1. *Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those? [relevant rubric items: "data exploration", "outlier investigation"]*

The goal of the project is to explore the Enron Corpus for useful clues that might characterize POIs (persons of interest) in the Enron fraud investigation. The Enron Corpus is a rich collection of financial information and emails exchanged between Enron employees leading up to Enron's collapse. My goal was to construct a machine learning classifier that leverages some of the data to identify POIs.

The dataset contains information describing 146 individuals, including 18 labeled POIs, and 128 non-POIs. The features in the dataset include things like an individual's salary, and the number of emails they sent and received. After going through the feature selection process (more on this later), I ended up with 5 features for my classifier to consider.

One of my first pieces of analysis of the data was to search for outliers. By comparing the *salary* and *bonus* features in a scatter plot, I identified one obvious outlier with a much higher salary and bonus than the others. Consulting the .pdf spreadsheet from which the data was collected, I realized the TOTAL row had been included in the set. Additionally, the penultimate row of the spreadsheet represented "THE TRAVEL AGENCY IN THE PARK," which I do not want to consider while classifying POIs. I removed these outliers.

2. *What features did you end up using in your POI identifier, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not? As part of the assignment, you should attempt to engineer your own feature that does not come ready-made in the dataset -- explain what feature you tried to make, and the rationale behind it. (You do not necessarily have to use it in the final analysis, only engineer and test it.) In your feature selection step, if you used an algorithm like a decision tree, please also give the feature importances of the features that you use, and if you used an automated feature selection function like SelectKBest, please report the feature scores and reasons for your choice of parameter values. [relevant rubric items: "create new features", "intelligently select features", "properly scale features"]*

My first step was to apply univariate feature analysis to score each of the features in the original dataset. I used sklearn's SelectKBest with a K representing all my numeric features, and the f_classif scoring function. This function is the default, and is suitable for our goal of classifying POIs.

Feature	Score
exercised_stock_options	25.09754153
total_stock_value	24.46765405
bonus	21.06000171
salary	18.57570327
deferred_income	11.59554766
long_term_incentive	10.07245453
restricted_stock	9.34670079
total_payments	8.86672154
shared_receipt_with_poi	8.74648553
loan_advances	7.2427304
expenses	6.23420114
from_poi_to_this_person	5.34494152
other	4.20497086
from_this_person_to_poi	2.42650813
director_fees	2.10765594
to_messages	1.69882435
deferral_payments	0.21705893
from_messages	0.1641645
restricted_stock_deferred	0.06498431

Table 1: Original features (with feature scores)

The top four features contain some redundancy. I created a new feature, *total_compensation*, representing the sum of *bonus* and *salary*, in order to (slightly) reduce the dimensional workload of the classifier algorithms. This feature earned a higher score than either of its constituent features:

Feature	Score
total_compensation	22.8879507

Additionally, this process generally ranked email features below financial features. In order to preserve a concise representation of the email data, I engineered a pair of new features, *fraction_to_poi* and *fraction_from_poi*, representing the fraction of an individual's to/from emails that were sent to/from a POI. The reasoning is that an individual who is a POI might send or receive more emails to or from other POIs. The feature is designed to capture those relationships compared to the total volume of email an individual may participate in.

Feature	Score
fraction_to_poi	4.16908382
fraction_from_poi	5.20965022

These features did not receive particularly high scores. However, I did choose to test algorithm performance with them in my feature list, so that the email data might influence the classification decision. Later on, when I was honing in on my classifier algorithm, I compared the performance of my top algorithms on both the feature set including my engineered features, and the feature set with those two features omitted. In most cases (and my final case), including the engineered features improved classifier performance.

The financial features in the dataset are represented uniformly in USD amounts. Since most of those numbers are huge, the email data (in the case of my engineered features, fractions between 0 and 1) will be dominated by the former when used together in (for example) an SVM or KNN classifier. Therefore, the feature set needed to be scaled. I used sklearn's MinMaxScaler to scale the selected feature set such that all the features were in the range [0,1].

I used sklearn's GridSearchCV in attempt to optimize the "k" hyper parameter of SelectKBest, with disappointing results. This method is designed to compare the performance of SelectKBest with all different values of "k"; however, it is limited to only contiguous selections of the top scoring features—it would not be able to freely select from all permutations of my feature set. Against my best-performing classifier (KNN), the SelectKBest selector was only able to achieve precision of 0.21 and recall of 0.19—with "k"=1, representing just *exercised_stock_options* as a feature set. I decided to do some manual tuning of 'k' along with comparisons with my engineered features:

Approach	**# Features**	**Feature Set**	**Accuracy**	**Precision**	**Recall**
GridSeachCV with SelectKBest	1	['exercised_stock_options']	0.79787	0.2146	0.194
SelectKBest	2	['exercised_stock_options', 'total_stock_value',]	0.86808	0.70622	0.244
SelectKBest	3	['exercised_stock_options', 'total_stock_value', 'total_compensation']	0.87954	0.76528	0.313
SelectKBest	5	['exercised_stock_options', 'total_stock_value', 'total_compensation', 'deferred_income', 'long_term_incentive']	0.87879	0.75635	0.2235
SelectKBest + Engineered Features	5	['poi', 'total_stock_value', 'exercised_stock_options', 'fraction_to_poi', 'fraction_from_poi', 'total_compensation'	**0.88964**	**0.76733**	**0.3265**

My selected feature set finally included *exercised_stock_options, total_stock_value, total_compensation, fraction_to_poi,* and *fraction_from_poi.*

3. *What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms? [relevant rubric item: "pick an algorithm"]*

I ended up using a K-Nearest Neighbors classifier. I also compared against a Gaussian Naïve Bayes classifier, and a Linear SVC. The Linear SVC had a substantial advantage in recall over the other two; however, due to its extremely low precision (many false positives), I reasoned that its high recall might have accountable to chance, compared to the others'.

The KNN and the NB were fairly similar in accuracy, with the KNN doing moderately better in precision, and slightly worse in recall. Due to the extremely small number of POIs in the dataset, I determined precision and recall to be more important than accuracy for evaluating my classifier (more on metrics later). I decided to accept the KNN's performance and attempt to improve its recall through tuning.

4. *What does it mean to tune the parameters of an algorithm, and what can happen if you don't do this well? How did you tune the parameters of your particular algorithm? What parameters did you tune? (Some algorithms do not have parameters that you need to tune -- if this is the case for the one you picked, identify and briefly explain how you would have done it for the model that was not your final choice or a different model that does utilize parameter tuning, e.g. a decision tree classifier). [relevant rubric items: "discuss parameter tuning", "tune the algorithm"]*

Parameter tuning pertains to the systematic adjustment of parameters that are used by a given machine learning algorithm to make calculations (in our case, calculations underlying classification decisions). Utilizing poor parameters, a machine learning algorithm's performance can suffer dramatically. Although optimizing parameters for a given algorithm can itself be automated, for my purposes I opted to tune the parameters by hand. This meant choosing a variety of values for each parameter and comparing the results.

I started by tuning the algorithm used to identify a point's nearest neighbors. Since this dataset is rather small and unlikely to overwhelm my CPU, I opted for the brute force approach (as opposed to a tree walk). However, this did not result in any change in performance. The reason is explained in the sklearn docs: "Note: fitting on sparse input will override the setting of this parameter, using brute force." The KNN was already using brute force for my data. As a result, the *leaf_size* parameter did not need to be tuned (this parameter is used for tree-based calculations). Nor did *n_jobs*, which would help multithread a computationally heavier problem.

I then moved to tuning the K parameter (the number of nearest neighbors contributing to the classification decision). Although reducing the value from the default 5 to 3 cost accuracy and precision, it did improve recall, which I had determined to be one of my most important performance metrics.

Next, I investigated the p parameter. This parameter represents the distance metric that is used to measure a point's proximity to its neighbors. Although the default value of 2 represents Euclidean distance, I decided to try a value of 1 for Manhattan distance. This change resulted in important increase in recall.

Finally, I decided to re-test the larger K=5 along with p=1. This was my final configuration.

Algorithm	K	p	Accuracy	Precision	Recall
auto	5	2	0.87464	0.63873	0.282
brute	5	2	0.87464	0.63873	0.282
brute	3	2	0.87371	0.61934	0.301
brute	3	1	0.874	0.61238	0.3215
brute	5	1	**0.88964**	**0.76733**	**0.3265**

Table 2: Parameter tuning performance impacts

The Linear SVC, on the other hand, would have needed to be tuned in terms of a different set of parameters: most notably gamma, the relative weight that points in the SVC have compared to their proximity to the decision boundary. In this particular SVC case, the linear kernel parameter is built in, so I would not need to tune that. It might also be worth investigating alternative kernels for a general SVC.

5. *What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis? [relevant rubric items: "discuss validation", "validation strategy"]*

When we create a supervised learning algorithm, we generally divide our dataset into a training set and a testing set. The model is fit to the training set, and we can measure its performance on the testing set to get a sense for how well we have done. However, in doing this we risk overfitting to the particular training set we have chosen.

Validation is the process by which we attempt to ensure that our classifier would work well when applied to additional data that was not present in our original training/testing dataset. We want to be sure that our model will work just as well for any sample of the same statistical population from which our training/testing dataset was sampled.

Using tester.py, sklearn's cross_validation.StratifiedShuffleSplit is invoked to perform k-fold cross validation (k=1000). It works by repeatedly and randomly dividing the dataset into ad hoc training and testing groups, and measuring the performance of the classifier on each iteration. The validation process averages the performances of each of these iterations to estimate the generalizable performance of the system—the performance we would expect to see from the system in practice.

6. *Give at least 2 evaluation metrics and your average performance for each of them. Explain an interpretation of your metrics that says something human-understandable about your algorithm's performance. [relevant rubric item: "usage of evaluation metrics"]*

My top 2 evaluation metrics for the purposes of this project are precision and recall.

Precision represents the proportion of selected points that *should* have been selected, according to their labels. In our example, this is the proportion of true POIs among the group of points selected by the classifier as POIs. A precision of 1.0 means that the classifier *only* selected true POIs, whereas a precision of 0.5 means that half of the points selected as POIs were not POIs in reality. My classifier achieved a precision of 0.767, which means only about a quarter of the individuals we would interrogate as a result of their classification would have been have been bothered for no reason.

Recall represents the proportion of POIs that were successfully identified by the classifier compared to the total number of POIs in the dataset. In other words, a recall of 1.0 would result in all the criminals going to jail, while a recall of 0.5 would mean half of them would go free. Unfortunately, my classifier only catches about a third of the POIs in the dataset. However, due to the classifier's good recall, I believe this is a reasonable outcome—we won't have to bother too many innocent people to catch those POIs.