The goal of this particular exercise is to build a model that can identify persons of interest (POIs) from the Enron dataset. In doing so, the model can help identify those individuals that may not have a POI tag but are indeed POIs. The Enron dataset includes a number of individuals that had worked at Enron in some capacity combined with a number of their compensation stats not limited to their salary, loan advances, bonuses, total stock value, expenses, and a variety of email stats like to and from messages.

The dataset in this project includes 145 individuals with 21 features per individual. One of those features represents status as a person of interest or POI. There are 18 POIs and 127 non-POIs.

There are quite a few missing values, 1358 out of a total of 3045 values or approximately 45%. As per the instructions, `featureFormat` replaces missing values with '0'. The immediate trouble here is that when treating the data for outliers, many of these '0' coded values will represent a large deviance and as such will be removed from the analysis. With this representing 45% of an already sparse dataset it likely will create a situation in which our model greatly overfits to the data. Additionally, certain features have higher percentages of missing values. For instance, loan_advances has only 3 data points and restricted_stock_deferred only has 17. These sparse features will likely be dropped out of the analysis as they cannot contain enough variance to affect the POI classification. However, this is an initial assumption and I will use a preprocessing method to suss this out.

The initial treatment for outliers was to remove individuals from the dataset that exhibited a feature more than 3 standard deviations away from the mean of the series. This step excludes any POI. POIs are initially identified and excluded from the outlier removal process. The reason here is that given such a small dataset and small sample of POIs, I can artificially manipulate the training set to retain POIs even if they are characteristic of an outlier and create a more generalizable model. By retaining the POIs, I offer the opportunity for the model to better train by allowing it to "see" a larger portion of POIs than otherwise would be the case. Non-POIs however I will remove from the analysis as they likely do not represent any meaningful pattern in the data.

For the outlier removal, I choose 3 standard deviations as this threshold excludes 5% of the data assuming the population exhibits a normal distribution. The transformation resulted in a reshaped dataset of 119 individuals. Before anything else, let's prepare to run a model.

First I'll want to split it. I choose a `StratifiedShuffleSplit` as this preserves the number of samples for each class. Given such a low number of POIs, it is important to be able to train on a model that at least has some examples of POIs. Once I have my features and labels split into training and test sets, I can proceed with building an initial model.

My first model was an `ExtraTreesClassifier`. I added a confusion matrix, precision and recall measures, and a mean `cross_val_score` with 10 folds to understand the model's performance. Precision identifies how many cases are true relative to how many cases I predicted to be true whereas recall identifies how many cases are true relative to my predictions. Precision let's me know how often my predictions of a POI turned out to actually be a POI and recall let's me know how often POIs were predicted to be a POI.

The model's parameters are tuned through the use of `GridSearchCV` with 5 folds. The `ExtraTreesClassifier` was tuned to have a *criterion* = 'gini', *n_estimators*=50, *max_depth*=4 and *min_samples_split*=5, and this resulted in a precision of .99 and recall of .20.

While this model yielded a high precision, recall left a little to be desired. I decided to switch from a tree based model to a Linear Support Vector Classifier (`LinearSVC`) which was tuned for its *C*, *class_weight*, and *loss*. *C* trades simplicity for more samples to use in classification, class weight adjusts weights of classes based on the inverse of their frequencies (if not present, all weights default to 1) and loss is either the standard SVC loss function or the squared result of that function. The classifier was tuned to *C*=10, *loss*=hinge, and *class_weight*=None. The resulting precision is .47 and recall is .19. While precision and recall are more closely matched, the resulting recall score is still below our threshold of .30.

I decided to engineer a couple features that may provide additional variance for the `LinearSVC` model to capture. These features captured the sum and the mean values across each row. These values represent factors that involve interactions of features, just not in an explicit manner. These types of features are computationally cheap to engineer and many times can provide additional insight for the model to train on. Including these features in the above `LinearSVC` results in a precision of .47 and a recall of .19. The model seems to have garnered no additional information through those particular features.

The next step was to identify the number of features that ought to be included in the pipeline. It could be the case that some features merely add noise to the model. Given the two engineered features add nothing to the scores I am evaluating, I incorporated `SelectKBest` into my `Pipeline`. I subjected the `Pipeline` to a `GridSearchCV` with 5 folds to capture the appropriate number of features. The result was a model with 5 features. The features with their respective scores were: salary (30.53), expenses (0.93), from_poi_to_this_person (24.87), deferral_payments (0.20), and exercised_stock_options (25.87).

Additionally the parameters for the `LinearSVC` changed. *C* became 1000000 and *loss* became 'squared hinge' while *class_weight* remained None. This is quite a large value of *C* however and as such I reduced this number to 10 to test the model. The resulting model scored a precision of .58 and a recall of .23. This is better than our other models, however, given the small number of POIs, I became suspicious of the homogenous class weighting. As such I altered the class weight to be balanced. The resulting model scored a precision of .40 and a recall of .61.

Parameter tuning is quite significant when building decision trees that may yield different results based on weightings of their leaf nodes, number of subsamples of the feature space or the depth of the tree being built. With so many parameters, the default model may not appropriately capture the variance in the data. Additionally, an improperly tuned model may highly over- or under-fit the particular dataset. The final `LinearSVC’s` *C* parameter highly fit the training data as evidenced by the Grid Search I performed, however, common sense and manual tuning is required when this type of parameter optimization results in extreme values. It is important to spend some time tuning. Alternatively, one may build a large enough ensemble of uncorrelated classifiers and/or regressors that tuning each model may become irrelevant.

It is also important to note why my model is trained on a portion of the data set split by `StratifiedShuffleSplit`. Without reserving a validation set, I would have no idea how my model may be performing with unseen data. The validation set allows me to train a model and check my results on an unseen dataset that contains actual y values. Additionally, a function such as `cross_val_score` can identify the average accuracy across different folds of the dataset. This prevents the model builder from exploiting a model which performs very well on a certain section of data but does not generalize to other portions of the data. For instance, if the data is split but the model trains on data that does not include any data of a particular class, it will not be able to evaluate instances of testing data in which that class is present. The cross validation technique allows one to inspect how well the model performs across a specified number of folds of the dataset.

The dataset was scaled to account for the `LinearSVC` model. As this model measures high dimensional planar distances, features with a large degree of variance or at very different numerical ranges can artificially add complexity to the model, resulting in skewed results. I used `MinMaxScaler` within the `Pipeline` to rescale the dataset to the range [-1,1] before testing on the model.