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: This project is an Enron dataset with several financial features and email features about people working in Enron. Example of financial features includes bonus, salary, total payments, expenses, stock payments etc. Email features include total from messages, to messages, messages sent and received to POI etc. The goal of this project is to use the above features to create a classifier for POI. The dataset is organized as a dictionary where each key is the person and the value for that key is another dictionary of features and their values. In my opinion the organization of this dataset in this dictionary format was one of the most challenging aspects of this project. The final tester.py code assumes a dictionary as an input so all transformations had to be done on dictionary.

This dataset has 146 people and for each person there are 20 features and one target POI – yes or no. There are 18 POIs in the dataset. Overall the quality of data is not very good. I found that three of the keys (persons) were bogus – Total (not a person), 'THE TRAVEL AGENCY IN THE PARK’ (again not a person) and 'LOCKHART EUGENE E' (all features were zero or Nan). I started by removing these keys. Also the dataset has a lot of features that are nan. My next step was to replace nan with zero. I did this because I assume Nan in most cases means that the value is zero. Example deferred income was nan for a lot of people.

I used matplotlib to plot the distribution of select features and found that the data has outliers – values that are either very high or low. I did not feel comfortable removing any more keys (people) with outliers since the dataset is already fairly small. So what I did was floor or cap outliers. Meaning if say the value of salary was greater than mean + standard deviation of salary on the entire set then salary for that person was capped at the value. This is an important step as the classifier might be sensitive to extreme values.

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

Answer: I spent a lot of my time in this project at this step. I did the following steps:

- Started by selecting most of the financial features
- For email features, I created 3 new features:

```
# 1. perc_emails_to_poi: 'from_this_person_to_poi'/ 'to_messages'
# 2: perc_emails_from_poi : 'from_poi_to_this_person'/'from_messages'
# 3: perc_emails_shared_poi: 'shared_receipt_with_poi'/'from_messages'
```

I did this to get the most value out of email features. I used matplotlib to look at the mean for poi and non poi for the above created features and saw that the features predicted poi or not. Hence I felt comfortable using the above features. The poi_id.py has the matplotlib code for this visualization.

- Created a feature list consisting of financial features and the above engineered email features
- Replace all NANs with zero and floored and capped outliers
- Transformation – I found that the features were not normally distributed but skewed. I did a log transformation so that the features are normally distributed
- Scaling of features – I scaled all features to have values b/w 0 and 1 by looking at the minimum and maximum value of that feature. This is an important step because some classifiers are sensitive to the scale of features and work better when the data is normalized in this fashion

As I mentioned that one of the main challenges with this dataset is that it is a dictionary of dictionary and this format needs to be preserved for tester.py. So for all the above steps, I created a function in my poi_id.py that takes the dictionary and for each feature, appends all the values to a list. This list is converted to a numpy array to do all the transformations. Finally the numpy array is converted back to a list and the new values are updated in the dictionary

I used matplotlib to visualize the mean of feature for poi and non-poi and confirmed that most selected features were valuable.

I first tuned all my algorithms on all the above features. However as a second step, I implemented PCA to identify principal components which explain most of the variance in the data. I found that the fist 5 components explained 85%+ variance in the data. As a second step, selected algorithms were tuned on Principle components.

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: I started by selecting the following 4 algorithms:

- Guassian Naïve Bayes
- Support Vector Machines
- Decision Tree Classifier
- K neighbors classifier

To validate the performance of these algorithms I used the code in tester.py to look at F1-score (combination of precision and recall), precision score, recall score and accuracy score. I decided to use the scores calculated in tester.py and not those from sklearn metrics as I found quite a bit of discrepancy in results and I wanted to evaluated myself against the custom code created for this project.

I found that SVM got a divide by zero error and precision and recall score could not be defined. So I proceeded with the other three algorithms. When I implemented these algorithms on PCA components, I found quite a bit of variance in performance. All had accuracy >80% but precision and recall varied quite a bit. For 5 principle components;

- Guassian NB; Precision: 0.31550 Recall: 0.08550
- Decision Tree: Precision: 0.43251 Recall: 0.37650
- K neighbors: Precision: 0.10801 Recall: 0.04250

The best performance was from Decision Tree algorithm by far

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”]

Answer: All algorithms have default parameters assumed when we used them from their sklearn library. Tuning an algorithm is the process of updating these parameters to further improve the performance of the algorithm. For my project here I chose two algorithms to further tune: 1. Decision Tree Classifier and 2. Guassian NB. I chose decision tree as it had the best performance and I chose Guassian NB as I saw that the performance was sensitive to no of principal components chosen and I wanted to experiment with that.

I struggled to use gridsearch CV to tune because for evaluating the performance I wanted to use the F1 score as defined by the custom code in tester.py. The F1 score as defined in sklearn metrics had different value from the F1 score in custom code.

So I iterated through the different parameters manually. For decision tree I optimized no of principal components and max_depth of the tree. For Guassian Naïve Bayes I could only optimize the no of principal components as the algorithm doesn’t have any parameters that can be tuned.

The results of the tuning were:

- Decision tree with optimal performance had 5 principal components and max depth = 8
  Accuracy: 0.85027 Precision: 0.42773 Recall: 0.36400
- Guassian Nb wth optimal performance had 8 principal components
  Accuracy: 0.83467 Precision: 0.38439 Recall: 0.39900

It was interesting to note that the performance of decision tree did not improve materially during tuning. However it served to verify that our original chosen parameters were the best.

The final classifier chosen was Decision Tree as it had better accuracy and precision compared to Guassian NB. Also I find that decision tree can be a more intuitive algorithm to understand and explain to others.

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 process of assessing the performance of an algorithm on a sample that is not used to train the algorithm. This is important as the model can overfit on the training sample and any performance metrics may not accurately reflect true learning on the model. To overcome this, there is a test set that is created that is not used during the training of the model and model performance on test set is used to evaluate the best algorithm.

In our case tester.py test_classifier function uses StratifiedShuffleSplit to make folds of the data. The dataset is trained on n-1 folds and performance is evaluated on nth fold. And this process repeats again and again.

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 two most critical evaluation metrics for this project are precision and recall. For the chosen algorithm, performance is

Precision: 0.42773 Recall: 0.36400

The precision score measures True Positive / (True Positive + False Positive)

While the recall score measures True Positive / (True Positive + False Negative)

High precision score implies that the algorithm is detecting more true positives (POI) with minimal false positives (predicting POI when someone is not POI). It is important to have high precision score because false positives is bad – accusing someone of being POI when they are not

High recall score implies that an algorithm is detecting more true positives with minimal false negatives (predicting someone is not POI when they are). To do justice we want to be able to accurately identify everyone who perpetrated fraud.

However I would argue that given 2 algorithms I would lean towards choosing one with higher precision score in this case even if recall is a bit worse as I think false positives have a higher cost here.