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 this project is to identify the persons of interest related to collapse of Enron in 2002. These POIs perpetrated widespread corporate accounting fraud to hide billions of dollars in debt from failed deals and projects.

The dataset is a dictionary with employee email and compensation data available publicly as part of the fraud investigation. Employees marked as POIs in this dataset were all indicted, reached a settlement or testified in exchange of prosecution immunity.

Here are some details about the dataset:

Dimension of the dataset: *144 employees, 19 features, and 1 label*

Allocation across classes: *18 POIs and 126 non-POIs*

Number of features used in the final set: *Used 11 existing features and 5 newly-created features. Out of that, the final classifier selected 4 most-important features.*

Missing values: *Here are the counts of NaNs in the dataset:*

```
'bonus': 64,
'deferral_payments': 107,
'deferred_income': 97,
'director_fees': 129,
'email_address': 35,
'exercised_stock_options': 44,
'expenses': 51,
'from_messages': 60,
'from_poi_to_this_person': 60,
'from_this_person_to_poi': 60,
'loan_advances': 142,
'long_term_incentive': 80,
'other': 53,
'poi': 0,
'restricted_stock': 36,
'restricted_stock_deferred': 128,
'salary': 51,
'shared_receipt_with_poi': 60,
'to_messages': 60,
'total_payments': 21,
'total_stock_value': 20
```

Using Machine learning, a classifier can be built that can identify POIs, given the list of email and financial attributes. Part of the dataset (usually 70%) is set aside for the classifier to learn from, and remaining 30% is put aside to validate the performance of the classifier. So, after fitting the classifier to the training set, it can be run against the validation set (which was not yet seen by the classifier). The results of the classifier against validation set can then be compared against the actual POI values to determine the true performance of our classifier.

Outliers are datapoints that are very far from remaining data-points. These data-points can skew the distribution of training data. Many machine learning algorithms are sensitive to changes in distribution of data. These algorithms might be misled while learning from the data to place more weight on those outliers. This ultimately results in poor performance of those models.

There was an outlier data-point in the Enron dataset. Upon further investigations, this dataset was associated with a record with total values for all attributes in the dataset. It was clearly an error, so I simply removed it.

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 started with entire set of features provided and ran a decision tree classifier on that set to come up with variable importance metrics. Then I selected only those variables with importance score > 0. Additionally, I kept variables **`from_messages`** and **`to_message`** because I needed them later to create new features. Here are all variables with importance scores >0 (and 2 extra variables to be used later):

```
total_payments :     0.112874779541
exercised_stock_options :     0.231746031746
bonus :     0.109451528031
restricted_stock :     0.120480432567
shared_receipt_with_poi :     0.0577200577201
expenses :     0.117875473623
other :     0.138852968951
from_this_person_to_poi :     0.0554431722656
from_poi_to_this_person :     0.0555555555556
to_messages :     0.0
from_messages :     0.0
```

Using the existing variables/attributes, I engineered 5 more attributes:

ratio_bonus_to_ex_stock:
ratio of bonus to exercised_stock_options

from_this_person_to_poi_pctof_from:
from_this_person_to_poi as % of from_messages

from_poi_to_this_person_pctof_to:
from_poi_to_this_person as % of to_messages

ratio_restrc_stock_to_total_pmnt:
ratio of restricted_stock to total_payments

ratio_ex_stock_to_total_pmnt:
ratio of exercised_stock_options to total_payments

In the final mode, out of the five new variables created, **from_this_person_to_poi_pctof_from** was selected as 2nd most important variable by SelectKBest() with importance score of 13.79.

For the final estimator, I passed grid-search parameters for SelectKBest(k) in range of 2,4 and 6. I gave lower values for K to make the training process run faster. Testing with larger values of K did not seem to improve the performance, and it was causing the pipeline to run much longer than necessary. The final estimator selected following 4 features:

(**exercised_stock_options,** 9.9561675820785211),
(**bonus,** 30.652282305660439),
(**shared_receipt_with_poi,** 10.669737359602689),
(**from_this_person_to_poi_pctof_from,** 13.791413236761116)]

Performing scaling and running classifiers, such as Naive Bayes, SVM and Random Forest on that data (without cross validation), seemed to improve performance (compared to unscaled dataset). However, after performing cross-validation, the performance of scaled dataset was equal or worse compared to unscaled dataset. Therefore, I decided to not use scaling for my final estimator.

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 SelectKBest() and RandomForestCalssifier() with cross-validation. I used this particular combination because its performance was very good (if not the best), it was not very complicated to fine-tune/implement, and it ran relatively quick.

I split the dataset into train and test sets. I tried to fit following classifiers on the training set:

GaussianNB():

This was a good starting point. This provided benchmark scores of precision and recall, both higher than .3.

StandardScaler() and svm.SVC():

Accuracy and precision score was high for this one. However, recall score seemed to fall behind. Also, this algorithm seemed more sensitive to randomly selected training samples.

StandardScaler() and RandomForestClassifier()

This combination did not yield good outcome. So it was abandoned.

SelectKBest() and RandomForestClassifier()

Besides GaussianNB, this combination provided the best result, so I decide to go ahead and start tuning parameters for this.

4. What does it mean to tune the parameters of an algorithm, and what can happen if you dont 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”]

Based on features and distribution of data, the algorithm needs to be fine-tuned to get best performance on the validation set. For example, applying a linear algorithm on a non-linear dataset will result in a biased model. The algorithm should therefore be optimized to work better with natural shape of the data. Sklearn has GridSearchCV to automatically grid-search a range of user-provided parameters, and it selects the best combination of parameters.

For the algorithm I chose for this project, I tuned the parameter k for SelectKBest, and parameters n_components and max_depth for RandomForestClassifier().

For SelectKBest(), parameter k indicates number of features to choose from the training dataset. Selecting small value for k might leave out some important features, whereas selecting too large value for k might cause the algorithm to run very slowly, and potentially cause overfitting. I provided the values [2,4,6] to the grid-search and it selected 4 as the best k value.

For RandomForestClassifier(), **n_components** selects the number of trees in the ensemble, and **max_depth** indicates the maximum depth of each tree. For n_components and max_depth, I provided the values [10, 20, 30, 40] and [1, 3, 5, 7] respectively. Grid-search selected 30 and 3 as best values.

I think this makes sense because large number trees in an ensemble using randomly selected features reduce the variance between trees. Also, having shallow trees of depth 3 or less potentially avoids overfitting.

5. What is validation, and whats a classic mistake you can make if you do it wrong? How did you validate your analysis? [relevant rubric items: “discuss validation”, “validation strategy”]

Validation is a process of running the classifier with previously unseen dataset to measure its true performance. The classic mistake is overfitting. This happens if the classifier is unnecessarily complex and it fits the training data very well, but does not fit well with validation dataset. In other words, the classifier does not generalize very well. This type of classifier has low bias, but very high variance. Ideally, we’d want the bias/variance for a classifier to be more balanced, allowing comparable performance for both training and validation sets.

I primarily relied on the validation script tester.py provided in the course to validate my final algorithm. This code takes as input the classifer, the dataset with finalized features, feature names and k-folds (default is 1000). Here are the steps it goes through:

- Split the data into labels and features
- Use StratifiedShuffleSplit() to get test/train indices for test/train splits. In this case, we have 1000 different sets of test/train splits.
- Fit the tuned classifier to each of the train split and use corresponding test split to predict labels. Calculate True/False Positive/Negatives by comparing the predicted label with actual label. Perform this on all 1000 splits.
- Sum the total True/False Positive/Negatives on all 1000 sets of train/test splits.
- Calculate accuracy, precision, recall, f1 and f2 base on total True/False Positive/Negatives. Here are the results of running this validation:

```
Accuracy: 0.84227
Precision: 0.41449
Recall: 0.44350
F1: 0.42850
F2: 0.43738

Total predictions: 15000
True positives:  887
False positives: 1253
False negatives: 1113
True negatives: 11747
```

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 algorithms performance. [relevant rubric item: “usage of evaluation metrics”]

   Here are two metrics and the average performance (as computed by tester.py).

   ```
   Precision: 0.414      Recall: 0.443
   ```

   Precision is the ratio of correctly predicted positive outcome against total outcome predicted as positive by the algorithm. So, if the algorithm predicted 100 POIs in the test set, precision gives us what proportion of that is actually POI.

   Recall is the ratio of correctly predicted positive outcome against actual positive outcomes. So, if there are actually 50 POIs in the test set, recall gives us what proportion of that was correctly identified by the algorithm.