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.

Enron Corporation was an American energy, commodities, and Services Company based in Houston, Texas during the 1990's. In 2000 Enron employed approximately 20,000 staff with claimed revenues of nearly $101 billion. However at the end of 2001, it was revealed that it's reported financial condition was sustained by institutionalized, systematic, and creatively planned accounting fraud, known since as the Enron scandal. Enron has since become a well-known example of willful corporate fraud and corruption. During the investigation by the Federal Energy Regulatory Commission (FERC) in the wake of the Enron bankruptcy vast amounts of data were collected including financial and email data. At the conclusion of the investigation the emails and information collected were deemed to be in the public domain and to be used for historical research and academic purposes.

The goal of this project is to use the given dataset and leverage machine learning methods to determine the most likely persons of interest (POIs) that committed fraud that lead to the ultimate demise of Enron. The dataset is broken down into two broad categories; emails and financial data.

The following list is some analysis of the dataset:

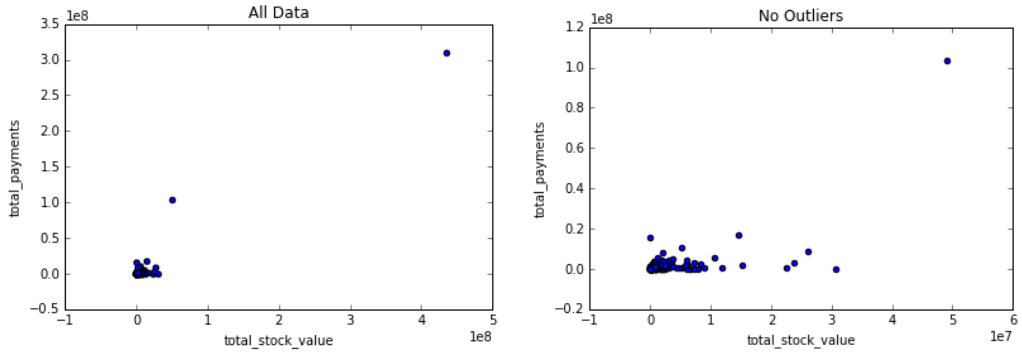
```
There are 146 people in the dataset
There are 21 features used per person
Total number of POI's in the dataset: 18
Total number of non-POI's in the dataset: 128

The total number of data points in the dataset is 3066
The total number of non-NaNs in the dataset is 1708 with percent of total = 0.56

Legend - feature|Total non-NaNs|%|non_Nans for POIs|%|non_Nans for non-POIs|%)

('poi', 146, 1.0, 18, 1.0, 128, 1.0)
('salary', 95, 0.65, 17, 0.94, 78, 0.61)
('bonus', 82, 0.56, 16, 0.89, 66, 0.52)
('long_term_incentive', 66, 0.45, 12, 0.67, 54, 0.42)
('deferred_income', 49, 0.34, 11, 0.61, 38, 0.3)
('deferral_payments', 39, 0.27, 5, 0.28, 34, 0.27)
('loan_advances', 4, 0.03, 1, 0.06, 3, 0.02)
('other', 93, 0.64, 18, 1.0, 75, 0.59)
('expenses', 95, 0.65, 18, 1.0, 77, 0.6)
('director_fees', 17, 0.12, 0, 0.0, 17, 0.13)
('total_payments', 125, 0.86, 18, 1.0, 107, 0.84)
('exercised_stock_options', 102, 0.7, 12, 0.67, 90, 0.7)
('restricted_stock', 110, 0.75, 17, 0.94, 93, 0.73)
('restricted_stock_deferred', 18, 0.12, 0, 0.0, 18, 0.14)
('total_stock_value', 126, 0.86, 18, 1.0, 108, 0.84)
('email_address', 111, 0.76, 18, 1.0, 93, 0.73)
('from_messages', 86, 0.59, 14, 0.78, 72, 0.56)
('from_poi_to_this_person', 86, 0.59, 14, 0.78, 72, 0.56)
('from_this_person_to_poi', 86, 0.59, 14, 0.78, 72, 0.56)
('shared_receipt_with_poi', 86, 0.59, 14, 0.78, 72, 0.56)
('to_messages', 86, 0.59, 14, 0.78, 72, 0.56)
```

There are 146 points in the dataset supposedly representing people however there are a couple of points that are not people, as indicated later in the outliers section, that will eventually be removed. Each person has 21 features (10 payments, 4 stock value, 6 email, and the POI label). The total number of possible data points are 3066 however and about 44% of the data points are missing (NaNs). I generated a table to help me understand how the NANs were distributed over various features. Turns out there was only one feature with scarce data points, loan_advances, with only 4 data points. With

the overall scarcity of data points available I decided on a philosophy use as many data points as possible and try to minimize the number I eliminate. Therefore, after removing the obvious outliers I will utilize all features at the beginning.

Were there any outliers in the data when you got it, and how did you handle those? [relevant rubric items: "data exploration", "outlier investigation"]

A scatter plot of the total_payments versus the total_stock_value revealed an obvious outlier called "Total". This data and all associated features were removed. Two other data points were eliminated as well, "THE TRAVEL AGENCY IN THE PARK" was removed as this has nothing to do with identifying the POIs, and "EUGENE LOCKHART" was removed as there is no data associated with this person.



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.)

The features I ended up using for my POI identifier were the financial data of payment and stock_value features which include 14 features. Following this I removed the email address feature because it was not useful and created exceptions. Also after further analysis it was found that the email features did not improve the models as they were less accurate with both precision and recall scores dropping off. Here is an example of this behavior:

Results with all data, including email features.

```
Pipeline(steps=[('scaler', StandardScaler(copy=True, with_mean=True, with_std=True)), ('select',
SelectKBest(k=10, score_func=<function f_classif at 0x0000000008EC7EB8>)), ('decomp',
PCA(copy=True, n_components=2, whiten=False)), ('classifier', GaussianNB())])
        Accuracy: 0.85380  Precision: 0.42455 Recall: 0.27150    F1: 0.33120        F2: 0.29260
        Total predictions: 15000    True positives:  543        False positives:  736
        False negatives: 1457       True negatives: 12264
```

Only financial features included, payment and stock_value features.

```
Pipeline(steps=[('scaler', StandardScaler(copy=True, with_mean=True, with_std=True)), ('select',
SelectKBest(k=10, score_func=<function f_classif at 0x0000000008EC7EB8>)), ('decomp',
PCA(copy=True, n_components=2, whiten=False)), ('classifier', GaussianNB())])
        Accuracy: 0.86567  Precision: 0.49454 Recall: 0.33950    F1: 0.40261        F2: 0.36221
```

```
        Total predictions: 15000    True positives:  679        False positives:  694
        False negatives: 1321       True negatives: 12306
```

Two new features were engineered both related to the email data: (1) msg_to_poi_ratio, and (2) msg_from_poi_ratio. The purpose is to determine if the ratio of messages to and from POIs is higher than non-POIs, thinking a higher ratio would indicate another POI. As mentioned before, eventually I would scrap all the email features, so these new features were not used in the final model. I did go back, however, and add only the two new email features to the payment and stock_value features to see if the model improved however it did not as seen below the recall scores went below 0.3 close to the same result when all email features were included.

```
Pipeline(steps=[('scaler', StandardScaler(copy=True, with_mean=True, with_std=True)), ('select',
SelectKBest(k=10, score_func=<function f_classif at 0x0000000009F20EB8>)), ('decomp',
PCA(copy=True, n_components=2, whiten=False)), ('classifier', GaussianNB())])
        Accuracy: 0.85807  Precision: 0.44917 Recall: 0.28500    F1: 0.34873        F2: 0.30748
        Total predictions: 15000    True positives:  570        False positives:  699
        False negatives: 1430       True negatives: 12301
```

Another combination of features that was tested was the payment features only. The selection of features increased the precision score but the recall score decreased.

```
Pipeline(steps=[('scaler', StandardScaler(copy=True, with_mean=True, with_std=True)), ('select',
SelectKBest(k=8, score_func=<function f_classif at 0x0000000009F20EB8>)), ('decomp',
PCA(copy=True, n_components=2, whiten=False)), ('classifier', GaussianNB())])
        Accuracy: 0.85546  Precision: 0.56498 Recall: 0.26300    F1: 0.35892        F2: 0.29448
        Total predictions: 13000    True positives:  526        False positives:  405
        False negatives: 1474       True negatives: 10595
```

Other combinations were tried as well but none matching the best results by using the payment and stock_value features.

Also with respect to feature selection I used SelectKBest within the GridSearchCV using the k values of 8, 10 and 'all'. The k value with the best estimator was K=10.

Scaling was implemented to standardize the features, as data for some features can be orders of magnitude different than others we are trying to compare. I found the StandardScaler to be more effective than the MinMaxScaler when comparing precision and recall scores from the tester.py. A possible reason could be that MinMaxScaler is sensitive to outliers and the financial data has a few outliers that are legitimate data points we want to include as part of the analysis.

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"]

The scaled features were passed to the SelectKBest feature selection algorithm where the features were scored using the default 'f_classif' scoring function. This provided the ANOVA F-value for each feature. Higher F-values indicate the feature variability of group means is large relative to the variability within the groups – making this feature more relevant to the analysis, because the higher scores indicate a feature is significantly more unique and good for the analysis.

```
Feature Selection by SelectKBest

feature                       score     percent nan
0  exercised_stock_options  24.815080         29.4
1        total_stock_value  24.182899         12.6
2                    bonus  20.792252         43.4
3                   salary  18.289684         34.3
4          deferred_income  11.458477         66.4
5      long_term_incentive   9.922186         54.5
6         restricted_stock   9.212811         23.8
7           total_payments   8.772778         14.0
8            loan_advances   7.184056         97.9
9                 expenses   6.094173         34.3
```

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"]

The algorithm I ended up using was GaussianNB. I also tried DecisionTree and as the favorite of Katie Malone I also tried Adaboost.

GaussianNB provided the best results for precision and recall. It was a fairly simple model as I didn't have to do a lot of preprocessing. There are no parameters associated with GaussianNB.

```
Pipeline(steps=[('scaler', StandardScaler(copy=True, with_mean=True, with_std=True)), ('select',
SelectKBest(k=10, score_func=<function f_classif at 0x0000000008EC7EB8>)), ('decomp',
PCA(copy=True, n_components=2, whiten=False)), ('classifier', GaussianNB())])
        Accuracy: 0.86567  Precision: 0.49454 Recall: 0.33950    F1: 0.40261        F2: 0.36221
        Total predictions: 15000    True positives:  679        False positives:  694
        False negatives: 1321       True negatives: 12306
```

The DecisionTreeClassifier was also tried however I was never able to get both precision and recall numbers above the success criteria of 0.3. I probably need some additional preprocessing to achieve a more accurate model.

```
Pipeline(steps=[('scaler', StandardScaler(copy=True, with_mean=True, with_std=True)), ('select',
SelectKBest(k='all', score_func=<function f_classif at 0x0000000009B06EB8>)), ('decomp',
PCA(copy=True, n_components=10, whiten=False)), ('classifier',
DecisionTreeClassifier(class_weight=None, criterion='entropy...it=2,
min_weight_fraction_leaf=0.0,
            presort=False, random state=None, splitter='best'))])
        Accuracy: 0.80700  Precision: 0.29079 Recall: 0.31100    F1: 0.30056        F2: 0.30674
        Total predictions: 15000    True positives:  622        False positives: 1517
        False negatives: 1378       True negatives: 11483
```

The AdaBoostClassifier was tried but again more preprocessing would be needed to get this model performing better.

```
Pipeline(steps=[('decomp', PCA(copy=True, n_components=10, whiten=False)), ('adaboost',
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=2,
          n_estimators=10, random_state=None))])
        Accuracy: 0.69993  Precision: 0.21855 Recall: 0.48550    F1: 0.30141        F2: 0.39018
        Total predictions: 15000    True positives:  971        False positives: 3472
        False negatives: 1029       True negatives: 9528
```

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 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"]

Tuning the parameters of an algorithm refers to the process of choosing a discrete range of values for the parameter then running the algorithm for each parameter combination and returning results that display the best score from the search. During my model building and testing I had to be judicious about the number of parameters I chose because with testing with 1000 folds multiplied by a large number of candidates would make the runtimes become prohibited. Also note there are no parameters associated with the GaussianNB classifier, however with preprocessing k value for SelectKBest and n_components for PCA we do ultimately have associated parameters for this model. In contrast the DecisionTreeClassifier has 21 associated parameters. I focused on the 'pruning' parameters such as the max_depth, min_samples_leaf, and max_leaf_nodes to try and avoid overfitting.

The goal of tuning the parameters of an algorithm is to find a model that provides the best generalization of the data. If tuning is not done well such as when a model is oversimplified and results in underfitting which does not provide the best generalization. Here the errors from both the training and testing data can be high. On the other hand when a model is over complicated and results in overfitting this also does not provide the best generalization of the data and testing errors can be much greater than training errors. When tuning is done well the model provides the best generalization of the data.

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"]

Validation is the process of determining how well the model performs or fits using a specific set of criteria. A classic mistake is to validate the model using the same data which the model was trained. This will result in overfitting. It is necessary to split data into separate training and testing datasets, which allows the model to be trained on a portion of the dataset then the prediction is tested using the data that was set aside for the testing. I performed cross validation of my classifiers by withholding 10% of the data for testing. I used StratifiedShuffleSplit to cross-validate up to 1000 folds (like what was done in the tester.py). The GridSearchCV I built uses f1_score to evaluate the best estimator. The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The best estimator was then used in the tester.py to determine if the precision and recall scores were in the success range. I tried to generate my own precision and recall scores however I found it to be a challenge to match up with the tester's results, so I decided to use the tester.py as the final evaluator of the best_estimator.

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"]

Comparing the precision and recall scores from tester.py for the best estimator from GaussianNB versus the Decision Tree:

GaussianNB:

```
Precision: 0.49454     Recall: 0.33950
```

Decision Tree:

```
Precision: 0.29079     Recall: 0.31100
```

For our study about the POI in the Enron project a false positive indicates a model that has identified a person as a POI when in fact they are not, and a false negative indicates a model that identifies a person as not a POI when in fact they are a POI. Therefore the designer of the model might have a reason to prefer a result that errs on the side of identifying POI even though they are not because they would rather cast a wide net to catch all the POI and even though it means more innocent people would be inconvenienced (i.e. unnecessarily investigated). Law enforcement might decide to design such a model – specifically a model that has a high recall score.

So for the GaussianNB model the precision is 0.49 and recall is 0.33. This model meets the success criteria for both precision and recall. The precision score is higher than recall so we could conclude that this model is better predicting a person as not a POI when in fact they are a POI, however the recall score does meet the success criteria of the project as well.

The Decision Tree model has a precision score of 0.29 and recall score of 0.31. Which is a more even distribution of scores between precision and recall (with the understanding that this model does not meet the success criteria). Here the model provides a balance between identifying a person as a POI when in fact they are not and identifying a person is a non-POI when in fact they are.

A note about the sklearn version. I did a pip install of the latest version however the version was already installed. Tester.py works fine in my jupyter notebook, I'm not sure what I can do here to get tester.py to work.

```
C:\>pip freeze
beautifulsoup4==4.4.1
brewer2mpl==1.4.1
Cerberus==1.1
certifi==2017.4.17
click==6.7
comtypes==1.1.2
cycler==0.10.0
deap==1.2.2
Flask==0.12
Flask-HTTPAuth==3.2.2
geocode==0.1.0
html5lib==0.9999999
httpauth==0.3
httplib2==0.10.3
itsdangerous==0.24
Jinja2==2.9.5
MarkupSafe==0.23
matplotlib==1.5.1
nltk==3.2.5
numpy==1.13.3
pandas==0.21.0
pandasql==0.7.3
passlib==1.7.1
patsy==0.4.1
pymongo==3.2.2
pyparsing==2.1.4
python-dateutil==2.5.3
pytz==2016.6.1
requests==2.10.0
```

```
schema==0.6.6
scikit-learn==0.19.1
scipy==1.0.0
seaborn==0.7.1
six==1.10.0
SQLAlchemy==1.0.14
stopit==1.1.1
TPOT==0.9.1
tqdm==4.19.4
unicodecsv==0.14.1
update-checker==0.16
utils==0.9.0
virtualenv==15.1.0
Werkzeug==0.11.15
wincertstore==0.2
xlrd==0.9.4
xlutils==1.7.1
xlwings==0.7.2
xlwt==1.0.0
```