The question of interest for this data set is: Can one design an effective algorithm for a binary classifier using the provided information / data? The data set itself consisted of a collection of emails obtained from Enron employee email accounts and financial information. The task was to use this data to construct a machine learning algorithm that would classify individuals as a Person of Interest or not a Person of Interest. In this context a "Person of Interest" (hence forth known as POI) is one that was "indicted, reached a settlement or plea deal … or testified in exchange for immunity" [1] Since the POI labels were provided, the assumed setting was in supervised learning.

There were many data features that had already been extracted from the dataset. These included financial information such as a person's total stock holdings in Enron, salary, exercised stock option values among others. Some features that were provided that were extracted from the email archive included basic information about total number of sent messages, received messages, and number of messages sent to a known POI. In total there were 144 data points with only 18 of them being labeled as POI. Below is a summary of missing data values by category (obtained from exploratoryPlots: count_missing_values())

{'bonus': 64,
'deferral_payments': 107,
'deferred_income': 97,
'director_fees': 129,
'email_address': 34,
'exercised_stock_options': 44,
'expenses': 51,
'from_messages': 59,
'from_poi_to_this_person': 59,
'from_this_person_to_poi': 59,
'loan_advances': 142,
'long_term_incentive': 80,
'other': 53,
'poi': 0,
'restricted_stock': 36,
'restricted_stock_deferred': 128,
'salary': 51,
'shared_receipt_with_poi': 59,
'to_messages': 59,
'total_payments': 21,
'total_stock_value': 20}

So we can see that deferral payments, director fees, loan advances and restricted stock deferred are missing many data values. Rather than imputing values, a model will be developed with and without these features and compared.

As a quick and dirty EDA I constructed showed that there were significant outliers in almost every feature. However, when I removed the outliers, the positive POI data points were being removed at a proportionally higher rate than non-POI. Since positive POI is a small set removing the outliers would make the set even more skewed, so the analysis was run with and

without outliers and was shown to perform better with the outliers left in. Further, since there were typically a few extreme outliers, the data was rescaled (log10) for positively valued features. The feature scaling boosted performance but ultimately lead to a more complex model were the best performance used nearly all features while without scaling the performance and complexity of the best performing model were reduced.

The features that I chose to use were PCA vectors. In a PCA the construction of the vectors are in order of decreasing variance capture. So the first vectors capture more of the variance among the features than the later vectors. The features were scaled first, just in case the PCA algorithm did not do this itself. Scaling is necessary in PCA because PCA uses a Euclidean distance measure to determine the amount of variance a component captures, and so different scales will favor one feature direction over another incorrectly simply due to scale. To decide how many features to principal components to use a grid search (my own code not GridSearchCV) was performed, and the "best" performer determined how many features to use. The performance measure that I used was an average of the average precision and recall scores on the test set during cross validation

The different algorithms that I tried were an Adaboost Classifier and similarly a Random Forest Classifier, a Logistic Regression Classifer with l2 penalty, and several Support Vector Classifiers. For the SVC, parameter tuning was performed with a grid search over the parameters of "C", "kernel", and the number of features to include. The performances were compared in metrics of precision and recall. Several of the models had performance and recall that exceeded the desired threshold of 0.30.

Parameter tuning refers to using a single model in which the parameters that the model depends upon are varied to "tune" the model so that performance is improved without changing the type of classifier being used. I chose a hands-on approach, and used a manual grid search instead of sklearn's GridSearchCV functionality. As I had chosed a support vector classifier, there were several parameters to tune that I chose: which kernel to use, penalty for misclassification (C), number of features to use - while this is more feature engineering and isn't a parameter of the model it is a refining of the algorithm as a whole. I then obtained a list of the top performers and selected the best average performer on metrics of recall and precision.

Validation is the process of testing a developed model on some new data to see how it performs outside of the data that was used in training. An incorrect approach would be to use data that the model has previously seen to validate how well the model will generalize. To validate my model, I used a stratified cross validation approach.

The two evaluation metrics that I used were average precision and recall on the test sets during stratified cross validation. Precision measures how much we can trust the model's conclusion of a positive classification (POI). Recall is a measurement of how effective the model is at correctly classifying positive (POI) results as being positive. So a recall score of 1.0 in this setting would mean that we correctly classified all POI's. While a precision score of 1.0 would mean that we did not incorrectly classify any non-POI's as POI's. The model that I arrived at (using a manual grid search and all features) had an average precision of 0.533 and average recall of 0.400 during cross validation. And when the model was run through the provided "test_classifier" function the precision and recall were 0.55841 and 0.89150 respectively and an accuracy of 0.86442. This means that approximately 56% of those that the model classified as

POI were labelled POI and that the model correctly labelled approximately 89% of POIs as POI. (Note that model construction and test_classifier function had to be applied to the modified features.) After removal of the four previously mentioned features that are missing many values the best model is an SVC with C=20, linear kernel and using the top 14 PCAs ranked in decreasing variance ratio had an average precision of 0.3833 and average recall of 0.50 over the cross validation sets and precision of 0.57820 and recall of 0.77450 using test_classifier. So a trade off of recall for a gain in precision when the four features are dropped.

I was curious if the Enron data set features of 'restricted_stock_deferred', 'restricted_stock', 'deferral_payments', 'total_stock_value' reflected a person's position in the stock asset. That is if 'restricted_stock_deferred' was negative did this mean that the person had a short position in the stock? If this is the case, I thought it could be interesting to see the total number of categories of the four that had short positions (value < 0). So, I constructed a simple count feature that counted the 'net_negative' or the total number of negative positions and 'net_negative_sum' which was simply the sum of these four features. Then retrained with manual grid search and stratified cross validation an SVC model. The result was that best classifier now had a test_classifier precision of 0.51269 and recall of 0.84850. As this was a slight decrease in performance these features will not be used in the final model.

References:
[1] https://classroom.udacity.com/nanodegrees/nd002/
[2] Raschka, Sebastian, and Randal S. Olson. *Python machine learning: unlock deeper insights into machine learning with this vital guide to cutting-edge predictive analytics*. Birmingham: Packt Publishing, 2016. Print.
[3] "Scikit-learn." Scikit-learn: machine learning in Python — scikit-learn 0.18.1 documentation. N.p., n.d. Web. 22 Apr. 2017.