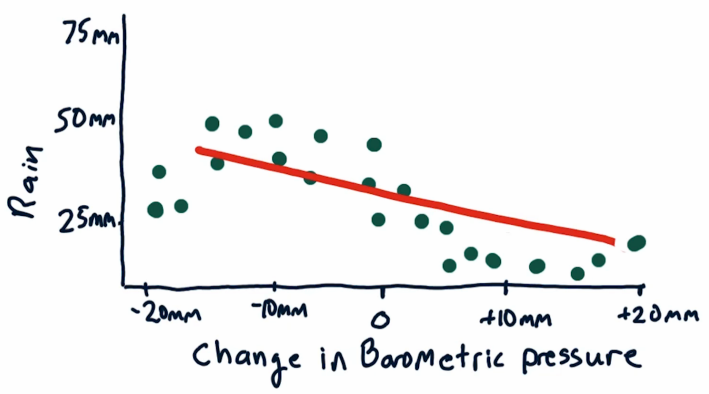


Figure 6.3: Using linear regression to fit a line to a data set mapping the correlation between changes in barometric pressure and rainfall. Notice that a higher-dimensional polynomial might have fit this data better, but this is the best we could do with our model (a line).

6.1.2 k-Nearest Neighbor

This data-centric approach (called an *instance* method, in contrast with the *parametric* method earlier) uses instances of the data points to create an approximate prediction based on the distance of a novel input to 1, 2, or k of its nearest neighbors in the data. In **k-nearest neighbor**, we just take the mean of the y values. This method actually works surprisingly well for $k > 1$, but isn't always ideal because it requires storing the data which might have many dimensions and take up a lot of space.

If we actually take the *weighted* mean of the y values based on the distance of the neighbors to the new input, that's actually called **kernel regression**.

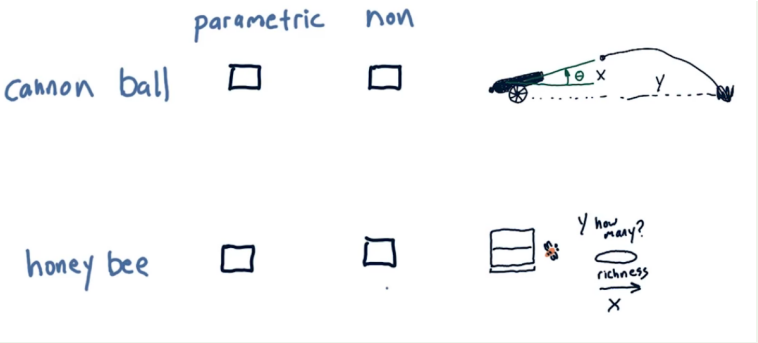
EXAMPLE 6.1: **Choosing an Approach**

How should we decide when a parametric approach like linear regression should be preferred to a situation over an instance or data-centric approach like kNN?

Suppose we have two situations we'd like to have a model for:

- A projectile is fired from a cannonball, and we'd like to predict where it lands. Our observable is θ, the angle of the cannon, and its landing point.
- A hive of honey bees is attracted to a food source; we'd like to predict how many bees will go to a particular food source given its richness. Our observable data is n, the number of bees, and a food "richness" metric.

To which of these should we apply a parametric model? A non-parametric model?

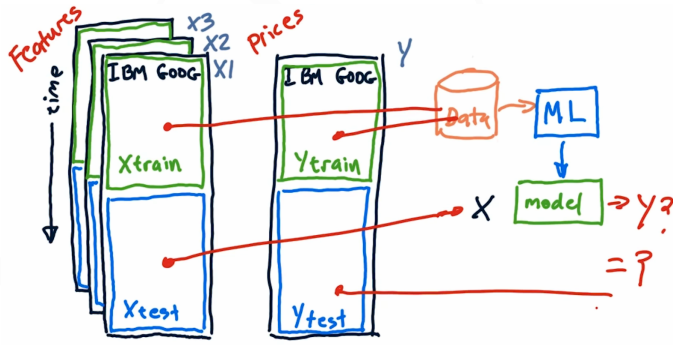


ANSWER: The cannon would probably use a parametric model, whereas the bees would be better modeled in an instance-based way.

Generally speaking, the more well-defined a problem is, the more applicable a mathematical model might be, meaning a parametric approach is more suitable. We don't really have much "mathematical bee theory," so crafting a model based on the things we see is much more applicable.

6.1.3 Training vs. Testing

We have a limited amount of historical stock data to work with, right? But we need to evaluate our model without risking real money. Thus, we need to split our data into **training data**—which is fed into the algorithm to create the model—and **testing data**—which we use to evaluate the quality of our model before taking out a 2nd mortgage on our house.



An important caveat of train-test splitting in historical price data that doesn't necessarily apply to other applications of machine learning algorithms is that we want training data to always be earlier in time than testing data.

6.1.4 Problems with Regression

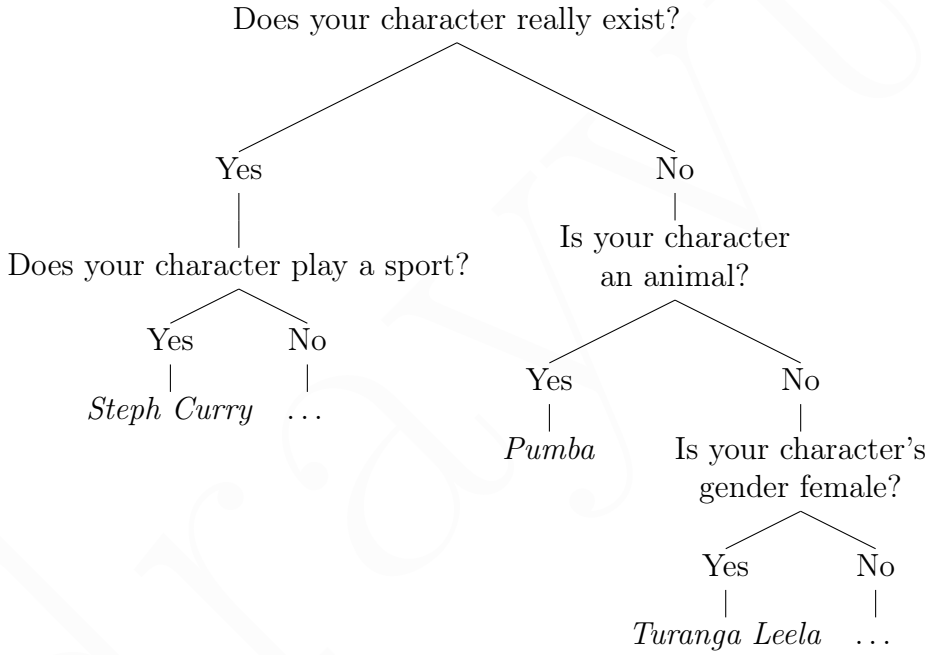
Obviously if this was a perfect solution we could call it here and become millionaires. Unfortunately, reality is often disappointing. Regression is a noisy and uncertain method. Furthermore, it's challenging to estimate confidence in a model and reflect that to the user in an understandable way (just showing the standard deviation is not very reliable nor user-friendly). There are also challenges nested in the problem that are specific to stocks: how

long do you hold for?[1] How do you optimize allocation ratios?

6.2 Decision Trees

We'll open our discussion of machine learning algorithms with **decision trees**.[2] What is a decision tree? It's fairly self-explanatory: it's a tree that maps various choices to diverging paths that end with some decision. For example, one can imagine the "intelligence" behind the famous "character guessing" AI Akinator: for each yes-or-no question it asks, there are branches the answers that lead down a tree of further questions until it can make a confident guess.

One could imagine the following (incredibly oversimplified) tree in Akinator's "brain:"

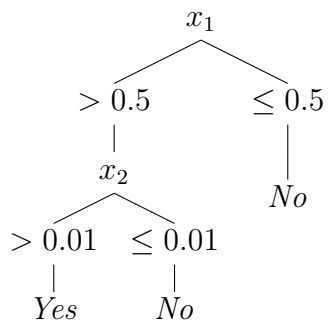


Except Akinator has played (and thus improved upon and expanded its decision tree) about 400 ***million*** games as of this writing.

In most applications of decision trees—especially with respect to financial algorithms—the branches are based on *numeric* boundaries in a particular **feature**. For example, we might decide to invest (or not invest) in a particular stock if its Sharpe ratio is above 0.5. Or, to branch further, we might only invest if $S > 0.5$ *and* its daily returns are above 1%. We'd say that $\mathbf{x}_1$ is the "feature vector" for Sharpe ratio (one element for each stock we're choosing from) and $\mathbf{x}_2$ is the feature vector for its daily returns:

[1] Remember, you pay lower taxes on your capital gains (i.e. profits from stocks) if you hold them for over a year! This kind of calculation is important to keep in mind when analyzing a strategy or model.

[2] This section is based in-part on the founding paper by J.R. Quinlan on decision trees.



Note that a feature can (and likely will) appear more than once in a decision tree. Similarly, some features may be completely disregarded when making a decision for a particular input value depending on the path it takes. Consider the table:

	x_1	x_2
\$JPM	1	0.10
\$AAPL	0.2	0.07
\$SPY	0.7	-0.04

Which stocks would we invest in based on our decision tree above? Probably just \$JPM.

Coming to a conclusion given a decision tree seems pretty straightforward. The hard part is *creating* the tree. If we're *given* a set of data like the table above with an additional column describing what our choice *should* be, how do we find the best tree? How do we choose the feature to split our tree along? What values do we split a particular feature along? Etc.

6.2.1 Representing a Decision Tree

Before we dive into this topic, let's discuss the data structure itself. There are two main approaches: an object-oriented approach that many of you are probably familiar with in which there are `Node` objects that point to each other, and a matrix-based approach in which each row represents a node.

The matrix representation is more compact and more efficient to process, so let's go over it first. Let's continue with our running example of the buy/don't-buy decision tree and the feature vectors in the table above. Each row represents a node in the tree; the columns are the factor, the splitting value, and the left and right node that follow. For "special" (i.e. leaf) nodes, the splitting value is actually the decision value, and the factor is -1 to indicate that it's a leaf. For the above, it'd look something like:

Index	**Factor**	**Split Value**	**Left**	**Right**
0	1	0.5	1	4
1	2	0.01	2	2
2	-1	Yes		
3	-1	No		
4	-1	No		

For a given input, say $\mathbf{x} = \begin{bmatrix}1, 0.10\end{bmatrix}$ (corresponding to \$JPM), we'd first check the root

(i.e. index = 0) and see that it corresponds to the left value since we "pass" the split value[3]. The next node is at index = 1; the input likewise "passes" and we move on to index = 2, which is a leaf node that says "Yes, buy!"

6.2.2 Learning a Decision Tree

Let's talk about "learning" a decision tree now.

TODO: Finish this section based on OH videos.

```
def build_tree(data):

  if data.shape[0] == 1:
    return [leaf, data.y, NA, NA]

  if all data.ysame:
    return [leaf, data.y, NA, NA]

  determine best feature i to split
  onSplitVal= data[:,i].median()

  lefttree = build_tree(data[data[:,i] <= SplitVal])
  righttree = build_tree(data[data[:,i] > SplitVal])
  root = [i, SplitVal, 1, lefttree.shape[0] + 1]

  return append(root, lefttree, righttree)
```

6.3 Evaluating a Learning Algorithm

We need a way to measure the effectiveness of machine learning algorithms besides relying on training vs. testing data.

We can compare performance of various versions of model using a number of metrics we'll discuss shortly by modifying its configurable parameters. For example, in the k-nearest neighbor modeling method, our parameter is k, the number of neighbors to average. Variations in k are demonstrated in Figure 6.4: as k decreases, the model gets closer and closer to overfitting the data set.

What about a polynomial fitting model (such as numpy.polyfit), in which we can vary d, the dimensionality of the polynomial?[4] We can see the effect of varying d in Figure 6.5: as d increases, the model fits the given training data better and better. We need to likewise be careful not to overfit. Notice, as well, that with a polynomial or parametric model, we can see the model's predictions beyond the domain of the training data: we can extrapolate and still predict (though likely unaccurately) at the tails.

[3] We arbitrarily decide that ($x_i >$ split) corresponds to the left index; this is just a matter of convention.
[4] For example, for $d = 3$ we would have a model in the form $y = m_1x + m_2x^2 + m_3x^3 + b$.

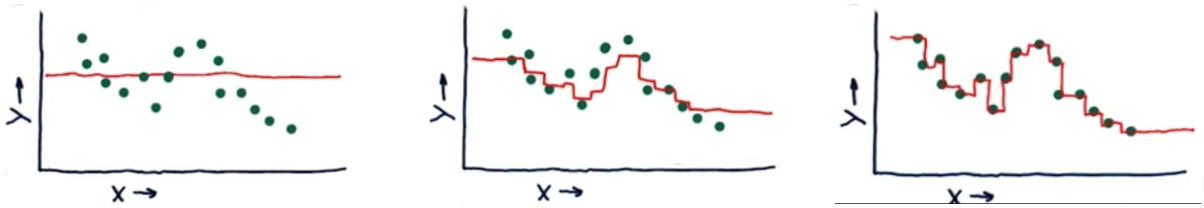


Figure 6.4: As k varies, so does the resulting kNN model fit. With $k = N$ (left), where N is the total number of data points, we can see that the model is just a flat line that is the average of the data's y values. With $k = 3$ (middle), we can see a tighter relationship to each point with sharp drop-offs as points go in and out of the kNN averaging "bubble." With $k = 1$ (right), we see intense overfitting with the model corresponding directly to the data.

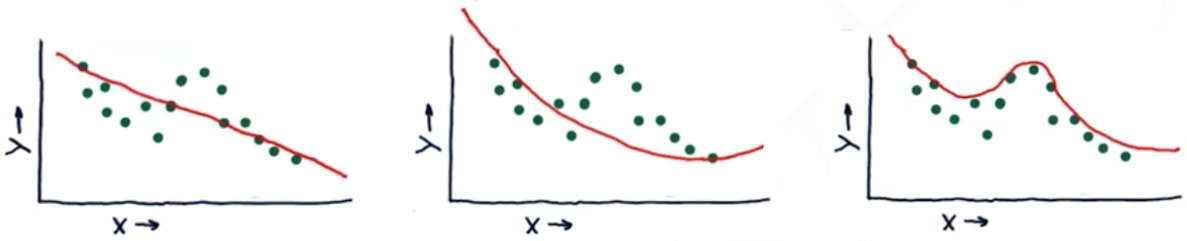


Figure 6.5: As d varies, so does the polynomial model's fit. With $d = 1$ (a line, left), we have basic linear regression; the model fits most points equally poorly and some points well. With $d = 2$ (middle), we can see more points closer to the model, but there are still a significant amount of outliers. With $d = 3$ (right), we see a closer fit to the model's shape given the training data, however it may be overfitting to the data we have.

6.3.1 Metrics

With an understanding of how models behave under certain parameter changes, we can now compare them against each other with error metrics.

RMS Error

Our first method is **root-mean-squared error**, or RMS error. This is a metric of the average "vertical" error of the data points to the model. At every data point (x, y), we can evaluate what the model f *would've* predicted, giving us $(x, f(x))$; the error of the model at that point is $f(x) - x$. The RMS is then:

$$E = \sqrt{\frac{\sum (Y_{\text{test}} - Y_{\text{predict}})^2}{N}} \tag{6.1}$$

Correlation

Another way to visualize and evaluate the accuracy of a regression algorithm is to look at the relationship between predicted and actual values on our dependent variable, y.

We fed our X_{test} data into our model and got a series of predictions, y_{predict}. A good model would have a tight **correlation** to the true y values (easily measured by numpy.corrcoef

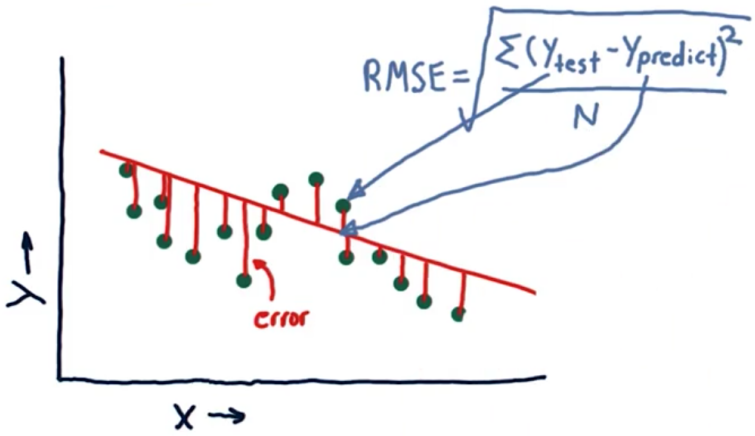
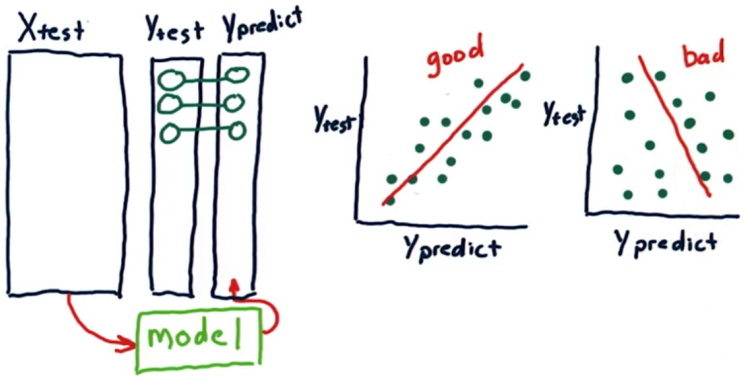


Figure 6.6: A visual explanation of the RMS error metric.

): correlation value near +1 indicates a strong positive correlation, −1 a strong *negative* correlation, and near 0 a weak correlation.[5]



6.3.2 Overfitting

We've mentioned overfitting several times already, but haven't explicitly defined what it means. Now that we have metrics for error, we can. **Overfitting** occurs when the error of our training data (or "in-sample error") decreases while *at the same time* the error of our testing data (or "out-of-sample error") *increases*.

[5] Note that the correlation isn't a measure of the slope of the line plotted between y_{predict} and y_{test}, but rather a measure of the "tightness" of the oval that bounds the points. More specifically, it can be thought of intuitively as a measure of the major and minor axes on the data's bounding ellipse, a concept tightly tied to principal components and the covariance matrix. This, unfortunately, disagrees with ya boi Khan in his video on the correlation coefficient, but I think a graduate school professor might be more trustworthy here?

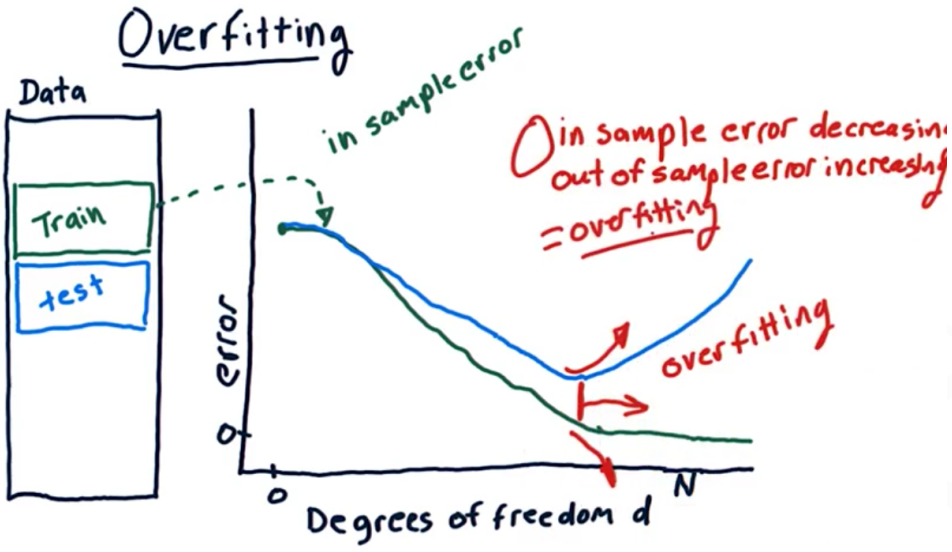


Figure 6.7: A visual explanation of overfitting: in-sample error decreases while out-of-sample error increases.

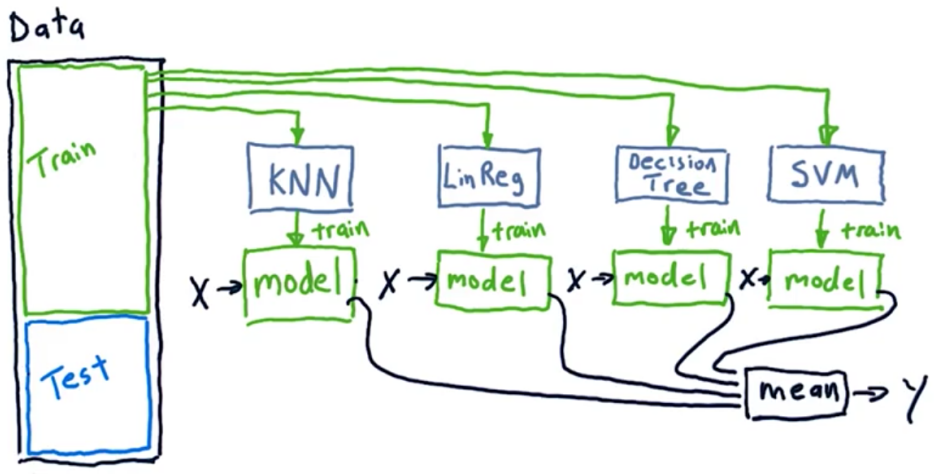
6.3.3 Cross-Validation

We've already discussed splitting our data into training and testing sets: the model is trained on the training data and evaluated on the testing data. We can use **cross validation** to generate "more" data sets from our single data set. For example, we could split our data into 5 equal parts. In one trial, we train on the first four parts and test on the last part; in another trial, we might train on the first two and last two parts and test on the third part. In this way, we can create many more trials and evaluations of our model from a single data set.

Unfortunately, cross validation in this simple form doesn't fit financial data very well. By allowing a training set occur after (as in, later in time) the testing set, we let our model "peek into the future," creating a far better prediction for the past than should be possible. Thus, we must specify a constraint of only allowing test data to occur later in time than training data; this is **roll forward cross validation**. We can still create multiple trials by "rolling" our data forward: we could use the first 10% of the data as training and 10% as testing, then that same testing 10% as training and the following 10% as testing, and so on.

6.4 Ensemble Learners

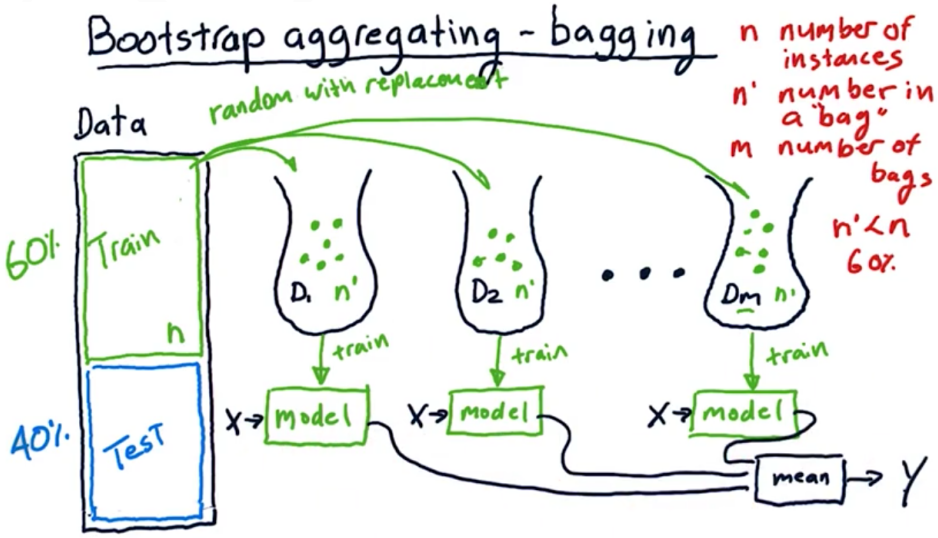
It's possible to combine a group of weaker learners together into an ensemble, ultimately creating a stronger learning algorithm that outperforms each of its parts. We can then take the mean or a carefully calibrated linear combination of the ensemble's predictions into our final prediction.



Ensemble learners often have lower error and less overfitting problems than an individual learner on its own. A single learner typically has some sort of bias—linear regression, for example, is biased towards making linear predictions—but a combination of biases effectively can cancel each other out.

6.4.1 Bootstrap Aggregating

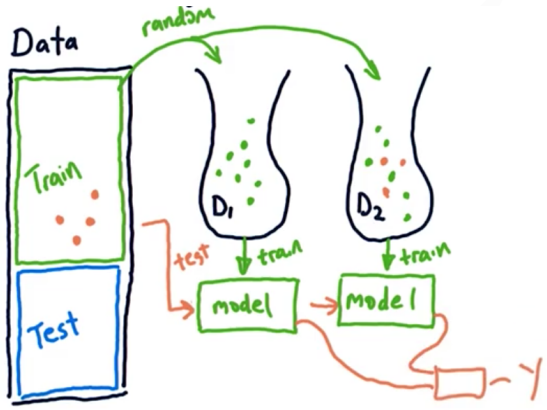
Instead of combining different types of learners into an ensemble, we can also have multiple learners of the same type that were trained on different subsets of our data set. This technique is called **b**ootstrap **agg**regat**ing**, or **bagging**. For each of our m models, we choose n' samples from our n-sized training data set (*with replacement*), where typically $n' \leq 0.6n$. Each of these "bags" of data is fed into a model, then the resulting prediction is the average of the m predictions.



6.4.2 Boosting

Not all of the learners in our ensemble are created equal—some will perform worse than others. Ideally, we would minimize the effect of weaker learners on the overall prediction. Furthermore, we can determine which parts of our training data is not predicted correctly by a learner and try to make the next learner fit that data better. This is the rationale behind **boosting**.[6]

We train our first model exactly like in bagging: take a random subset of the training data and create a model. Then, we test the training data on the model and identify which samples are predicted incorrectly. We weigh these samples higher in the training data set: the next model, when sampling, is more likely to train on the higher-weighted samples. We repeat this weight adjustment every time for the incorrect samples (restoring the correct ones to their normalized weight) and similarly take the mean for our final prediction on new data.



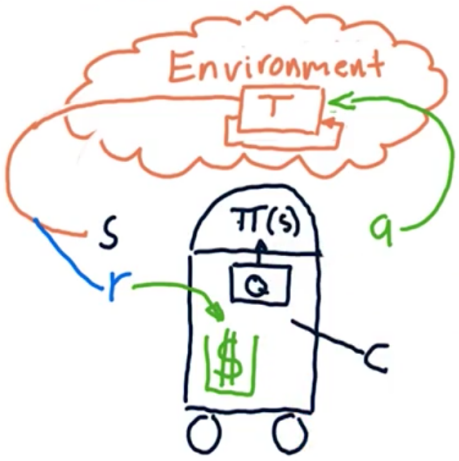
Note that as m, our number of models in the ensemble, increases, boosting methods typically overfit more than bagging methods. This is because boosting methods increasingly tries harder and harder to fit data that has been predicted incorrectly, and thus is creating a model catered more towards the training set than what is representative of the overall set.

[6] You can read about the specifics of a popular boosting algorithm: AdaBoost (**ada**ptive **boost**ing) on Wikipedia or in my notes on *Computer Vision* which also touches on this topic.

Reinforcement Learning

Our strategies and algorithms have been catered towards forecasting price data based on historical data. However, that's not enough, right? We need to be able to make informed decisions based on that price data, ideally automatically. That is the goal of *reinforcement* learning and what this chapter is all about.

Reinforcement learning algorithms come up with policies that specify which actions to take to reach a particular goal. They're broken down into three main parts: sensing, thinking, and acting. RL is often discussed in the context of robotics but it parallels to many other problems. The environment can be described by a particular state, S. An action by the agent, a, is fed into a transition function, T, which describes how the environment responds to the action; this cycle continues until the end of time.



An agent has a policy, $\Pi(S)$, which describes its decision-making process of converting environment statement into an action. This policy (and resulting action) is often computed with the goal of maximizing a reward, r. The reinforcement learning algorithm learns which actions maximize the reward and uses that to enforce a policy.

We need to model trading as a reinforcement problem. Much of the things we've discussed thus far can be associated with a state, action, or reward in the RL system. For example, buying and selling are clearly actions; Bollinger bands and trading strategies (like holding long or selling short) are state; and returns from trades are rewards. Some thing can be a little more ambiguous, though: daily returns can be considered as both state and the reward, depending on the trading approach. We will learn our policy by analyzing how the actions we take (buying and selling) affect our portfolio value and various returns.

The system we've been discussing is well-known as a **Markov decision problem**, or MDP. An MDP can be defined as a system with:

- a set of **states**, S;

- a set of **actions**, A;
- a **transition function**, $T[s, a, s']$; and
- a **reward function**, $R[s, a]$.

The transition function is a probability distribution: the sum of all possible "next state"s must be 1. We want to find a **policy**, $\Pi(s)$, that will maximize our reward. We call this the *optimal* policy, denoted $\Pi^*(s)$. There are two algorithms that can help us find this optimal policy: **policy iteration** and **value iteration**. Unfortunately, though, we won't know T or R in advance and so these algorithms can't help us directly.

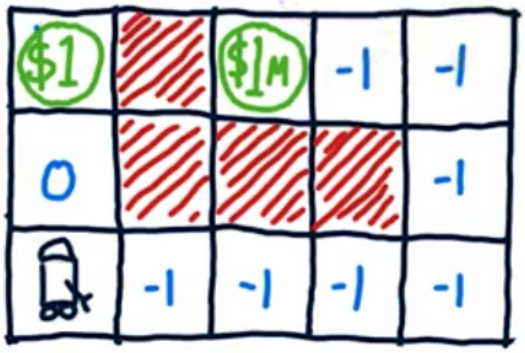
In a sense, our agent needs to learn about the available transitions as well as what a good reward is before it can tackle doing that optimally. We can think about our agent's "life" as a series of *experience tuples* which are much like the individual samples of training data we received during supervised regression learning:

$$\begin{aligned}&\langle s_1, a_1, s'_1, r_1\rangle\\&\langle s_2, a_2, s'_2, r_2\rangle\\&\langle s_3, \ldots\rangle\\&\vdots\end{aligned}$$

We are saying that we took action a_1 while in the state s_1; that led us to the new state s'_1 and we received the reward r_1. We call that new state s_2, now, and repeat the process which led to s'_2 (a.k.a. s_3), and so on...

Given enough of this data, we can actually construct a probability distribution which is an approximation of the world based on what we've observed, and then treat that as our T and R for policy/value iteration. This is called **model-based learning**. In contrast, **model-free learning** methods develop policies directly by looking at the data rather than by converting them to what essentially amounts to a tabular lookup. We'll specifically be looking at Q-learning, a well-known model-free learning algorithm.

Our concept of "maximizing the reward" is fairly vague; we can define it more rigorously. Suppose we have a robot that can navigate a maze, where each step has a particular cost (i.e. a negative reward):



The $1M cell is one-time use, whereas the $1 cell can be hit over and over for infinite returns. What would be the optimal strategy? To a human, it's obvious: get the $1M then get the $1 over and over. The cost incurred by reaching the $1M cell ($-14$ to get there and back) is insignificant.

This strategy hinges on a pretty important assumption, though: we can take (at least) 15 steps. More accurately, we probably assumed that we could take infinite steps; this is called an **infinite horizon**: $\sum_{i=1}^{\infty} r_i$. If we wanted to optimize our reward over 3 steps (a **finite horizon**), things would be much different: $\sum_{i=1}^{3} r_i$.

Much like when we discussed intrinsic value and discount rate, a dollar now is worth more than a dollar later; likewise, a reward sooner is better than a reward later and we should factor that into our optimization target. We typically call this factor $\gamma \in (0, 1.0]$ and it causes our reward to decay over time:

$$\sum_{i=1}^{\infty} \gamma^{i-1} r_i$$

This has the convenience of converging cleanly as well as being more representative of the real-world; γ is like an interest rate on the reward that lets the learning algorithm prefer instant gratification over "potential" gains in the future.

7.1 Q-Learning

Recall that **Q-learning** is a model-free learning method that will glean information about the real-world as it interacts with it, iteratively deducing the properties of T (the transition function) and R (the reward function). The beauty of Q-learning is that it's *guaranteed* to provide an optimal policy.

The name "Q-learning" comes from the fact that the method relies on a function $Q[s, a]$ which represents as a 2D table for *s*tates and *a*ctions that grows as our agent interacts with the world. The value in the table is the **total reward** for taking action a while in the state s which is composed of the *immediate reward* and the *total discounted reward* for future actions.

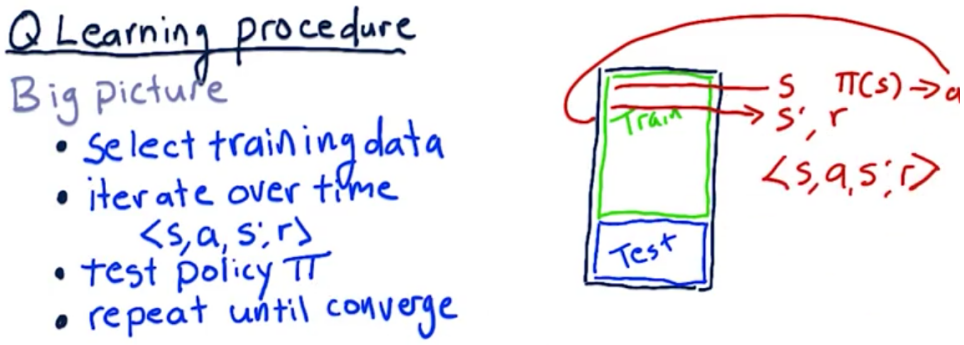
For now, suppose we fully know Q for our state space. What can we do with that? We can craft a policy $\Pi(s)$ which is just whatever action maximizes Q given our state:

$$\Pi(s) = \arg\max_a Q[s, a]$$

Eventually, given enough observations, we converge to the optimal policy $\Pi^*(s)$ and our full table Q^*. Now let's figure out how to train our agent and build Q.

7.1.1 Training a Q-Learner

At a high level, we will first split our data into training and testing data as we've done with our other machine learning methods, then iterate over time and integrate our experience tuples $\langle s, a, s', r\rangle$ into our policy Π until we converge to $\Pi^*(s)$.



"Converging" means we've reached a point at which the policy doesn't perform any better on the test data given more experience tuples. The iteration process can be further broken down as follows:

- We choose a "start time" and initialize Q to some default state. This is usually done by initializing the Q table to small random numbers.
- We compute s and select a given our policy, $a = \Pi(s)$. In other words, we consult Q to find the best action given the current state.
- We observe the reward r and subsequent state s' from the world.
- We use that information to update and improve $Q[s, a]$.

Let's look at the learning procedure in further detail.

A Learning Step

Given an experience tuple, $\langle s, a, s', r\rangle$, how do we update our Q table? We introduce a scalar α that represents our **learning rate** which is a metric of how much we "trust" new information. Without this, we would expect our q-values[1] to flail wildly with each observation;

[1] Here, we use q to signify a specific value in the Q table, so $q = Q[s, a]$ for some s, a.

with it, our values adjust more smoothly and eventually converge. A typical value for the learning rate is $\alpha \approx 0.20$.

Updating our Q table is a matter of integrating our improved estimate with our old estimate:

$$Q'[s,a] = (1-\alpha)Q[s,a] + \alpha \cdot \text{improved estimate}$$

This begs the question: what's the improved estimate? Well as we've already discussed, it's our immediate reward and our discounted future rewards:

$$Q'[s,a] = (1-\alpha)Q[s,a] + \alpha(r + \gamma \cdot \text{later rewards})$$

But what are our discounted future rewards? We haven't seen the future yet! But we can. If we assume that we will behave optimally from this point forward, we can say that our future reward is the q-value of the best action we would've chosen from our new state, s':

$$Q'[s,a] = (1-\alpha)Q[s,a] + \alpha \left(r + \gamma \cdot \underbrace{Q\left[s', \arg\max_{a'} (Q[s',a'])\right]}_{} \right)$$

Again, the underbraced part is the q-value of the best action a' from our new state s'.

Exploration

The more of our state space we explore, the more likely we are to hone in on the optimal actions to take. If we get one single observation that tells us a state change from $s_1 \xrightarrow{a} s_2$ is really *really* good (by accident), the learner has no incentive to ever try another action when in the state s_1.

To alleviate this pitfall, we can introduce the probability of choosing a random action regardless of the best action. Naturally, we want this probability to decrease over time as we converge on the optimal policy, so we additionally need a decay rate.

7.1.2 Trading as an MDP

Breaking down trading on the stock market into the elements of a Markov decision process is fairly straightforward:

- **Actions**: buy, sell, or do nothing.
- **Rewards**: profit (or loss). More generally, we can use daily returns as our reward at each timestep (a day).
- **State**: this one is tougher. We don't want something like price in our state since that doesn't generalize well to a learning model that can be applied to many different

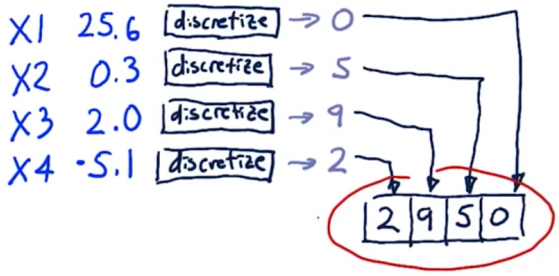
stocks. Instead, we should prefer relative values. Things like $^{\text{price}}/_{\text{SMA}}$, Bollinger bands' values, and P/E ratios all fall into this category.

Additionally, we should track state about how we're interacting with the market. Whether or not we're holding a stock is obviously useful, and that very fact may impact our decision-making. We may want to ride out a storm if we're already holding a stock, but buy any dip if we're not holding yet, for example. Our total return since entering a position is also useful if we have a particular target profit in mind, for example.

Creating the State

In our formulas and discussion, we've been assuming that our state is just an integer. However, clearly the state space we discussed above is far more complicated. We need to take some steps to consolidate our factors into something that can be used as an index to lookup a q value in our Q table.[2]

We need to discretize and combine each of our factors. Suppose we limit each of our 4 example factors, x_1, x_2, x_3, and x_4 to the range $[0, 9]$. Then, we perform our discretization and concatenate the numbers together into a single number:



A simple method of discretization is shown in Snippet 7.1. It has the benefit of naturally scaling the thresholds to the densities of the dataset. Areas that have a lot of values get smaller thresholds whereas areas that are spread out get larger ones.

Snippet 7.1: A basic pseudocode algorithm for discretizing our learning factors from arbitrary real numbers to fixed-range integers.

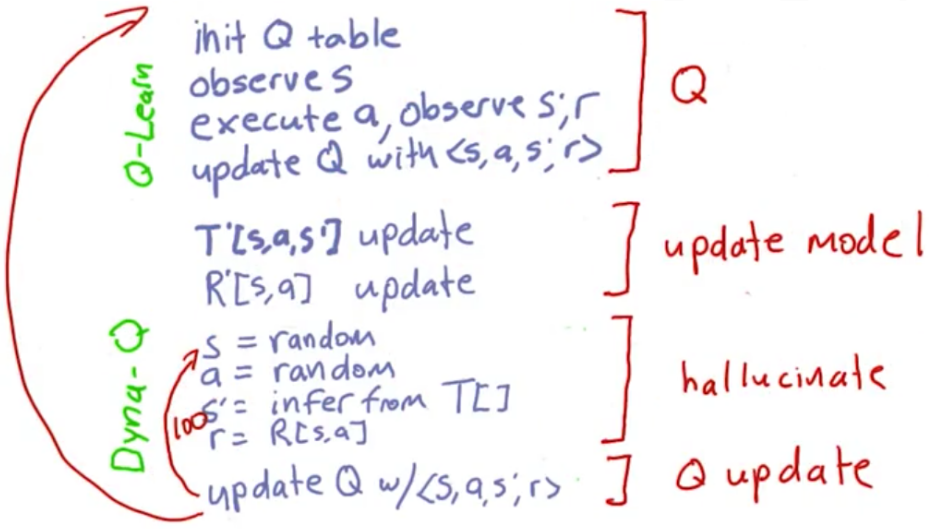
```
step_size = size(data) / steps
data.sort()
for i in range(0, steps):
  threshold[i] = data[(i + 1) * stepsize]
```

[2] There absolutely are reinforcement learning algorithms that operate with robust state spaces and don't require these steps, but they're unfortunately far beyond the scope of this class.

7.1.3 Dyna-Q

Q-learning's biggest limitation is that it requires many experience tuples (and thus real-world steps) to converge. Given our context of trading on the stock market, we'd probably burn through a lot of capital before we reach $\Pi^*(s)$. To alleviate this, we introduce the **Dyna** algorithm which will allow us to learn T and R by "hallucinating" many interactions based on a single real-world interaction, expanding our total number of experiences tuples without costing us big bucks.

Dyna-Q is a blend of model-based and model-free methods that builds on traditional Q-learning. We leverage the expensive experiences we gained on real data to develop a model, then use that model to simulate hundreds more experiences and make our Q table even better. We can repeat this for all of our real experiences, essentially using each real experience to bring back our model closer to reality.



Let's look at these steps in more detail.

Hallucination

How do we create fake experiences based on the real world? As our model of T and R gets better with real observations, our fake ones will get more and more representative of the world.

We choose a random s and random a,[3] then we *infer* the resulting state s' from our approximate transition model, so $s' = T[s, a]$. Similarly, we infer our reward from our approximate reward model, so $r = R[s, a]$.

This begs the question, of course, of how we approximate T and R.[4]

[3] Of course, the subset of available actions must be restricted to those available for the chosen s.

[4] The specifics of these methods are Dr. Balch's concoction for the course; they may not accurately reflect Sutton's implementation's in his original paper.

Learning Transitions Recall that $T[s, a, s']$ represents the probability to land in state s' given that you took the action a from state s. To learn T, we simply observe how often these transitions actually occur and count:

1. Initialize T_{count} to some small number to avoid division by zero: $T_{\text{count}} = 1\text{e}{-6}$.
2. For every experience with the real world, observe $\langle s, a, s' \rangle$.
3. Given experience, simply increment $T_{\text{count}}[s, a, s']$.

How do we use T_{count} to then evaluate a transition probability of $T[s, a, s']$? Just divide the instance's number of occurrences by the total:

$$T[s, a, s'] = \frac{T_{\text{count}}[s, a, s']}{\sum_{i \in S} T_{\text{count}}[s, a, i]}$$

Where S is the set of all states we've observed when taking action a from state s.

Learning Rewards Recall that $R[s, a]$ is the total (immediate and discounted future) reward for taking the action a from state s. We are given the immediate reward r as part of our experience tuple. To build our model, then, we simply discount old rewards and integrate immedate ones (note that $\mathbf{R} = 0$ to start):

$$R'[s, a] = (1 - \alpha)R[s, a] + \alpha r$$

Where α is our learning rate (often something like 0.2) which quantifies our trust in the new information.

To be continued...

Index of Terms

Symbols

k-nearest neighbor ... 58, **60**, 64

A

arbitrage pricing theory ... **43**
assets under management ... **24**

B

backtesting ... **59**
bagging ... **68**
basis points ... **17**
bear ... **23**
bid-ask spread ... **30**
Bollinger bands ... **8**, 46, 48, 70, 75
bond ... **38**
book value ... 38, **39**, 41
boosting ... **69**
bull ... **23**, 36

C

cap-weighted ... **41**
capital asset pricing model ... **41**, 52
convex functions ... **18**
correlation ... **14**, **65**
cross validation ... **67**
cross validation, roll forward ... **67**
cumulative returns ... **9**, 26

D

daily returns ... **8**, 10, 70, 74
dark pool ... **33**
decision forests ... 58
decision trees ... 58, **62**
discount rate ... **39**, 72
dividends ... **38**, 49, 51
Dyna ... **76**

E

efficient markets hypothesis ... **52**
expense ratio ... **24**

F

feature ... **62**
finite horizon ... **72**
fundamental analysis ... **45**

G

gradient descent ... 18
Grinold's fundamental law ... **54**

H

histogram ... **10**

I

indicators ... **45**
infinite horizon ... **72**
information ratio ... **54**
intrinsic value ... **38**, 38, 41, 51, 72

K

kernel regression ... **60**
kurtosis ... **12**

L

large-cap stock ... **23**
learning rate ... **73**
linear regression ... 13, 58, **59**, 68
liquid ... **23**

M

market capitalization ... **23**, 38, 40, 41
market porfolio ... **41**
Markov decision problem ... **70**
Markov decision process ... 74
model-based learning ... **71**

model-free learning **71**, 72
momentum **46**, 47

N

normal distribution **11**

O

order **28**
order book 28, **29**
overfit 64
overfitting 64, **66**

P

policy iteration **71**
portfolio **15**, 41
portfolio management, active **42**
portfolio management, passive **42**

Q

Q-learning 71, **72**

R

rolling mean **7**
rolling standard deviation **7**
root-mean-squared error **65**

S

scatter plot **10**, 13, 42
sell short 26, 29, **35**, 50, 70
Sharpe ratio **16**, 20, 26, 56
simple moving average **47**, 48
slope **13**, 42
stock split 49, **50**, 51
supervised regression learning **58**, 71

T

technical analysis **45**
tick **50**
two and twenty **25**

V

value iteration **71**
volume **23**