1. In 2000 Enron was seen as an innovative company that employed over 20,000 people, reported profits of one hundred and one billion dollars and had won numerous awards for innovation in business. By the end of 2001 it was bankrupt. On top of this, it was revealed that Enron was propped up by "…institutionalized, systematic, and creatively planned accounting fraud".

   The goal of this project is to use machine learning to create a classifier capable of identifying people of interest (POI in the dataset). The dataset consisted of information from real Enron employees including salary and payment information, information regarding emails and whether or not the individual was counted as a POI by the official investigation. The dataset consists of 146 individual records with 14 financial features, 6 email features, and 1 labeled feature (POI). There are 18 POIs in the data set.

   In the case of this dataset, the POIs had already been obtained for all legal purposes. However this project is useful in several ways. It allows for the exploration and expansion of what machine learning can accomplish as a tool. It also creates a framework for similar investigations in the future.

   In this case it is important to keep in mind that we are interested in a relatively wider net when it comes to false positives. Since being identified as a POI only results in being investigated, we would ideally want a classifier with more false positives than false negatives.

   This data set, being a real world (read messy) data set has numerous gaps and several outliers. The most glaring outlier is the value TOTAL. This was probably due to the data coming from a spreadsheet that included a total value for most numerical values. Since I am not interested in the total values, I removed this from the data. Similarly, I removed THE TRAVEL AGENCY IN THE PARK from the list of individuals, as it would represent an institution and not an individual. I also removed Eugene Lockhart, who had no non NaN values in the data.

   There were also many missing values in the data set. In the case of the email data, I chose to fill in the missing data with the median values. There was a wide range of emails sent so I felt that the median would be a more appropriate value. For financial information, I mostly replaced NaN values with 0. In the case of bonuses or raises, I made the assumption that missing values could safely represent no bonus. However, I decided to replace salary with the median value again on the assumption that median salary for an employee would probably be a more realistic substitution than no salary.

2. I created three new features: the total financial assets (total_assets), the rate of emails to a POI (rate_to_poi) and the rate of emails from a POI (rate_from_poi).

The rate variables have the potential to cause data leakage since they contain information that is used to create the feature and information about the target. However, none were selected by SelectKBest for the best features. When the engineered features were included, the importances increased their values and in several cases, different features were ranked more highly than others.

I began by testing four algorithms using the bare feature list including my engineered features. AdaBoost and Decision Tree work the best so I will be testing these two more fully. I then manually computed the scores of the ten highest features for each classifiers. The results are as follows

```
 Decision Tree Feature Importances Without Engineered Features:

exercised_stock_options : 0.1998
restricted_stock : 0.1205
total_payments : 0.1130
bonus : 0.1091
long_term_incentive : 0.1090
from_messages : 0.0874
expenses : 0.0649
shared_receipt_with_poi : 0.0578
from_this_person_to_poi : 0.0555
deferred_income : 0.0530

Decision Tree Features With Engineered Features:

exercised_stock_options : 0.2165
shared_receipt_with_poi : 0.1579
deferred_income : 0.1326
total_stock_value : 0.1033
from_this_person_to_poi : 0.0585
salary : 0.0424
expenses : 0.0424
restricted_stock : 0.0252
from_poi_to_this_person : 0.0152
to_messages : 0.0000



Ada Boost Feature Importances Without Engineered Features:

expenses : 0.3383
other : 0.1160
exercised_stock_options : 0.0937
from_this_person_to_poi : 0.0792
total_payments : 0.0557
bonus : 0.0549
```

```
restricted_stock : 0.0469
restricted_stock_deferred : 0.0357
to_messages : 0.0303
total_stock_value : 0.0278

Ada Boost Feature Importances With Engineered Features:


expenses : 0.1800
total_payments : 0.1200
restricted_stock : 0.1200
other : 0.1000
salary : 0.0800
shared_receipt_with_poi : 0.0600
from_messages : 0.0600
deferred_income : 0.0600
exercised_stock_options : 0.0400
bonus : 0.0400
```

I then used SelectKBest in conjunction with GridSearchCV to test for the optimum number of features and included this information later in the process. The optimum number chosen for the decision tree classifier was 6 and for the adaboost classifier it was 11. The optimal features used in my final classifier, decision tree, were:

```
exercised_stock_options : 0.2165
deferred_income : 0.1326
expenses : 0.0908
shared_receipt_with_poi : 0.0898
restricted_stock : 0.0681
from_poi_to_this_person : 0.0585
```

Using SelectKBest with gridsearchCV allowed me to test all parameters to see which returns the best score. I used a scoring parameter of F1, which combines the accuracy and precision into a single score, allowing me to maximize both. I also set cv to 10 splits, which helps protect against overfitting.

Since I primarily focused on AdaBoost and Decision Tree classifiers, it was not strictly necessary to scale the features, although I did so using a MinMax scaler.

3. I initially tested Gaussian NB, AdaBoost, Decision Tree and KMeans classifiers but I settled on AdaBoost and Decision tree for the two I would continue to tune. Most of the classifiers

scored between .2 and .3 on precision and there was a range of accuracy from around .2 to an abysmal .07250.

All classifiers tested had a fairly high accuracy score. In this case, accuracy is the least important variable. All that accuracy tells us is what percentage of the data was correctly classified. Recall and precision can tell us more. Recall is the probability that the classifier labeled an event (being a poi) as true given that it is true. In a sense, precision measures the inverse of recall. Precision is the probability that an event is true (again, the person is a poi) given that the classifier predicted it was true.

By the end Decision Tree provided the best scores, as measured in F1, the harmonic mean of the accuracy and precision.

4. In machine learning, selecting the best algorithm is only the first step. Since the ideal is to maximize the usefulness of the algorithm, tuning is an important step. Many algorithms come with a variety of parameters that can be set to different values. Depending on the dataset, tuning these algorithms can result in more accurate (better levels of accuracy, precision and recall) results.

   It is important, however not to overfit your data. In the real world, a robust algorithm must be able to handle not just the data from the training set but from future sets. An overfit algorithm will perform extremely well on its training set and fail on others. Similarly an underfit algorithm will perform poorly on its training data right off the bat. The ideal is the happy medium; generalized enough to cover data from multiple sets but specific enough to perform well on its training.

5. Validation is checking your algorithm against more than one set of data, to ensure that you avoid over and underfitting, as described above. Improper tuning and improper validation go hand in hand. This problem is largely circumvented by the stratified shuffle split, which splits the training data into training and testing sets that preserve the ratios of true and false. This is especially useful when dealing with limited and small data sets, such as the one we have here.

   Within both classifiers that I tested, I tuned the various parameters using a pipeline with selectKBest to check exhaustively the best parameters to set. Since I ended up using a decision tree I updated the criterion, max depth, max features and minimum sample split.

   With the adaboost classifier I tuned the base estimator, learning rate and n estimators. After tuning both, the decision tree classifier returned the best results.

6. Using the tester.py script to evaluate my algorithm, I got scores of:

```
Accuracy: 0.82227 Precision: 0.32547 Recall: 0.31050 F1: 0.31781
```

F1 is the harmonic mean of the precision and recall score designed to show the relationship of the two. For more discussion on precision and recall see question 3. These are all over .3, the requirement for this project.

The accuracy is the rate of correct predictions. This doesn’t tell us all that much. Although it is a good score. A classifier that classified everyone as non POIs would return an accuracy of 87.67%. This is higher than the accuracy of my classifier, but it is much less helpful. Accuracy is not all it is cracked up to be.

The precision and recall are more important. Precision is the probability that the person is a POI given that the algorithm said they were. Our score is 32.547%. Note that random guessing would give us a probability of 18/146 or 13%. This is nearly a threefold increase. Not bad.

The recall is perhaps the most important measure here. Recall refers to the probability that the algorithm will predict someone is a POI given that they are labelled as such in the dataset. This score is 31.05 %. This essentially means that 69 percent of the people labeled as POIs in the data set go undetected. Although this number is nice, it could be a lot better.

This also shows that although data analysis and machine learning are extremely useful tools, they have to be properly understood and utilized well. They are best used along with other tools to obtain the best outcome.