# Problem Statement:

Most financial analysts present different estimates of the earnings for different companies. The consensus earnings estimate is an average of these estimates. This consensus is correct approximately 60% of the time. There are a number of reasons why analysts incorrectly predict earnings for companies; some of the common ones being conflicting interests, personal biases, and manipulation.

Nobel-laurate Daniel Kahneman mentions that more and more trading firms should trust algorithms, not people. Being cognizant of the bias humans might present, the goal with this project was to build a model that predicts the likelihood of a company’s actual eps beating analysts’ estimates using stock market movements in the past quarter. This model can then be used to forecast which companies are going to beat the Wall Street estimate of EPS and guide trading decisions from this forecast.

# Step 1: Data Collection

In order to build such a model, we need to collect the right data.

1. The first dataset we used is from estimize (sourced from [https://data.estimize.com](https://data.estimize.com/)). This data contains a date (the date that the earnings were released), the ticker code of the stock, the consensus wall street estimate for the EPS in that month, and the fiscal year and quarter corresponding to the estimate. We have this data for 39 Major Tech Companies (listed in finviz\_tech.csv), for each quarter from 2000 - 2014. The code that pulls together the data can be found in estimize.py

A screenshot of a cell phone

Description automatically generated

1. Along with the earnings data, we also want to get stock price data to use for our forecast. Given that earnings happen once every quarter, we chose to gather data by the day which was easily available from Yahoo finance. In this dataset, we can see that we have the daily Open, High, Low, Close, and Trading Volume associated with the stock. This data set was pulled for the same 39 companies and the same time period and was sourced from Yahoo Finance. The code that pulls together the data can be found in yahoo\_quote.py

A screenshot of a cell phone

Description automatically generated

# Step 2: Data Cleaning and Preparation

The stock data that we have collected is daily, and the EPS is released quarterly. Each company has its own earnings release date and fiscal calendar. Our first task is to align each company’s fiscal calendar to its corresponding fiscal year and fiscal quarter. The steps to do this are laid out in the Code Presentation notebook.

A close up of a piece of paper

Description automatically generated

As you can see in the sample dataset, we have correctly aligned the fiscal quarters for Adobe (ADBE) stock to the right trading days – the quarter changes from Q1 to Q2 when going from February to March. While we have picked this example for its simplicity to illustrate the point, many stocks do not have such straightforward reporting periods. The goal of the data preparation task was to line up all of the datasets correctly in order for our model to use them.

# Step 3: Feature Creation

Selecting the right features is one of the most important steps for generating a good model. When creating features, there are certain challenges that we need to keep in mind:

1. The challenge is to select features that are predictive of the desired target (beat/non-beat).
2. The features chosen should make sense to a human ‘analyst’ (visually be able to make predictions using them)
3. The features chosen should conform to input expectation of the model (i.e. cannot use negative features in Naïve Bayes models, or have null values in Random Forests)

The features can come directly from the data (e.g. quarter, or industry sector) or be engineered (i.e. created from hypotheses or technical analysis).

In our case, we are dealing with companies in a specific industry (technology) so we mostly created engineered features based on hypothesis. The main underlying hypothesis that we wanted to investigate was whether the movement in stocks were indicative of some kind of “market information” that would lead to the company beating EPS estimates. With this underlying hypothesis in mind, we aimed to create a mathematical representation of this hypothesis (see point 3 above) so that the features met the input expectation of the model.

The features that we engineered are:

* If the Closing Price > Opening Price more than 50% of the time (or % of days close > open)
* If the Price on any given day was above 10-day simple moving average (sma) more than 50% of days (or % of days close price is above 10-day sma)
* If the Price on any given day above 60-day simple moving average (sma) more than 50% of days (or % of days close price is above 60-day sma)
* % of times daily price or volume went above a certain threshold

# Step 4: Building the Model

## Step 4.1: Thinking about testing

Before we choose to build any model, we need to lay out the methodology of how this model can be evaluated. In order to do such an evaluation, we need data for which the truth is known (i.e. we need to hold out a subset of our prepared data that the model doesn't see).

Hence, we are left with two choices:

1. Train the model using all historical data available and test the model’s performance when new data arrives
2. “Hold Out” a small subset of the historical data, and pretend to run the model on that subset. Since the data that is held out is historical, the truth is already known and we can evaluate our model effectively.

We chose option 2 to test our model. As such, we put 90% of the data into the training set, and 10% of the data into the holdout testing set. In addition, we take the step to stratify our sampling across the ticker codes (i.e. we want to make sure that our 90% and 10% split of the overall data contain the same relative proportion of companies). It is important to stratify the dataset when sub-populations are involved.

## Step 4.2: Building the Model

We can use multiple algorithms to train the model. Since we have framed our problem in the following way:

Target: Find cases where the actual EPS was higher than the wall street consensus EPS. This will allow us to train a predictor that can, in the future, help us forecast if the EPS is going to be higher than the consensus estimate.

Mathematically, what we are building is called two-class supervised learning classifier. The model takes inputs and predicts the probabilities that the output is one of two "classes" - in our case, whether the company will "beat" the EPS consensus or not. We need to label the classes in binary form: following convention, we will label the class that we are interested in as a 1, and the opposite case a 0 (i.e. when EPS > Consensus, then "beat" = 1, else 0).

Multiple algorithms exist for a two-class supervised learning classifier. Such as: Logistic Regression, Ensemble Models, Random Forests, XGBoost. Upon trying multiple models, it was found that the Random Forest provides the best accuracy.

## Step 5: Results

When evaluating the result of a model, it is important that we choose more than one metric that provides a holistic picture. Amongst the many different available metrics, the following three were selected. In each case, the higher the number, the better the model

##### ***Precision – How correct is each prediction by the model?***

Precision looks at how often our model has false positives. This is represented by the formula: (Number of Predicted Actual Positives)/(/Total number of Predicted Positives). Higher value in this case is better, since it means that we have very little actual 0’s falsely classified as 1’s.

##### ***Recall – How correct is the model in finding all the opportunities?***

Recall looks at how sensitive our model is. It seeks to answer the question – how many of the actual 1’s did we correctly identify. This is represented by the formula: (Number of Predicted Actual Positives)/(/Total number of Actual Positives). Here too, higher number is better, as it means that we have fewer actual 0’s classified as 1’s.

##### ***Accuracy – How often do we make the right decision?***

Accuracy looks to answer the question – how often do we make the right decision. In other words, does model correctly classify 1’s when actual value is 1, and 0’s when actual value is 0. This is captured in the formula: (Number of Correct Examples)/(Total number of Examples) . This metric is however sensitive to imbalanced classes (cases when number of 1’s is greater than 0’s, as is the case here since companies more often beat consensus estimates based on the way we calculate a beat)

Our best model had the following metrics:

Model Precision: 0.752222

Model Recall: 0.980944

Model Accuracy: 0.752389

The baseline accuracy that we are comparing ourselves to is 66%. This baseline comes from the fact that if we were to say that every single company will beat the EPS estimate in every quarter, we will be right 66% of the time.

We can see from the results that we improve on the baseline (66%) precision and accuracy by 9%. Furthermore, we improve this baseline without sacrificing much in the way of recall.

In human judgement, we trade off heavily between precision and recall - e.g. saying every stock is going to beat estimates every quarter is a recall of 1 (since you get the actual beats right 100% of the time). However, the recall suffers substantially (only 66% of the predictions will be accurate).

In this model, we only trade off 2% of the recall (100% to 98.1%) for a 9% increase in recall. This is a great start and the increased accuracy from such a model can be leveraged well to make trading decisions.

The model can be improved by trying more advanced methods (such as RNNs, CNNs, Deep Learning, etc.) and creating more features (using hypotheses and data) as we demonstrated. These steps should be taken incrementally with an eye on the precision and recall tradeoffs.