1) The goal of this project is to utilize machine learning techniques to analyze a data set and build a predictive model. Specifically, we analyze the Enron data set which contains financial and email data from a number of employees of that former corporation which suffered bankruptcy due to employee fraud. Our model attempts to predict whether an individual in the data set is a person of interest (i.e. whether they may have committed fraud), a feature for which each person in the data set has a value.

An exploration of the data set revealed it to be relatively scant. It contains 146 individuals of which only 18 are persons of interest. There are 21 features (including the target) and many features contained various levels of missing/non-existent values. Each individual in the data set had the 'person of interest' (POI) feature and value as expected, but many features such as 'salary' and 'email address' had many missing entries. For example, Andrew Fastow, who was arguably the head chef of cooking Enron's financial books, did not have any email data and (if documented) his communication accountant Arthur Andersen would have proved to be valuable. In order to avoid data leakage I imputed missing numerical data with a zero value (instead of an average) and text data with an empty string. There were several outliers in the data set that were removed because they either represented a cumulative total, were not a person but an organization, or did not have enough relevant data. These were identified graphically, by manual inspection, and programmatically.

2) Before pruning any features, I engineered several word features (based on a training data set) for which the value is a numerical count of the frequency of that word used by each person. The features were created by stemming each word (linguistic root) in a person's emails, computing term frequency-inverse document frequency (TF-IDF) of all words for all people in the data set, fitting a model using the TF-IDF to predict POIs on the training set, extracting the most important words from the model, and using those important words to count how many times each was written by a person. Unfortunately, and as expected, the words features did not help much in predicting POIs so they were discounted. I believe that the small number of POIs, some of which had no email data, affected this.

Features selection proved to be the most difficult part of the project. I tried several different ways to programmatically determine the best features to use in my model but they were ultimately identified by manual inspection. In a drawn out attempt to identify these features, I created a combination of univariate analysis, pearson correlation analysis, recursive features selection, and three tree based models (decision tree, random forest, adaBoost). In the end, I only used them as a guideline for the manual selection.

Below are the scores from the Kbest analysis:
[('bonus', 30.652282305660439), ('salary', 15.806090087437418), ('total_stock_value', 10.814634863040405), ('exercised_stock_options', 9.9561675820785211), ('total_payments', 8.9627155009973993), ('deferred_income', 8.4934970305461821), ('restricted_stock', 8.0511018969982544), ('long_term_incentive', 7.5345222400328424), ('loan_advances', 7.0379327981934612), ('expenses', 4.3143955730810664), ('other', 3.1966845043285219), ('director_fees', 1.6410979261701475), ('restricted_stock_deferred', 0.67928033895169282), ('deferral_payments', 0.0098194464190455126)]

3) My final algorithm was a k-nearest-neighbors classifier that uses range scaling to preprocess the data. It was the best at maximizing recall scores, followed closely by an adaboost classifier which uses normalized scaling and principal component analysis. A gaussian naïve bayes classifier was used as a benchmark to compare model scores. I also tried using a linear regression classifier, random forest classifier, and k-means clustering (unsupervised prediction of 2 clusters) but none performed as well as the previous three.

4) Both of the top performing algorithms that I used accomplished their best F1 scores after parameter tuning. For example k-nearest-neighbors performed best when it used either 1 or 2 nearest neighbors. Adaboost performed best at learning rate much higher (10 or 20) than the default of 1. The gaussian naïve bayes algorithm was the only model that did not need any tuning as it did not have any parameters to adjust.

In order to tune these parameters I used a grid search over a stratified shuffle split cross validation used on the training portion of the data (all model generation was done with the same training set to avoid overfitting). The parameters tuned for the k-nearest-neighbor classifier were the number of nearest neighbors (from 1 to 6), and the weights (uniform or inverse distance). Tuning was critical to optimize algorithm performance. For example, had I used 3 or more nearest neighbors for the final algorithm, the evaluation metrics decreased significantly.

5) Validation was the defining moment of model selection as it established which models were superior to others. It was first done by fitting on a single training portion of the data and scoring on another test portion. Because the holdout testing portioning of the data set was so small, inaccurate results were reported. This gave the impression that some models were better than others when in fact they were worse. To rectify this I used a stratified shuffle split with 200 folds to preserve the proportion of POIs across multiple train/test splits.

6) Accuracy, precision, recall, and F1 scores were reported to gauge model performance. Following are the parameters and scores for the benchmark model (gaussian naïve bayes), the second best model (adaboost), and the best model (k-nearest-neighbors):

```
GaussianNB()
        Accuracy: 0.84531     Precision: 0.49578     Recall: 0.32300        F1: 0.39116    F2: 0.34720
        Total predictions: 13000        True positives:  646    False positives:  657   False negatives: 1354
        True negatives: 10343

Pipeline(steps=[('scaler', StandardScaler(copy=True, with_mean=True, with_std=True)), ('reducer',
PCA(copy=True, n_components=None, whiten=False)), ('ada',
AdaBoostClassifier(algorithm='SAMME.R',
       base_estimator=DecisionTreeClassifier(compute_importances=None, criterion='gini',
         max_depth...om_state=None, splitter='best'),
       learning_rate=20.0, n_estimators=50, random_state=None))])
        Accuracy: 0.82238     Precision: 0.41924     Recall: 0.40100        F1: 0.40992    F2: 0.40452
        Total predictions: 13000        True positives:  802    False positives: 1111   False negatives: 1198
        True negatives: 9889

Pipeline(steps=[('scaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('knn',
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
        metric_params=None, n_neighbors=1, p=2, weights='uniform'))])
        Accuracy: 0.83715     Precision: 0.46704     Recall: 0.41450        F1: 0.43921    F2: 0.42404
        Total predictions: 13000        True positives:  829    False positives:  946   False negatives: 1171
        True negatives: 10054
```

The most accurate and precise model happened to be the benchmark gaussian naïve bayes classifier (0.84, 0.49). Accuracy is the proportion of times that we're able to correctly predict the class of all observations. In terms of our data set, it means that if you give us a person, our model can correctly predict whether or not they are a POI about 84% of the time. Precision represents the proportion of times we are correct when our model has chosen a positive classification. In other words, when we evaluate somebody and our model says that they are indeed a person of interest, we are correct about 49% of the time.

I would argue that recall is probably the most important metric in terms of our data set and this is why I chose to rank model performance based upon it. Recall represents the proportion that the model can correctly identify a positive classification given it is has one. In the context of this data set it represents how often we can identify a POI when we are actually looking at one. You might call this the 'catch the bad guy' ratio and the k-nearest-neighbors model achieved this about 41% of the time. Although this isn't great, I believe that had we more observations our model generation and evaluation metrics would be better.