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 attempt to identify accurately a person of interest in the Enron Fraud investigation. Utilizing machine learning gives us a way, when there is not a clear pattern or answer, to determine the pattern or answer efficiently.

   There were some outliers in the data, some invalid sets of data, and some “NaN”. I removed some data points I deemed “invalid” (TOTAL, THE TRAVEL AGENCY IN THE PARK).
   When I attempted to remove a certain percentage of values, I found my model performed worse than if I left them in. I did end up cleaning up the “NaN” and just changing their values to 0.

   In the end, I had the following data that I would evaluate:

   - 144 total records after TOTAL, THE TRAVEL AGENCY IN THE PARK items were removed as they had many missing values. After the code was completed and I was evaluating additional output, I noticed that LOCKHART, EUGENE E. was also a value with 0 values (I did not end up removing him).
   - 25 features total, 22 initial features before adding new features, 23 that would be used (2 features not used in the analysis were the names of the POI and their email address).
   - 7 features used for the classifier
   - 18 were identified as POI, 126 were not POI.

2. What features did you end up using in your POI identifier, and what selection process did you use to pick them?

   I ended up using the following feature list:
   features_list = ['poi','salary','exercised_stock_options','total_stock_value','bonus', 'transformed_salary','composite_poi_email_data']

   I used the features above after running some correlation against the data, and iteratively testing as I added/modified features. Though the correlation was not strong on any of the features, I found a balance that got me past the “.3” threshold on precision and recall for the project. I used correlation as this is indirectly a regression problem and I might be able to use that concept to point me in the direction of passing the threshold. During this manual process, I would add a feature after reviewing top correlation coefficients related to POI, then test, and repeat until I hit my target for precision and recall (.3 and .3 respectively).
   Additionally, I did use SelectKBest feature selection to compare my correlation idea with a pattern discuss in the course. Here is the comparison, both using Naive Bayes:

Feature Selector	features selected	Precision	Recall
SelectKBest (I took the top 6)	features_list =['poi','expenses', 'director_fees', 'deferred_income', 'exercised_stock_options', 'total_payments','bonus']	0.32052	0.47950
Manual coefficient selection	features_list = ['poi','salary','exercised_stock_options', 'total_stock_value','bonus', 'transformed_salary','composite_poi_email_data',]	0.50038	0.32750

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”, “properly scale features”, “intelligently select feature”]

I ended up doing a couple feature scales and modifications. I did MinMaxScaler on the “transformed_salary” and did a composite (combined) feature for poi email data (added the 'from_this_person_to_poi'] + 'from_poi_to_this_person’. I created these features to attempt to lift the either precision and/or recall. While the new email features (composite_poi_email_data) did not provide any value, the “transformed_salary” in conjunction with the original “salary” field did add ~3.5% increase to the precision (as described below):
Took my correlation data and transformed features and received:

GaussianNB(priors=None)
Accuracy: 0.85721 Precision: 0.50038 Recall: 0.32750 F1: 0.39589 F2: 0.35181
Total predictions: 14000 True positives: 655 False positives: 654 False negatives: 1345 True negatives: 11346

Removed the "salary" and just had the transformed salary and received:
features_list = ['poi','exercised_stock_options','total_stock_value','bonus', 'transformed_salary']
GaussianNB(priors=None)
Accuracy: 0.83862 Precision: 0.46475 Recall: 0.32300 F1: 0.38112 F2: 0.34398
Total predictions: 13000 True positives: 646 False positives: 744 False negatives: 1354 True negatives: 10256

So the transformed salary in conjunction with the salary improved the precision by ~3.5%

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 the initial Naïve Bayes. I tried using Decision Trees at one point, and really tried to use RandomForestClassification. I found that I got the best performance using Naïve Bayes.
   Below are the output results of the two. As you will see below, precision was close between the two, but recall for RandomForestClassification did meet the minimum of 0.3

algorithm	precision	recall
RandomForestClassification	0.51421	.19000
GaussianNB	0.50038	0.32750

4. What does it mean to tune the parameters of an algorithm, and what can happen if you don’t do this well?

   Parameter tuning is specifying different parameter options in an algorithm. The outcome of poor parameter tuning can be unexpected results or poor performance in your evaluation metrics. Additionally, you can also over-fit your algorithm so that it may appear to perform well, but when put towards additional data performs poorly.

   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”]

   When I was working with the RandomForestClassifier, I used GridSearchCV to automatically select the “best” combination. Below is the parameter grid I used and the best parameter fit that GridSearchCV chose.
   clf = RandomForestClassifier(n_jobs=10,max_features= 'sqrt' ,n_estimators=50,

```
max_depth=None)
    param_grid = {
        'n_estimators': [100, 200,300],
        'max_features': ['auto', 'sqrt', 'log2'],
        'max_depth': [5,10,15,]
    }

    CV_rfc = GridSearchCV(estimator=clf, param_grid=param_grid, cv= 5)
    CV_rfc.fit(features, labels)
    print CV_rfc.best_params_
    clf = RandomForestClassifier(max_features = 'auto', n_estimators = 100, max_depth = 5)
```

Interestingly, I found that it did not perform as well as I thought it might when I used GridSearchCV versus the default parameters.
I ended up using Naïve Bayes, which does not have any appropriate parameters to tune in this case.

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 assessing our model against testing data and reviewing the performance. A classic mistake for not performing validation appropriately is over-fitting, which essentially means that we perform well against our testing data, but then perform poorly against our training data.
   The way that the tester.py (which is what I used for validation/evaluation) performed validation is by using cross-validation, specifically StratifiedShuffleSplit.
   Cross Validation is a validation technique for evaluating how a model will perform against a more generalized set of data.
   StratifiedShuffleSplit specifically attempts to overcome the fact that the number of positive/negative (poi) is skewed towards one way, and ensure that during the validation of your data, relative frequency of the positive/negative are preserved in each testing and training (see: http://scikit-learn.org/stable/modules/cross_validation.html - Section 3.1.4).

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"]

   I ended up using precision and recall from the tester.py to evaluation the performance of my algorithms. At one point I tried using Average Precision Recall, but it was very touchy for splitting the data directly in half and ultimately I believe the precision and recall scores gives an appropriate evaluation metrics.

   For the precision metric, I was hitting 0.50038 with Naïve Bayes. What this means is that for all of the values that the classifier marked as positive (1 in this case), ~50% were actually positive.
   For the recall metric, I was hitting 0.32750 with Naïve Bayes. What this means is that of all of the positive measures that were there, my classifier appropriately classified ~32.8%.

To say it another way, my classifier was more likely to classify someone who was not POI than to actually find all of the POI's.