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 data points, 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 our 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 we 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. 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 111 individuals. Six of the 34 individuals in the excluded data were POIs such as Ken Lay and Jeff Skilling. Given 12% of the original dataset were POIs the excluded data represents a ~33% increase in the number of POIs identified. This may pose a problem in building a model as our outlier detection disproportionately reject POIs. To test this, let's run a model.

Before I submit the data to a model, 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 a `RandomForestClassifier`. 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 the proportion of positive cases I predicted to true cases and recall identifies the proportion of positive predictions that were true. Our main concerns are precision and recall and for this model precision was poor (~.19) and recall was worse (~.09).

A number of different algorithms were implemented untuned. K-Means Clustering and K Neighbors Classifier performed best but in alternate directions. K-Means resulted in higher recall and lower precision while K Neighbors resulted in higher precision and lower recall. The final model combines both models in a `Pipeline`. K-Means allows a fit-transform resulting in a transformed dataset that can be modeled by `KNeighborsClassifier`. The resulting model offered a poor precision of .002 and a recall of .001.

I decided that adding features that take the sum and the mean across columns in the dataframe could yield some additional information about the relationship between the variables that is not currently captured. This was not the case as precision dropped slightly and recall remained the same.

The next step was to identify the number of features that ought to be included in the model. It could be the case that some features merely add noise to the model. I used `SelectKBest` by incorporating it in our `Pipeline`. I subjected the `Pipeline` to a `GridSearchCV` with 5 folds to capture the appropriate number of features. The result was a model with 9 features. The resulting precision was approximately .6 and recall .35. The features selected were: salary, total_payments, bonus, restricted_stock, shared_receipt_with_poi, expenses, other, deferred_income, from_poi_to_this_person.

This final model requires little tuning. Both `KNeighborsClassifier` and `KMeans` clustering involve a single major parameter. `KNeighborsClassifier` was tested on its default values (`n_neighbors=3`) and `KMeans` only makes sense with `n_clusters=2` for this particular problem as it is a binary classification task. However, 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. The most tuned parameter in our dataset is for `SelectKBest` in which we try to determine the most relevant features for our model. Additionally, an improperly tuned model may highly over- or under-fit the particular dataset. 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.