Solution Statement:

"Use the Machine Learning Workflow to process and transform Enron data to create a prediction model. This model must predict which people are likely to a Person of interest in the Enron scandal with .30 or greater precision and recall values."

Understanding the Dataset and Question

Data Exploration:

The data set contains financial and email information for 146 people in the Enron scandal. In the data set each of these individuals is labeled whether they were or were not a person of interest in the Enron scandal. Of the 146 individuals 18 are labeled as being a person of interest and 126 are not. This is indicated by a boo value of True or False. I changed these values to numeric where 1 equals True and 0 equals False

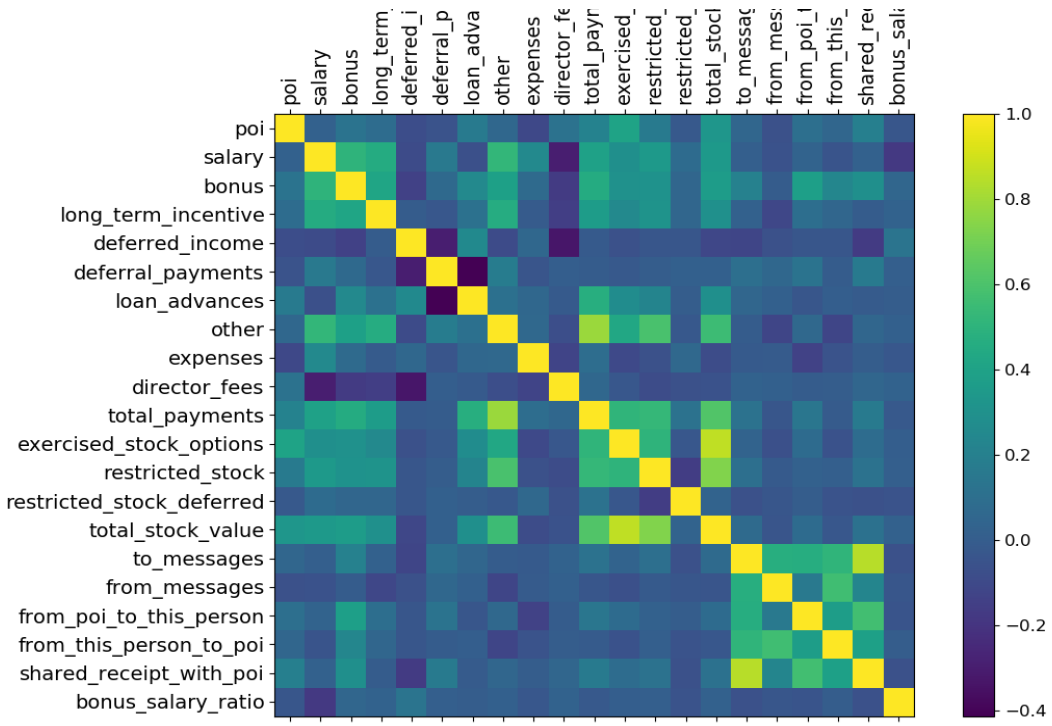
There are a total of 19 features provided with the data set. I will begin my investigation by removing total payments and total stock value from the feature list as these are derived values from other features. Also, I will be creating one additional feature, salary to bonus ratio.

Salary to bonus ratio = salary / bonus.

I chose this new feature because a bonus payment could be a way that POIs benefitted financially compared to those that are not POIs due to the fraud that was going on within Enron. This ratio could be an indicator of POIs.

Missing values plagues this data set with only 52 individuals not missing any data values. To fix this I will be using imputation to fill in any missing values with the mean value of that column. With such a small data set, simply ignoring or dropping these individuals would lead to a small data set that is not conducive to being able to reach our goal outlined in the solution statement. Once these values have been imputed, I created a correlation matrix to view the relationship between our features, as can be seen below.

When looking at correlations to poi exercised stock options, total stock value, and shared receipt with poi have the strongest correlation out of any of our other features. Even with this I still intend to leave total stock value from the final features list.



Correlation Matrix

Outlier Investigation:

To investigate outliers, I first calculated the 1st and 3rd quantiles for each column of the data set. I then used these values to calculate the interquartile range. Using this I created a list of persons by the number of outlier values they had and printed the top 15 results, detailed below.

Name	# of outliers
LAY KENNETH L	13
FREVERT MARK A	12
DELAINEY DAVID W	9
BELDEN TIMOTHY N	8
LAVORATO JOHN J	8
KITCHEN LOUISE	7
HAEDICKE MARK E	7
KEAN STEVEN J	7
SKILLING JEFFREY K	6
RICE KENNETH D	6
ALLEN PHILLIP K	6
BAXTER JOHN C	6
WHALLEY LAWRENCE G	6
REYNOLDS LAWRENCE	5
BECK SALLY W	5

I decided that I would keep all outliers in the data. One reason being that our primary POIs are included as outliers. Both Kenneth Lay and Jeffrey Skilling are present in our outliers. Since we are keeping outliers in our data set, I applied a SciKit Learns RobustScaler to our data set which is robust to outliers in the data. This Scaler removes the median and scales the data according to the quantile range. However, I removed this scaling because I decided to not use an algorithm that required scaling of features.

During the outlier exploration, I also noticed that there were to entries in the data that did not belong to individuals. There is a Total row and a row for Travel Agency in the Park, these values were dropped from the data set as they are not representative of people. The total row is derived from all entries and Travel Agency in the Park is a business entity.

Optimize Feature Selection/Engineering

I began my feature selection by running algorithms with my original features, as mentioned earlier this was all the available features. The results of which were as follows:

GaussianNB: Accuracy: 0.35413, Precision: 0.15200, Recall: 0.83950

DecisionTreeClassifier: Accuracy: 0.81153, Precision: 0.2970, Recall: 0.30250

LogisticRegression: Accuracy: 0.85000, Precision: 0.37245, Recall: 0.18250

Based on these results I chose to continue to pursue and tune features for the decision tree classifier. To do this I ran feature importance, which returned the following results.

Feature: 0, Score: 0.00
Feature: 1, Score: 0.00
Feature: 2, Score: 0.05
Feature: 3, Score: 0.08
Feature: 4, Score: 0.00
Feature: 5, Score: 0.00
Feature: 6, Score: 0.05
Feature: 7, Score: 0.19
Feature: 8, Score: 0.00
Feature: 9, Score: 0.23
Feature: 10, Score: 0.04
Feature: 11, Score: 0.00
Feature: 12, Score: 0.05
Feature: 13, Score: 0.00
Feature: 14, Score: 0.19
Feature: 15, Score: 0.05
Feature: 16, Score: 0.07

With these results I removed all features with a score of 0 and reran the decision tree classifier algorithm. The results of which are below.

DecisionTreeClassifier: Accuracy: 0.81080, Precision: 0.30329, Recall: 0.32300

We see a decrease in accuracy but an increase in both precision and recall values by removing these features. However, this means I will not be able to use my added feature of salary to bonus ratio as salary was one of the features removed during this process.

Pick and Tune an Algorithm

After feature testing and optimization, I have decided to move forward with and tune the decision true algorithm. During the tuning process I will be selecting all hyperparameters by hand. During this stage of learning and knowledge of machine learning algorithms, I want to see the impact of each hyperparameter change as it happens while maintaining control of these changes.

Tuning of our algorithm is important as this will control over-fitting and under-fitting of the model. Over-fitting occurs when our model fits closely to our training data but when tested with testing data the model will likely perform poorly as it has high variance. Alternatively, under-fitting will perform poorly on both training data and test data. Tuning of the models hyperparameters is necessary to find a best fit of our model to our data set.

DecisionTreeClassifier(class_weight='balanced')
Accuracy: 0.82293 Precision: 0.32289 Recall: 0.29900

DecisionTreeClassifier(class_weight='balanced', criterion='entropy')
Accuracy: 0.82493 Precision: 0.33172 Recall: 0.30850

DecisionTreeClassifier(class_weight='balanced', criterion='entropy', max_depth = 10)
Accuracy: 0.82467 Precision: 0.33209 Recall: 0.31150

DecisionTreeClassifier(class_weight='balanced', criterion='entropy', max_depth = 7)
Accuracy: 0.81547 Precision: 0.31870 Recall: 0.33750

Based on the performance of the decision tree classifier using different hyperparamters, the final model choice is

DecisionTreeClassifier(class_weight='balanced', criterion='entropy', max_depth = 7)
Accuracy: 0.81547 Precision: 0.31870 Recall: 0.33750

Validate and Evaluate

Validation:

Validation is a process of holding a portion of our data set to be used for training and testing of our model. This is important as we do not want our model to become over fitted to our training set and when we deploy the model for it to perform poorly on real world data. To prevent this, we validate our model off of a test set of data after training our model using our training set. This allows us to validate the performance metrics that we saw during training will closely match what we can expect on real world data.

This was performed using cross validation using stratified shuffle split. Cross validation is the process of withholding a random sampling of the data set to be used for training and testing. Stratified shuffle split retains the same portion of target classes in each sample set that is created. During testing we utilized 1000 random samples during cross validation. We use these random samplings to prevent our model from over fitting to our training set and allows us to better tune our models hyperparameters to perform well on both training and test data.

Evaluation:

When evaluation the model's performance I was focused on both precision and recall. My solution statement stated that I was attempting to achieve a .30 or higher in both performance areas.

In an imbalanced classification problem with two classes, precision is calculated as the number of true positives divided by the total number of true positives and false positives.

- Precision = TruePositives / (TruePositives + FalsePositives)

Whereas recall is calculated as the number of true positives divided by the total number of true positives and false negatives.

- Recall = TruePositives / (TruePositives + FalseNegatives)

The result is a value between 0.0 for no precision and 1.0 for perfect precision or recall.

References:

https://scikit-learn.org/

https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/

https://www.pluralsight.com/courses/python-understanding-machine-learning

https://github.com/mspbannister/dand-p5-enron