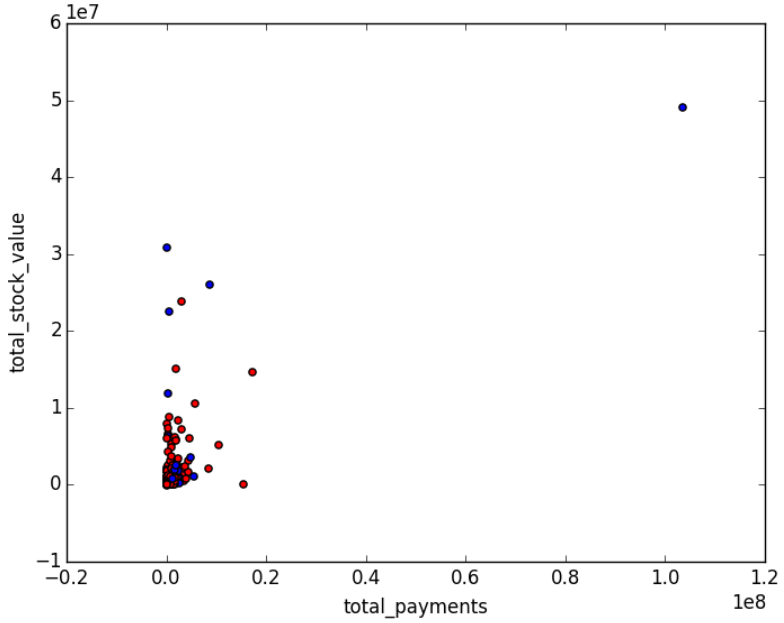
1. Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those?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

   The goal of this project is to build a classifier using machine learning that can accurately predict whether someone is a person of interest (POI) in the Enron bankruptcy case. The prediction will use certain features provided in the enron dataset, which includes information about employees, such as financial info (salaries, bonuses, stock options, etc.) and email info (how many emails they sent, how many the received from a poi, etc.).

   A little exploratory data analysis showed that this dataset contains 145 data points, 18 of whish are POIs. Each data point has 20 features and a label as to whether or not they are a POI. All of the features have at least some missing values. For some of the features (loan_advances, deferral_payments), over half of the data points are missing that feature. Most seem to have about 1/3 (40-60 data points) missing. I ended up only using 4 features in my model, and none had more than 60 missing values.

   There was an outlier in the original dataset with an order of magnitude higher bonus and salary, which was actually the total bonus and salary of all the employees (key: “TOTAL”), so I removed it from the dataset. After that, I did some exploratory data analysis and visualized the features using scatter plots to look for more outliers. When there did appear to be an outlier, I went back to the financial info pdf to check whether it was a real data point or a data entry error. For example, I made a scatter plot of total payments vs. total stock value which is shown below.



There is clearly one outlier on the top right. Looking back at the financial data info, I found this to be Kenneth Lay, and the data point agreed with the financial record, so I did not remove it. Some of the potential outliers I investigated were: Kenneth Lay's total payments, loan advances, total stock value, exercised stock options, and other payments; Mark A Frevert's deferral payments; Sanjay Bhatnagar's restricted stock deferred; Amanda K Martin's long term incentive; John J Lovarato's from poi messages; Wincenty J Kaminsky's from messages (Each of these was found in the same method described above). All of the financial data was as represented in the pdf so I left it in the dataset. I had no way to check the accuracy of any message count outliers, but as they were not orders of magnitude above other message counts, I thought there was a good chance they were accurate, so I left them in the dataset as well.

2. What features did you end up using in your POI identifier, and what selection process did you use to pick them? 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", "properly scale features", "intelligently select feature"]

   I ran an initial decision tree classifier with all of the features possible (except email address). Then I removed all features with <0.05 feature importance. Then I reran with the remaining features and removed features with <0.1 feature importance to end up with only five features: bonus, expenses, exercised stock options, other, and from_this_person_to_poi. This process is summarized in the table below.

	First iteration	Second iteration	Third iteration
Salary	.024	-	-
Deferral_payments	.016	-	-
Total_payments	.064	.092	-
Loan_advances	.000	-	-
Bonus	.149	.174	.212
Restricted_stock_deferred	.001	-	-
Deferred_income	.055	.080	-
Total_stock_value	.040	-	-
Expenses	.114	.167	.240
Exercised_stock_options	.151	.182	.228
Other	.105	.126	.192
Long_term_incentive	.029	-	-
Restricted_stock	.057	.078	-
Director_fees	.000	-	-
To_messages	.023	-	-
From_poi_to_this_person	.028	-	-
From_messages	.042	-	-
From_this_person_to_poi	.058	.101	.128
Shared_receipt_with_poi	.042	-	-
Precision	.226	.246	.350
Recall	.216	.228	.324

Once I was down to five features, I tried to see if reducing further would increase performance. I tried each possible combination of four features, shown below.

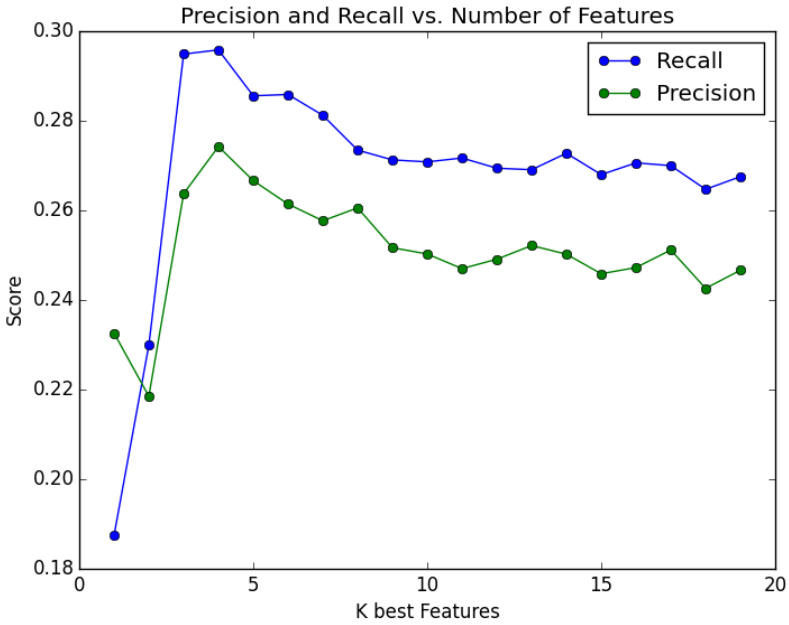
	First combo	Second combo	Third combo	Fourth combo	Fifth combo
Bonus	.255	.224	.296	.273	-
Expenses	.253	.336	.304	-	.372
Exercised_stock_options	.243	.311	-	.281	.250
Other	.249	-	.315	.280	.186
From_this_person_to_poi	-	.128	.085	.165	.192
Precsion	.384	.375	.357	.273	.544
Recall	.364	.350	.336	.254	.498

Despite 'bonus' usually having a high feature importance and 'from_this_person_to_poi' usually having a low one, the best performance resulted from leaving out bonus and sticking with the other four features (fifth combo above). A few smaller feature combinations were tried but the precision and recall always dropped compared to the best four-feature combination.

I also tried selecting features using SelectKBest, which found salary, bonus, total_stock_value and exercised_stock_options as the highest scoring features. The select K best scores are shown in the table below.

Feature	Score	Feature	Score	Feature	Score	Feature	Score
Salary	18.57	Res stock deferred	0.06	Other	4.2	From poi	5.34
Deferral payments	0.22	Deferred income	11.59	Long term incentive	10.07	From messages	0.16
Total payments	8.87	Total stock val	24.46	Res stock	9.35	To poi	2.42
Loan advances	7.24	Expenses	6.23	Director fees	2.11	Shared w/ pois	8.75
bonus	21.06	Exercised stock opt	25.10	To messages	1.70	-	-

I ran 1000 training/test splits and fit a model with each of 1 through 19 features using the SelectKBest function. I found the average recall and precision for all models of 1 feature, all models of 2 features, etc. It is clear that selectKbest is in agreement that about 4 features is best for performance, but the model with the four top features (by selectKBest) had worse performance than my previous best. I think this may be because of the scoring mechanism of the SelectKBest algorithm, but I'm not entirely sure. The feature importances for this model were: salary = 0.16929751, bonus = 0.30991686, total_stock_value = 0.20666879, exercised_stock_options = 0.31411683. The precision was 0.33 and the recall was 0.34. I ended up sticking with the four features I found using feature importances. The results of the average precision and recall for all selectKbest attempts are plotted below.



Did you have to do any scaling? Why or why not?

I did not use any feature scaling because scaling should not affect decision trees.

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.)

I tried engineering a few features that I thought might contain more information than the given features. One was the percent of total stock value that was exercised stock, thinking that maybe POIs were more likely to exercise their stock options. I also tried percent of payments that came from the 'other' category, thinking that maybe POIs would try to hide their payments in that non-specific, 'miscellaneous' category. Another feature I tried was bonus to salary ratio, thinking that maybe POIs would take an unusually large percent of their salary as a bonus. Similarly, I also tried stocks to payments ratio, thinking maybe POIs would try to 'hide' more money in stocks. The final feature I created was 'MOIs' or messages of interest which I defined as messages to, from, or shared receipt with POIs over total messages (to and from), just as a overall way to gauge what proportion of your email correspondence had some connection to a POI. None of the features I created scored well in feature importance, so none made it into the final model.

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 choosing a decision tree classifier after trying svm, naïve bayes, and adaboost as well. I was actually able to get higher accuracy with some of the other algorithms, but (as I discuss further in the evaluation section) for this application I

don't think accuracy is the best metric of success. I picked a decision tree classifier because I was able to achieve recall of ~0.5, which was often 2 or 3 times the recall I could get out of naïve bayes or adaboost. I also was able to get a precision of ~0.54 which was also considerably higher than the other algorithms. I wasn't able to get svm to identify *any* POIs even after reducing the penalty parameter to a very low value.

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? (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 item: "tune the algorithm"]

   Algorithms have various parameters that affect their performance, such as how much to penalize an error or how deep a tree is allowed to go. If you don't tune the parameters you risk decreasing the model performance, increasing the computation time, and overfitting the training data.

   I tried a few methods to tune my algorithm (decision tree). I first tried to use GridSearchCV to tune things like min_samples_leaf and max_depth. I thought it would find that an intermediate number of samples in a leaf and a not too deep tree would do a better job not overfitting the training data. That turned out to be true when the scoring metric was accuracy (default). But what I saw was the parameters that resulted often had a detrimental affect on precision and recall. Digging a little deeper, I realized that POIs only make up ~12% of the data. This means just guessing non-POI for everyone gives pretty good accuracy, and accuracy is therefore not a great metric for performance of the decision tree (in this specific application). I found that a closely fit tree (low samples per leaf and deep trees) actually gives better performance using precision, recall, f1 and f2 as metrics. I ended up choosing a min_samples_leaf of 1 and no max_depth, which increased the roc and average precision scores (discussed below) as well as the metrics mentioned above (but *not* accuracy).

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 item: "validation strategy"]

   Validation is the idea of holding back a section of your data to test your model on after training the model on the remaining data. This way, if your model overfits your training data, you will get poor scores (accuracy, precision, etc) when you try to apply it to the validation/test data. I used the stratified shuffle split cross validation that was already in the tester script as the ultimate way to validate my analysis. This was picked over other cross validation strategies because it has the advantages of shuffling the data points and picking random folds (good for relatively small datasets) and stratifying the folds so that they have approximately equal proportions of classes (good for the low percent of POIs).

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"]

For this problem, accuracy score is not really a good evaluation metric, because there are so few POIs compared to non-POIs, that just predicting non-POI for everyone gives us a pretty high accuracy (~87%). Precision and recall are better metrics. Precision tells out of the people our machine learning algorithm predicted to be POIs, what percent actually were POIs (~50%). Recall is the percent of actual POIs that we were able to pick out with our algorithm (~54%). Another interesting metrics is area under the ROC curve, which is a curve plotting true positive rate vs false positive rate as our classification threshold is changed (the classifier initially yields a probability which is then compared to some threshold (0.5 as default) to determine which class the data point belongs to). The area under this curve tells us, if we were presented with one POI and one non-POI, what percent of the time our classifier would pick the POI correctly, and my model gave a average score of ~71%. Similarly, the precision and recall can also change depending on what value we pick as our probability threshold. The average precision score measures the area under the precision vs. recall curve and may be more appropriate for this example since we have a large imbalance in classes (a relatively small amount of POIs). My model achieved an average precision score of ~61%. However, in this specific model, because the leafs were allowed to go down to one data point, the threshold probability is somewhat meaningless. Therefore ROC AUC and average precision may not be appropriate metrics, so I focused more on recall and precision.

In this application, we are probably willing to sacrifice some in our false positive rate, since we presumably would be using this classifier only to pick out potential POIs and then investigating further from there, and I think it's better to have too many suspects than let some POIs go free (which translates to high recall). However if we were using this to actually determine guilt than maybe a low false positive rate would be preferred over a high precision.

Website resources:
for ideas on which classifier to choose:
http://scikit-learn.org/stable/tutorial/machine_learning_map/

for color in matplotlib:
http://stackoverflow.com/questions/19139621/python-matplotlib-scatter-plot-changing-colour-of-data-points-based-on-given-c

plotting with matplotlib:
https://bespokeblog.wordpress.com/2011/07/07/basic-data-plotting-with-matplotlib-part-2-lines-points-formatting/

auc info:
https://www.kaggle.com/forums/f/15/kaggle-forum/t/7517/precision-recall-auc-vs-roc-auc-for-class-imbalance-problems/41179

great article on feature engineering:

http://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/

I used a lot of the sklearn documentation:
http://scikit-learn.org/

I hereby confirm that this submission is my work. I have cited above the origins of any parts of the submission that were taken from Websites, books, forums, blog posts, github repositories, etc.