1) In 2001, it was discovered that Enron had engaged in many complicated and unethical business practices to hide billions of dollars of debt. All this ultimately led to its bankruptcy, which was the largest bankruptcy at the time. The majority of the problems that led to Enron's demise were perpetuated either directly or indirectly by a group of executives that we will refer to as persons of interest (POI). Although the Enron scandal was an egregious failure in many ways, one good thing did come out of it and that is a large corpus of interesting financial and email data.

   The goal of this project is to create a classifier that will try and determine whether or not a person is a POI based on available information. This problem can be viewed as a supervised learning problem because labelled data already exists (i.e. POI or non-POI). The available information has been preprocessed into a dataset that includes 146 entries. Two of the entries had the name `"TOTAL"` and `"THE TRAVEL AGENCY IN THE PARK"`, which I considered to be outliers because they do not represent people. These two values were removed leaving 144 entries in the dataset. Each entry has 21 features. 14 of these features are financial features (e.g. salary, bonus, exercised stock options) and 6 email features (e.g. email address, number of from messages, number of to messages). One feature is the 'poi' feature which is true if the person was a person-of-interest and false otherwise. 35 people did not have an email address associated with them and so did not have email features. Some financial information was missing as well.

2) I created a feature that counts the number of emails sent to people whose domain name did not include Enron (`"to_not_enron_messages"`). I thought that maybe people who frequently communicated with outside parties (e.g. auditors, accountants, etc.) might be POIs. There were 25 people in `data_dict` who couldn't be found in the Enron dataset so this feature was not assigned for them.

Feature scaling would be necessary if I decided to use SVM because SVM is not affine transformation invariant. The financial features have numbers that are several magnitudes of order larger than the email features and so, without scaling the features would not be weighted equally.

I used the following features to begin with: `"shared_receipt_with_poi"`, `"to_not_enron_messages"`, `"total_payments"`, and `"bonus"`. This gave me the following scores and feature importances:

- `Precision: 0.24963`
- `Recall: 0.25000`
- `fscore: 0.24981`
- `[ 0.20855256  0.04807692  0.53546139  0.20790913]`

I tried using only email features (`"shared_receipt_with_poi"` and my engineered feature `"to_not_enron_messages"`). Together, these gave me the following scores:

- `Precision: 0.27142`
- `Recall: 0.39600`
- `fscore: 0.32208`
- `[ 0.66827324  0.33172676]`

Because my engineered feature did not have a high importance compared to `"shared_receipt_with_poi"` I created a classifier using only this and got the following scores and feature importances:

- `Precision: 0.30198`
- `Recall: 0.32100`
- `fscore: 0.31120`
- `[ 1.]`

The best two features based on feature importance were `"total_payments"` and `"bonus"`. These were also the best two features selected using a Chi-squared goodness of fit test with SelectKBest. This gave me the following scores and feature importances:

- `Precision: 0.31074`
- `Recall: 0.31400`
- `fscore: 0.31236`
- `[ 0.56691466  0.43308534]`

Ultimately, I decided recall to be more important than precision. Therefore, I decided to just use `"shared_receipt_with_poi"` because the recall score was higher – although precision was lower.

3) I ended up using a decision tree classifier. I also tried a support vector machine classifier as well as a Naïve Bayes. I tried all three algorithms initially without any parameter tuning and found the following results:

   - SVC did not classify anybody as a POI so I got:
     - `Precision: 0.00000`
     - `Recall: 0.00000`
     - `fscore: 0.00000`
   - Decision tree gave me results that were close to the required recall and precision scores. With different feature selection and parameter tuning I think this could work:
     - `Precision: 0.21528`
     - `Recall: 0.21700`
     - `fscore: 0.21614`
   - Naïve Bayes gave me results that were not very good for recall:
     - `Precision: 0.13630`
     - `Recall: 0.05200`
     - `fscore: 0.07528`

4) Tuning the parameters of an algorithm changes its performance and the ultimate goal is to find a good balance between bias and variance. If an algorithm's parameters are not tuned, the default settings can cause the classifier to be overfit (high variance) or too generalized (high bias).

   I used `GridSearchCV` to tune parameters when I used several features and got:

   - `Precision: 0.25085`
   - `Recall: 0.25950`
   - `fscore: 0.25510`

   However, after deciding to only use one feature I found the default settings for the `DecisionTreeClassifier` gave me the best results:

   - `Precision: 0.30198`
   - `Recall: 0.32100`
   - `fscore: 0.31120`
   - `[ 1.]`

5) Validation is necessary to make sure the results you have obtained from training are trustworthy. If validation is not done correctly you could have a classifier that is either too closely fitted to the training set (high variance) or is oversimplified (high bias). Having a classifier that performed much better on training data than test data would not be robust enough to be used in a practical application.

   I do believe there is a flaw in the way that this project is set up because we do not set aside a test set to evaluate the performance of the classifier. Instead, the entire dataset is used by `test_classifier` and this would likely to give us a lower error than we would see on a test set that hadn't been 'seen' before. (Hastie, Tibshriani, & Friedman, 2008):

   > If we are in a data-rich situation, the best approach for both problems is to randomly divide the dataset into three parts: a training set, a validation set, and a test set. The training set is used to fit the models; the validation set is used to estimate prediction error for model selection; the test set is used for assessment of the generalization error of the final chosen model. Ideally, the test set should be kept in a "vault," and be brought out only at the end of the data analysis. Suppose instead that we use the test-set repeatedly, choosing the model with smallest test-set error. Then the test set error of the final chosen model will underestimate the true test error, sometimes substantially.

   Unfortunately, I don't think anything can really be done about this because our dataset is quite small. I used stratified shuffle split to validate my classifier. It was important to use stratified sample selection because the percentage of POIs within the dataset is small. There are only 18 POIs and the remaining 126 people are innocent. Stratified Shuffled Split will preserve the percentage of samples for each class because given the small number of POIs, without stratified samples it is possible no POIs end up in the training or test set.

6) I used recall and precision as evaluation metrics to gauge the performance of my classifier. Recall is the ability to find and identify all samples of a certain label. Precision is the ability to correctly label a sample. In other words, precision is a measure of the likelihood that a person identified as a POI is truly a POI; and, recall is a measure of the likelihood that a POI is identified as such. I think that it makes more sense for this classifier to have higher recall – perhaps at the expense of precision. It would be easier to dismiss a person erroneously identified as a POI over the course of an investigation. However, I think it would be dangerous to miss identifying a person as a potential POI altogether, because it is less likely that investigators would even look into a person who had to been flagged at all.