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?

- The goal for this project was to generate a machine learning algorithm that could aid in the identification or *person's of interest* and the identification for *non-person's of interest.* Using both financial and available email data from the Enron Corporate Scandal. During some background investigation, I discovered that Enron executives were involved in what is known as mark-to-market accounting. This is when you declare an asset as profitable once it is built before it is actually profitable.
- Generally when investigating corporate fraud you are looking at deep rooted corruption and based on how the nature of many corporations run, checks and balances are implemented at very high levels of the company. This is important because as we find in the financial data: salary, bonus and stock options tend to run quite high with executives. It is also common place for those who are horizontal in rank often communicate in the same circles (it is not common for an every day worker to have contact with c-level executives) by analyzing email data, it may also be possible to shed insight on who was involved with the scandal.
- Because executives typically make a lot of money, outliers were almost exclusively what we were looking for in our investigation. Rather than Winsorizing the data, I looked to remove erroneous records that contained all NaN values. There were only a couple of these that I found through debugging and they were both removed. The two values removed were "Total" and "The Travel Agency in the Park" as all other rows were people, these two records did not seem to fit in to the data and I removed them.
- When taking a look at the shape of the data after the removal of the two outliers we have **144 observations with 21 features**.

- It should also be noted that we have a total of **18 Persons of Interest** 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”]

- I wound up using SelectKBest to narrow down my feature list from an original 21 (all but email address to keep my values numeric) to a subset of 12. During the tuning process, I treated the top attributes as a tunable parameter in my algorithm, I experimented with 5, 10 and 15 attributes; however, 12 gave me the best results. I decided to go with the approach of using a feature selection mechanism because I was not familiar with the dataset enough to know all of the ins and outs of what people were identified as POI’s aside from the fact that there were a large number of executives involved.
- The scoring from the different features were as follows:
  - '**deferral_payments** 71.4929886655’,
  - '**total_payments** 59.3250716109’,
  - '**loan_advances** 166.222170058’,
  - '**restricted_stock_deferred** 91.0010724601’,
  - '**deferred_income** 188.703723602’,
  - '**total_stock_value** 45.8373211569’,
  - '**expenses** 195.826430776’,

- '**other** 72.8488512742',
- '**long_term_incentive** 69.6442695656',
- 'from_this_person_to_poi 68.3628654715',
- 'shared_receipt_with_poi 79.328213342'

- I created one new feature "Bonus_Percentage_Salary" with the concept of identifying the highly paid employees at Enron. I created this column by dividing the salary by the bonus. When testing at the end of my script, there was a negligible impact to the performance of the algorithm scoring of precision and recall was between .01 and .03 after the addition of the added feature.
- When you are working with numeric data, machine learning algorithms can sometimes struggle when different parameters are used at different scales of magnitude. Because of this, I implemented a standard scaler as part of my preprocessing implementation. By getting all of my numbers on an equal scale, I was able to maintain my upper limit outliers and use my numeric features.

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"]

- The algorithm selection process was actually a really intriguing and frustrating part of this project. Since I started with a Gaussian Naive Bayes already in code, I used that as a first test to get my code working properly. One interesting thing I found on the NB algorithm is that even without tuning, it seemed to perform relatively well with recall.
- In addition to the Naive Bayes algorithm, I ran a Support Vector Machine which would perform either very good or very bad depending on the random set generated. I also tried an AdaBoost algorithm which was my lead pick for a long time.
- The AdaBoost algorithm showed a lot of promise and I experimented for a long time with the different hyper-parameters as well as the base classifier; however, I had trouble getting the precision and recall to the desired level. Using the sklearn precision and recall scoring, it performed

very well; however, with the custom tester class, it was very slow to complete and did not yield as good of results.

- In the end I decided to go with The RandomForestClassifier, this algorithm gave me the most consistent results and the tuning performed as expected, which was an issue I had with the AdaBoost algorithm.

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"]

- Tuning the hyper-parameters of the algorithm involves the actual parameters passed into (or adjusted after instantiation) the estimator object. They vary with the estimator, so the parameters for an SVC may not be the same as the parameters for a Random Forest. Tuning this is a critical part of getting a good functioning algorithm and is done to adjust overfitting or under-fitting to your data.
- I worked on my Random Forest Classifier for a long time on this step and it is partially because I attempted this project once already - previous to developing the my_poi module that I used for the project submission. Originally I used GridSearchCV and RandomizedSearch functions to experiment with depth, number of leaves, number of estimators, etc. Then I would pass the best_estimator out to the tester class and would record the values that it went with for best performance. I was actually getting pretty good readings in my early tests and used those tuning parameters as a starting point for my final algorithm. Because the testing can take so long with the model_selection functions, when I was finishing up my algorithm, the tuning I did was by hand. I

had already started in a place that I knew from previous tests would work well and after that I was able to focus on small changes to the estimators and depth.

- During the tuning portion of the algorithm there was automatic tuning with GridSearchCV as well as manual tuning. Ultimately, I decided between 3 hand picked features to decide the best performance of the algorithm; they were as follows:
    - **Criterion**: I chose between "gini" and "entropy", I was receiving better performance with entropy so that is what I opted for in the end
    - **Max_Depth**: When I was testing max_depth, I stayed on the lower scale of numbers, during GridSearch, I routinely received the lowest of the options given so I continued to reduce during the testing phase. In the end I was going between "8", "4" and "2" with 2 giving me the best performance.
    - **n_estimators**: This feature was one that had a sizable impact on performance and surprisingly, smaller numbers seemed to perform better than larger numbers. In the testing phase I tried ranges anywhere between 7 and 500 finding 30 to give me the most desirable results.
    - **Max_Features**: This feature was also very interesting, and there was a trade off between performance in larger features and speed in less features. I found log2 to be the best cross-section between performance and speed and decided to go with that one.
- The tuning portion was actually very involved with the AdaBoost algorithm (my runner up) and I spent a long time adjusting the curve between the learning rate and the number of estimators. After attempting so many combinations, I was worried that I was relying too much on tuning instead of going back to the preprocessing step and trying to fix things higher up. That was when I switched to the random forest and implemented the SelectKBest function.

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"]

- It's almost easy to over-simplify what validation is, ideally you are validating that the algorithm you have made is capable of performing on your data. When it comes to tasks like this that sound relatively simple, the devil is always in the details and this is no different; there are many ways to mess up the validation step. One of the most important, and easy to overlook parts is the separation of your training and testing data, you do this to avoid training your algorithm to be very good at one dataset, while this sounds almost counter-intuitive, it's really quite simple, if you train your algorithm to only perform well on a set of 1000 records, when it receives a new record, it's could perform very poorly. This is because it will have been trained to only deal with a finite set of inputs and will not have been given the flexibility to react to new data. When you split training and testing data, you are effectively having your algorithm evaluate new data when you get to the prediction steps. This will give you an un-biased example of how your algorithm performs.
- For our dataset, we performed a **stratified shuffle split** cross validation to ensure that we had a representation of PoI in both the training and the testing data 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"]

- When I was doing research for this project, I learned early on that accuracy-score was not a good evaluation metric to indicate the performance of your algorithm. This was demonstrated on the famous MNIST dataset for handwritten number recognition. If you designed an algorithm to identify when a written number was not a 5 and it performed with an accuracy of 90%, that may not be a good representation of performance, based on distribution of numbers, it may be able to simply say that none of the numbers are 5 and given representation of non-5 numbers it would show a very high accuracy score. This is problematic because in essence your algorithm would not be doing anything to predict what a number is, rather just suggesting that there are no 5's at all.

- Because of the accuracy bias, there are two scores that are frequently used to measure the effectiveness of a machine learning algorithm - precision and recall. In the following sections I'll explain what each one means and explain why these two score are intertwined.
- Precision is the ability to detect a true positive in regards to the prediction of the algorithm vs the actual result. Depending on the use case that you are trying to solve, precision can be one of the most important metrics you can strive for. For instance, in healthcare testing, specifically tests for cancer would require a very high precision rate so that you can accurately identify that the result of a biopsy was actually cancer. Having a high rate of true positives would be critical in diagnosis.
- Recall is the ability to detect a true negative, in other words, the algorithm may not be great at detecting whether or not a positive is true or not, but if the result is negative, a high recall suggests that it is definitely negative.
- There is a balance between precision and recall in which when one is raised, the other is reduced and vise versa. In the event that an algorithm accurately identifies 100% true positives and 100% true negatives; it would be known as a perfect model.
- In our problem set, precision and recall represent the ability to effectively identify a person as a person of interest or not a person of interest respectively. A higher rate of precision indicates that the algorithm is able to identify a person is a person of interest based on the prediction vs the label. The recall is the opposite, showing that the algorithm can indicate that the prediction matches the label in determining that a person is not a person of interest.