ANSWER THE FOLLOWING QUESTION RELATED TO THE PROJECT

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 find persons of interest in the Enron scandal making use of the financial data provided by FindLaw as well as the publicly available Enron email data.
The fact that this is a well-known legal case developed in the earlier 2000's makes it possible for us to know in advance who the persons of interest (poi) were. Whether a person was a poi or not can be accounted for using a label with two possible values (True or False, 0 or 1, etc.) and this detail makes it suitable for supervised machine learning. In such an approach, we train a classifier in a set of known features and labels and with that knowledge we try to predict the outcome of a subset of data unknown to the classifier.
The availability of financial data limits the dataset to just 144 former Enron employees, 18 of whom were poi. Initially, we found 285 missing entries, among them some corresponding to 4 poi. We performed a thorough search in the Enron email dataset, and we were able to find email-related data for 44 people (that would have potentially reduced those 285 entries down to 70). We then realized that these data were altered and therefore not able to be included without having an impact on the performance of the classifiers. We imputed the missing values with the median of the given features taking into account if they belonged to the poi or the non-poi group. For the reasons explained above we highly valuate each of our data points. We detected a clear outlier ('TOTAL') in the financial features that could be harmlessly taken out. There were more outliers, but we decided to keep the all the remaining data unless it has a significant impact on the performance of our model. We also removed two more data points: THE TRAVEL AGENCY IN THE PARK (because it is not a person), and EUGENE E. LOCKHART, because there is no data associated with him, except for the fact that he was not a poi.

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"]

We ended up using:

- expenses (0.14)
- shared_receipt_with_poi (0.12)
- to_poi_rate (0.12)
- from_poi_rate (0.12)
- from_this_person_to_poi (0.1)
- long_term_incentive (0.08)
- to_poi_median_pubIndex (0.08)
- other (0.06)
- salary (0.04)
- exercised_stock_options (0.04)

We used the feature importance that comes to the classifier we used (AdaBoost), and we ranked all our 23 features from highest to lowest. These features were such that their importance was higher than 0.02.
We did feature scaling, but just while we were in the classifier selection process. It was necessary for selecting the more relevant features using KBest in the case of Naïve Bayes. As we ended up using AdaBoost which has Decision Tree as a weak classifier by default, we did not need to use feature scaling.

We devised two kinds of new features.

- to_poi_rate: ratio of from_this_person_to_poi / from_messages
- from_poi_rate: ratio of from_poi_to_this_person / to_messages

One was a ratio of existing features, such as "to_poi_rate" it accounts for the fraction of the emails sent by a person to poi relative to all the emails that person sent. It measures somehow the "strength" of their relationship or association.

Another kind of features were extracted from the Enron email dataset.

Using the whole email dataset, we created an intermediate feature called pubIndex. It accounts for the number of people implicated in one email message, or a measure of how public a given message was. Its value equals zero if it's an email sent from one person only to himself (or herself). It is equal to one if there is just one recipient and increases as the number of people involved in a given communication goes up. Its minimum value is zero and there is in principle no upper limit.

- from_messages_median_pubIndex: We grouped all the emails sent by a given person and took the median of the pubIndex feature.
- to_poi_median_pubIndex: The same as above but taking into account only those messages sent to poi.

The logic behind defining such features is that maybe the more public the communication with poi was, the less probable a poi this person is liked to be.

As we can see, the list of the most relevant features shown above contains three out of our four new 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”]

All the algorithms we tried out complied with the minimum requirement for this project. We ended using AdaBoost. We also tried Naïve Bayes and Decision Tree. Among them, the worse performance was attained by Naïve Bayes as the following table shows. Although Decision Tree and AdaBoost classifiers were close in performance, the small difference in F1 and the fact that AdaBoost is more robust when facing new data, tipped the balance in this case.

CLASSIFIER	ACCURACY	PRECISION	RECALL	F1
Naïve Bayes	0.864	0.489	0.368	0.420
Decision Tree	0.907	0.612	0.829	0.704
AdaBoost	0.923	0.674	0.819	0.739

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 tun e -- if this is the case for the one you picked, identify and briefly explain how you would h ave done it for the model that was not your final choice or a different model that does util ize parameter tuning, e.g. a decision tree classifier). [relevant rubric items: “discuss para meter tuning”, “tune the algorithm”]

To tune the parameters of an algorithm means selecting them in such a way that the ensemble of all the values selected for the parameters ended up maximizing the algorithm’s performance. If we fail doing the tuning, we could live under the impression that our parameters were optimized and then, the model scores poorly when facing new data.

We used GridSearchCV, it allows to construct a grid of all the combinations of parameters, tries each combination, and then reports back the best combination/model. In our case, we used it twice, in series. First, we tuned the parameters of the AdaBoost classifier (n_estimators and learning_rate), using the default parameters for the base estimator (Decision Tree). Then we used the best parameters found in the first step for AdaBoost and did a grid search for some parameters of the base estimator (criterion, splitter, max_depth, min_samples_leaf, min_samples_split). This approach is faster than using a single grid search with all the parameters and works just fine optimizing the classifier

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"]

When a predictive model is fitted with the training set, one does not know how well it will perform in a case of unseen data. That is what validation is about.

An issue often seen in machine learning is when a model is performing very well on the training set but does worse when facing unseen data. There are some commonly made mistakes behind this behavior. Overfitting is one of them. However, the contamination of the data with "leaking" from the validation set into the training set could also play a role.

The cross-validation method I used was Stratified Shuffle Split. It randomly creates multiple train test sets of data which works well with uneven datasets.

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"]

We used several metrics to evaluate the performance of our algorithms: Accuracy, precision, recall, and F1. All of them vary between 0 and 1.

Accuracy is just the ratio of the correctly predicted observations to the total number of observations. It constitutes a good measure of the performance in symmetric cases. In ours, accuracy alone is not the right way to assess the overall performance. A quick glance at our data shows that from all 143 people only 18 were poi.

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Where: TP: True Positives, FP: False Positives, TN: True Negatives, FN: False Negatives

Precision: The rate of the correctly predicted positive cases to the number of all positive predicted cases. A high value of precision means few occurrences of false positives. In our case, that means we did well finding actual poi and do not "incriminating" innocent employees.

Precision = TP/(TP+FP)

Recall: Recall is the ratio of correctly predicted positive observations to all observations in a class. The question recall answers in our case is: Of all actual poi people, how many did we label? A high value of recall indicates that we found almost all existing poi in our dataset.

Recall = TP/ (TP + FN)

F1 score: F1 Score is the weighted average of Precision and Recall.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

This score works better than accuracy when we face an uneven class distribution.