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”]

ANSWER:
The goal of this project was to identity persons of interest (poi) using the Enron data set which had a variety of financial variables such as exercised_stock_options, bonuses as well as other variables related to communications with persons of interest such as number of emails sent to and from persons of interest. Machine learning allows us to detect patterns in the structure of the data in order to predict whom we think are persons of interest (those whom committed fraud, of which there were 18 out of 146 data points) based on a values among a variety of financial and non-financial variables. There were obvious outliers in the data set that needed to be removed such as the "THE TRAVEL AGENCY IN THE PARK" which was not a person, and points such as "TOTAL" which represents the total value of each variable summed across all pois and non-pois. Other outliers were mainly attributed to Kenneth Lay, but as he was the key architect of the Enron fraud, super-high outliers in certain financial variables should come as no surprise. After removing the outliers, there were only 144 data points in the data set.

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”]

ANSWER:
The features: exercised_stock_options, prop_from_poi, and shared_receipt_with_poi, other, were the three that were chosen by using the features_importance_ parameter of the DecisionTreeClassifier. A overfit decision tree is fit with all the features from the features_list and then the importance of each feature is gauged into a list, whereby the top features were chosen based on an arbitrary threshold. In this case, the threshold of 0.05 on either the decision trees or random forests algorithm was chosen because after testing with different thresholds ranging from 0.05 to 0.12, this was found to give among the best results in each of the ML methods used. *Note that the threshold of 0.1 on k-nearest neighbors worked equally well, but was not chosen due to problems with reviewers generating similar results.*

The chart below shows the results of trying out different thresholds in order to capture a different number of features being included into each of three of the best performing algorithms below.

Decision Trees/ Random Forests - Results using different thresholds for feature importance

Threshold - Feature importance	Precision	Recall	Features
0.05	0.456	0.365	6
0.06	0.34	0.285	5
0.08	0.23	0.095	4
0.10	0.41	0.34	3
0.12	0.48	0.31	2

K-nearest neighbors

Threshold - Feature importance	Precision	Recall	Features
0.05	0.29	0.22	6
0.06	0.27	0.22	5
0.08	0.65	0.14	4
0.10	0.49	0.37	3
0.12	0.42	0.355	2

For the chosen threshold of 0.05 on decision trees/ random forests (which gave the best results overall), the variables chosen and their feature importance value is shown below in the chart. It can be shown that exercised_stock_options was the single most important indicator of being a poi, with the other five being less important.

Feature	Feature importance value
Exercised_stock_options	0.323
Prop_from_poi	0.133
Shared_receipt_with_poi	0.113
Other	0.09
Deferred_income	0.058
Long_term_incentive	0.054

Scaling was done for K-nearest neighbors and SVM because of the disproportionate effect of variables with larger values in biasing the results of these algorithms as these two methods are distance-based. Several features were engineered, such as "prop_from_poi" which calculates the proportion of inbox messages from a poi, "prop_to_poi", which calculates the proportion of sent messages to a poi, "total_comp_salary_ratio" which is the ratio of salary to total payments, and the "sent_received_ratio" which is the ratio between sent and received messages to and from pois. The first two features were chosen because it signifies the levels of interaction with pois relative to non-pois and in fact "prop_from_poi" was one of the predictor variables chosen based on a threshold of 0.1. The "total_comp_salary_ratio" variable was created because a high salary relative to total payments (overall compensation) might indicate that they are less likely to be pois and vice versa, while a higher sent_received_ratio might indicate higher likelihood to be a poi as they send more messages to pois than they receive (received messages could be mass communication messages).

Three algorithms performed fairly well given the nature of the data set, and these were decision trees, random forests and k-nearest neighbors. However, only decision trees/ random forests gave a recall and precision of 0.3 and above for both myself and the reviewers when run with the tester.py script.

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”]

ANSWER:
In the end, the final algorithm being used was the random forests algorithm (could have gone with decision trees as well as it gave the exact same results), as it gave consistently the recall and precision when run using the tester.py script. Although other algorithms such as k-nearest neighbors gave equally good results, but decision trees/ random forests was the only good result that was consistently above 0.3 in recall and precision. This algorithm was chosen and works well because of the fact that pois tend to be closer to other pois based on those variables chosen earlier, and so boundaries of different partitions of the data set are split in such a way that can pick up pois within a certain boundary (though the recall and precision was still not particularly high). Model performance did not differ substantially when compared to k-nearest neighbors as precision was actually slightly lower (though above 0.3) for decision trees/ random forests However, other ones such as SVM and logistic regression did not perform well at all based on these selected features, and marginal improvements even when tuned to have the best parameters meant that they were not included.

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”]

ANSWER:
Tuning the parameters of an algorithm is important in finding the best parameters possible to improve the performance of the algorithm and to potentially obtain better results than with default parameters, and if this is not done well then you could end up with worse results than without tuning. I tuned the parameters of my algorithm using the GridSearchCV function on a set of important parameters for that particular algorithm. The best_estimator_ was subsequently outputted as the best result from the GridSearchCV as the best estimate of the classifier. In the case of random forests, the chosen algorithm, the parameters tuned were max_depth (representing the maximum depth of the tree), min_samples_split, (the minimum number of samples required to split an internal node), min_weight_fraction_leaf (the minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node), and min_samples_leaf, (the minimum number of samples required to be at a leaf node) with a random_state being set so that the results can be reproduced.

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”]

ANSWER:
Validation is done by creating two separate training and test sets, whereby the data is trained by algorithms on the training data set and then the fitted parameters/estimators are then applied towards the test set in order to develop our predictions. This is done in order to ensure that your machine learning algorithm performs well and can be generalized to new data. One classic mistake is "overfitting", the model is trained and performs very well on the training dataset, but does much worse on the cross-validation and test datasets. In validation, I partitioned the data into the training set being 70% of the data set and the test set being 30% of the data set.

Further, aside from doing cross-validation, I also used the test_classifer function provided in order to gauge how well cross validation worked on each of the algorithms I chose to work with. The StratifiedShuffleSplit splits the data into stratified and randomized folds (1000 times) based on the split between training and test data (0.7 and 0.3 in this case), and allows to have a much better gauge of how well our classifier is at predicting the poi based on the three features over many partitions of the training and test data. This was an optional task that was done to further ensure that validation had been done correctly and consistently, but because this training/test set split is done 1000 times, the results were slightly worse than if it was done once. This was also done due to the class imbalance of poi versus non-poi data points in the data set as there were 18 for pois and 126 for non-pois, thus making the results more prone to fluctuation depending on partitions of the data is split into training and test sets.

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”]

The main evaluation metrics utilized were precision and recall. Precision gives us the ratio of true positives (employees predicted to be pois that are actually pois) to the records that are flagged/predicted as being pois. Recall would give us the ratio of true positives to the sum of true positives and false negatives, which is the sum of pois that were correctly identified and pois that were incorrectly classified as non-pois, which is thus the ratio of pois that were correctly identified to all records that actually are and should have been identified as pois. Accuracy was not chosen because of the limited number of data points and unbalanced nature of the data set (far more non-pois than pois), thus we can have high accuracy and still never predict a poi correctly.

	Precision	Recall	Features
Decision Trees	0.41	0.34	3
Random Forests	0.41	0.34	3
K-nearest Neighbors	0.49	0.37	3

Using the tester.py script, the best performing algorithm was k-nearest neighbors with a precision of 0.53 and a recall of 0.31. However, better results were achieved with the default parameters for both decision trees and random forests (instead of tuning) for both precision and recall (although very slightly).