Question 1:

Machine learning techniques generate predictive models to predict continuous quantitative response (regression) or a categorical output (classification) given some input features. The question of concern in this project is that can we predict 'persons of interest' involved in the Enron fraud case given the financial and email data of the enron employees. We have been given a data set that contains

1. Financial and email data of 146 entities (145 individuals excluding the 'TOTAL' entry which is an outlier that was omitted from the dataset for analysis)
2. 20 features related to finances and emails of the 145 enron employees.
3. 21st variable ('poi') is the the response variable and tells us whether a person is originally a 'person of interest' or not. In the given dataset, 18 of 145 individuals are 'person of interest' which is an indicator of them being involved in the fraud case.
4. There are some missing values in the above mentioned features as well which have been given in the following table

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

Given the above characteristics of the data and the nature of our question, we can employ machine learning algorithms for classifying enron employees as 'poi' or 'non-poi' and test our performance by comparing it with the actual response (poi) data.

Outliers:

In the given dataset (python dictionary), there is a key named, 'TOTAL' in addition to individual employees of Enron. The features for TOTAL are the aggregated/ sum values of all employees' features and does not represent any individual. For this reason, the entry has been completely removed from the dataset.

Other than that scatter plots of various features also reveal certain extreme values but they have been kept in place (not removed) because they are most likely to be valid data points despite their extreme values. A few of these scatter plots have been attached in the appendix section for reference.

Question 2:

In my initial exploration, I fit a decision tree and manually selected the best performing features by using the '`features_importances_`' method. At this point I had not used `tester.py` to implement cross validation to evauate my decision tree. I was just using the test data (30 % of the entire data) obtained using train_test_split function.Then, I changed my approach and decided to use automated feature selection methods (`SelectKBest`) available in sklearn. First, I investigated the missing values in the available features and did not use the features with more than 50 % of the data missing. Then, I added the rest of the features in the features_list variable in poi_id.py (The list can be reviewed in the python script or the appendix of this document) and used all 8 features for the K-Nearest Neighbor classifier. The evaluation metrics given by tester.py function have been copied below:

```
Accuracy: 0.82687      Precision: 0.33333     Recall: 0.29850
```

Afterwards, i followed up with lower values of K (K=4, 5, 6, 7) for SelectKBest. I observed that the performance improved as i reduced the value of K and was optimal for K = 5. The performance metrics given by tester.py for K=8 were:

```
Accuracy: 0.86647  Precision: 0.49904   Recall: 0.39150  F1: 0.43878
```

This led me to the conclusion that the performance of the KNN classifier is being driven by the 5 strongest features. To check which features are driving performance, i used the get_support() and the score_ method. According to the output of these methods (copied below), the most important feature in the features_list were (in order of importance based on the scores output):

1. bonus
2. fraction_emails_from_poi
3. salary
4. shared_receipt with_poi
5. fraction_emails_to_poi

```
## get_support() output
array([ True,  True,  True, False, False, False,  True,  True, False],
dtype=bool)

##scores output
array([  9.32492268,  10.97716328,   3.903868  ,   3.48558479,
3.66938716,   2.96112262,   7.79264219,  13.02958944,   2.77410815])
```

New feature generation and feature scaling:

The above mentioned feature, 'fraction_emails_from_poi' and 'fraction_emails_to_poi' were not part of the original dataset and were added later using the already existing features. The features have also been tested in the algorithm which shows that the features 'fraction_emails_to_poi' and 'fraction_emails_to_poi' drive much of the performance of the algorithm. I also performed feature scaling for input features as I was expecting that because the knn algorithm depends solely on distance, consistency in feature scales would be crucial. But later i realized that parameter tunes for distance metric are available such as 'chebyshev' in knn algorithm which can give the same performance for unscaled features as 'euclidean' distance metric in case of scaled features.

Question 3:

The two major algorithms i used were:

1. K Nearest Neighbor classifier (Parameter k tuned through cross validation)
2. Decision tree classifier (Parameter '`min_samples_split`' tuned through cross validation)

In both the algorithms, I used SelectKBest to identify the best features. K Nearest neighbor was my algorithm of choice because of better performance (based on evaluation metrics output reported by tester.py)

For 5 best features, The decision tree classifier gave me the following evaluation metrics (copied from tester.py output)

```
Accuracy: 0.82720  Precision: 0.32890   Recall: 0.28450
```

Whereas for K nearest neighbor, which was the final choice algorithm, i got the following performance metrics using tester.py

```
Accuracy: 0.86647  Precision: 0.49904   Recall: 0.39150  F1: 0.43878
```

Question 4:

The performance of a machine learning algorithm greatly depends upon the chosen values of tunable parameters. The right choice of these parameters can depend on the nature of the data and the the kind of question we are trying to address. Not tuning the parameters can adversely affect performance of the chosen algorithm as measured by metrics such as accuracy, precision and recall.

For KNN classifier, I tuned the `'n_neighbors'` (number of neighbors) parameter and the distance metric by implementing cross validation (GridSearchCV in sklearn). The cross validation however gave me surprising results for `'n_neighbors'` parameter. The optimal value i got was n_neighbors=1, which is generally thought of a value that overfits the data as only a single nearest neighbor is used to

make classification decisions. But for our purposes in the given case, KNN with n_neighbors = 1 seems to give the best performance and optimizes the 'recall score'.

Question 5:

Validation is an essential process in machine learning where we evaluate the performance of our algorithm on a test data rather than the training data used to train the algorithm. This can be achieved in various ways. The simplest way is to split the data into training and test portions. Since this simple method provides a compromise between the size of training and test dataset, general methods like cross validation are also available. If the validation process is not done properly, there is always a danger of overfitting to the training data. The overfit data will have high 'variance' and hence will not be able to perform well generally when faced with other contexts and situations.

Initially I used the simple train_test_split in poi_id.py script. Later, to get more robust performance metrics, I used the tester.py script provided by Udacity (which implements and performs cross validation).

Question 6:

The recall, precision and accuracy scores as reported by the tester.py script for the final algorithm (KNN with n = 1) are copied below:

```
---------------------------------------------------------------------------------------------------------------------
-
Accuracy: 0.86647      Precision: 0.49904     Recall: 0.39150  F1: 0.43878
F2: 0.40913
Total predictions: 15000     True positives:  783   False positives:  786
False negatives: 1217  True negatives: 12214

---------------------------------------------------------------------------
```

The interpretation for the performance metrics is:

1. The accuracy metric shows that the algorithm gets 85.414 % of the classifications right (poi and non poi). The formula for calculating the accuracy is:

   accuracy = (true positives + true negatives) / Total predictions

   accuracy = (783 + 12214) / 15000 = 0.86647

2. The precision (0.48716 or 48.716 %) shows that out the predicted pois, what fraction/ percentage were actually pois. The formula is:

   precision = true positives / (true positives + false positives)

   precision = 783/ (783 + 786) = 0.49904

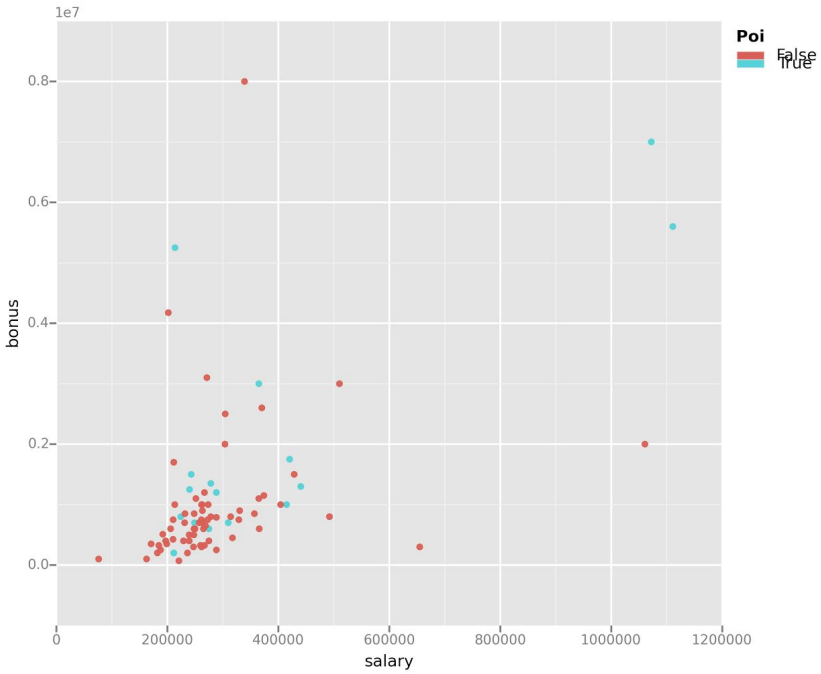
3. The recall (0.39850 or 39.850 %) shows that out of the actual pois, what fraction/ percentage of them were predicted poi's. The formula for calculation using tester.py output is:

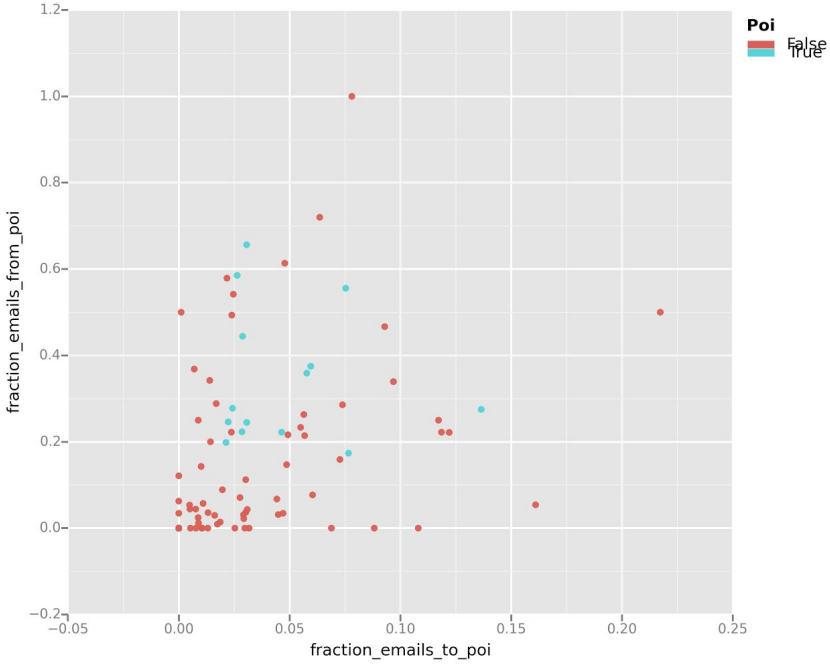
   recall = true positives / (true positives + false negatives)

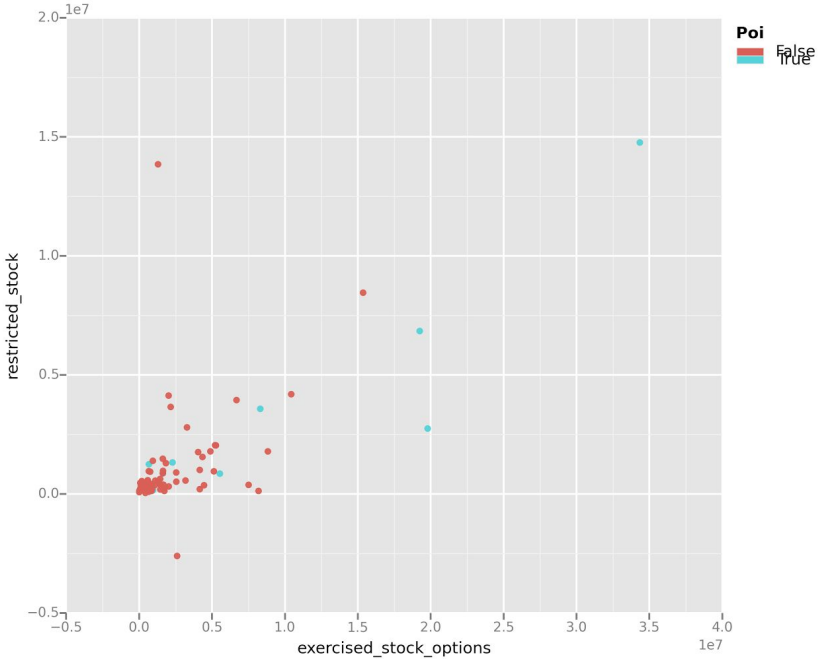
   recall = 783/ (783 + 1217) = 0.39150

Appendix:

Some exploratory plots for outlier detection:




Poi
False
True
fraction_emails_from_poi
fraction_emails_to_poi
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.05
0.00
0.05
0.10
0.15
0.20
0.25


Poi
False
True
restricted_stock
exercised_stock_options
1e7
2.0
1.5
1.0
0.5
0.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0