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 explore the Enron Dataset in order to construct a fraud detection algorithm.

In the early 2000's, Enron was one of the US' largest companies. However, after a series of investigations, the source of their 'success' was revealed and the Enron Scandal came to light, ending in one of the most notorious instances of fraud in the US. With this investigation came thousands of emails and previously private financial information being put into public record.

By combining the various pieces of financial information, emails, and a manually generated list of persons of interest (POIs) within the investigation, it should be possible to create a fraud detection algorithm. Using machine learning, we can 'sift' through the emails and compensation data to find patterns of POIs and hopefully use that to distinguish between your average Enron employee and someone to flag for investigation (if this wasn't done years after).

The dataset consists of 146 people with 19 features per person. We have information on 18 POIs that have been added into the dataset by hand. The dataset is not entirely complete with several missing values scattered across people and features. Unfilled features are denoted with 'NaN' but are replaced with '0' once transformed into an array. Most features actually had missing values, but the reasoning why is unclear. This may be due to the data actually having an equivalent value of 0 (if someone did not receive a bonus, their bonus value would essentially be 0), or because the data was removed purposefully (potentially at the request of the person). For example, only 95 people have salary information and only 111 have email addresses. To provide perspective, 14% of the people in the dataset have NaN for their total payments, but 0% of POIs have 'NaN' for their total payments, which could be problematic in using this data as missing values may be associated with non-POIs. How this is handled would depend on what data we plan to have access to in future situations. If normally we would only have access to certain features, then maybe we should rely solely on those for our algorithm.

After digging into the dataset a bit, several outliers came up in the financial data. Most of this seemed normal as the Enron Scandal had many executives with compensation packages that dwarfed that of other employees. However, there were a few outliers that seemed to stick out, even among the executives. One turned out to be the 'Total' line in the financial spreadsheet used to full some financial information in. This was determined to be not of interest for our purposes of identifying fraud, as this would throw off potential regressions. The other was an item labeled 'The Travel Agency In The Park", which although is fairly suspicious, had little information except for expenses and 'other'. After some quick searching, this was determined to be a travel agency that did business with Enron, but that wasn't of interest in the fraud case. Since this isn't an Enron employee per se, it also wouldn't be of interest for our use unless we had more instances of companies that did business with Enron and were looking to identify any suspicious activities (assuming there were any). As such, these were promptly removed while other 'normal' outliers (Enron employees with large compensations) were kept. With fraud detection, our goal is effectively is to identify outliers. POIs are usually the "top-dogs", who receive the highest compensation across various categories and who run in a selective circle within the company, apart from the rest. Otherwise, our POI list would be much longer.

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 used all original available features for the POI identifier. Because little information was available on how each feature was sourced for POIs/non-POIs, this seemed a safe option. Since the amount of data we're working with is not overwhelming, the additional cost of less important features is small. If this were not the case, I would have likely chosen 3-4 using an automated features selection function like SelectKBest. As the features varied greatly in ranges, (multimillion dollar financials and ratio of emails) I scaled all variables.

I also created two additional features to better describe and put into context a person's email habits. As per the suggestion in the Udacity lessons, rather than focus on absolute numbers of emails to/from POIs, I wanted to incorporate the proportion of a persons total emails that were to/from POIs. The effect of these added features varied across algorithms but increased performance slightly with my final SVC and were included in the final set.

Without added features:

- POI_id evaluation FINAL Classifier starts here: SVC PCA
    - SVC, all features, scaled, PCA: 1 PC, gamma = .3, c = 1

        - training time: 0.001 s
        - predicting time: 0.0 s
        - Accuracy = 0.8372093023255814
        - Precision = 0.4
        - Recall = 0.8
        - F1 = 0.5333333333333333
- Tester.py Evaluation:
    - SVC, all features, scaled, PCA: 1 PC, gamma = .3, c = 1
        - Accuracy: 0.76927
        - Precision: 0.31520
        - Recall: 0.62300
        - F1: 0.41861
        - F2: 0.52121
        - Total predictions: 15000
        - True positives: 1246
        - False positives: 2707
        - False negatives: 754
        - True negatives: 102

With added features:

- POI_id evaluation:
    - SVC, all features, scaled, PCA: 1 PC, gamma = .3, c = 1
        - training time: 0.001 s
        - predicting time: 0.0 s
        - Accuracy = 0.7906976744186046
        - Precision = 0.3333333333333333
        - Recall = 0.8
        - F1 = 0.47058823529411764
- Tester.py Evaluation:
    - SVC, all features, scaled, PCA: 1 PC, gamma = .3, c = 1
        - Accuracy: 0.75773
        - Precision: 0.32829
        - Recall: 0.78100
        - F1: 0.46227
        - F2: 0.61216
        - Total predictions: 15000
        - True positives: 1562
        - False positives: 3196
        - False negatives: 438
        - True negatives: 9804

Once features were scaled and created, I used PCA to hopefully capture any latent features driving patterns in the data and reduce the dimensionality after incorporating all available variables. As fraud is by nature deceptive and with little knowledge of various strategies one might use, this could help capture anything missed by throwing out any seemingly useless variables completely. This hopefully will also allow better performance on the algorithm with fewer inputs. I also used GridSearchCV in order to pick the number of PCs for use in the final algorithm. The optimal number was 1 PC.

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"]**

Ultimately, I chose to use an SVC as my final algorithm. I attempted many, (Gaussian NB Classifier, Decision Tree Classifier, Radius Neighbors Classifier, and a Ridge Classifier) but the SVC had the highest performance for recall without as much sacrifice in precision. I tried several iterations of each classifier (before/after: feature scaling, PCA, and a bit of parameter tuning) with the high scores actually faring similarly (examples below and in poi_id.py file). Oddly enough, the Gaussian NB had the highest performance out of the box and without much modification if any, but did not perform well with the tester.py file. Overall I'd say the models fared differently in tuning; a few fared better with more variables and some better with fewer. Below are a few of the options I tried and their performance.

- Gaussian NB model performance
  - Using all features prior to scaling/PCA:
    - Accuracy = 0.9090909090909091
    - Precision = 0.6
    - Recall = 0.6
    - F1 = 0.6

  - After scaling:
    - Accuracy = 0.36363636363636365
    - Precision = 0.12903225806451613
    - Recall = 0.8
    - F1 = 0.2222222222222222

  - Using PCA, all features, 1 PC:
    - Accuracy = 0.8837209302325582
    - Precision = 0.5
    - Recall = 0.2
    - F1 = 0.28571428571428575

- Radius Neighbors Classifier:
  - Without new features:
    - Non PCA: Radius .4, post scaling, all original features
      - Accuracy = 0.813953488372093
      - Precision = 0.3333333333333333

      - Recall = 0.6
      - F1 = 0.42857142857142855

  - With new features:
    - Non PCA: Radius .4, post scaling, all original features
      - Accuracy = 0.8604651162790697
      - Precision = 0.42857142857142855
      - Recall = 0.6
      - F1 = 0.5

- Ridge Classification:
  - With new features
    - Non-PCA Set: class_weight = 'balanced', all features
      - Accuracy = 0.7209302325581395
      - Precision = 0.18181818181818182
      - Recall = 0.4
      - F1 = 0.25000000000000006

    - PCA Set: class_weight = 'balanced', all features, 1 PC
      - Accuracy = 0.8372093023255814
      - Precision = 0.375
      - Recall = 0.6
      - F1 = 0.4615384615384615

  - Without new features
    - Non-PCA Set: class_weight = 'balanced', all features
      - Accuracy = 0.7674418604651163
      - Precision = 0.2222222222222222
      - Recall = 0.4
      - F1 = 0.2857142857142857

    - PCA Set: class_weight = 'balanced', all features, 1 PC
      - Accuracy = 0.8604651162790697
      - Precision = 0.4444444444444444
      - Recall = 0.8
      - F1 = 0.5714285714285714

- **SVC**
  - **GridSearch**
  - **{'clf__gamma': 0.3, 'clf__C': 1, 'reduce_dim__n_components': 1}**
    - **Accuracy = 0.8837209302325582**
    - **Precision = 0.5**
    - **Recall = 0.6**
    - **F1 = 0.5454545454545454**

- **Tester.py Evaluation:**
- **SVC, all features, scaled, PCA: 1 PC, gamma = .3, c = 1**
  - **Accuracy: 0.75773**
  - **Precision: 0.32829**
  - **Recall: 0.78100**
  - **F1: 0.46227**
  - **F2: 0.61216**
  - **Total predictions: 15000**
  - **True positives: 1562**
  - **False positives: 3196**
  - **False negatives: 438**
  - **True negatives: 9804**

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 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 adjusting the inputs/options of an algorithm to optimize the results to fit your needs. Many algorithms have various options that depend on the data you're working with and affect the performance. If you don't tune the algorithm, you'll likely have worse performance than you could have had otherwise. Tuning depends on both the algorithm and the data, and just like with validation, you can tune the parameters such that you overfit the data and it does not generalize well to new observations.

I tuned the parameters of the SVC first by hand, then by using GridSearchCV, which tries various options that you can specify and returns the optimal combination. I also made sure to specify that it optimize based on the F1 score rather than the Accuracy since my hope is to correctly identify most POIs with fewer amounts of False Positives with less priority on the non-POIs. This will hopefully strike a balance between identifying as many POIs as possible and minimizing False Positives. I would have used Recall, but other classifiers could easily get 100% recall with many False Positives. Reducing False Positives with fraud would allow for efficient use of resources in investigating the pinpointed POIs, who would then hopefully lead to the remaining POIs.

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"]**

Validation refers to the process of confirming your algorithm works the way it is intended to; specifically, with new data. In our case, our hope is to validate that our algorithm can identify fraud given certain inputs. We go through and check our algorithm to ensure it can do this for new data it wasn't trained on. I validated my analysis by using train_test_split to split the data into a training set (70% of the data) and testing set (30% of the data). I also used a stratified shuffle split within my GridSearchCV (with 10 splits) to further ensure that the data is split with equal ratios of POIs/non-POIs for training/testing sets.

If done wrong, you may end up over fitting the data such that it has a high performance on the training data, but low performance on any new data since it has been closely tailored to the training set.

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"]**

Evaluation Metrics using tester.py:

- Accuracy = 0.75773
  - On average, the algorithm correctly classified 75.8% of the testing dataset, meaning as we sorted employees, we correctly flagged or did not flag a person as a POI 75.8% of the time.
- Precision = 0.32829
  - Out of the total number of people flagged as a POI by our classifier, about 33% of them are actually POIs.
- Recall = 0.78100
  - Out of all of the POIs in our testing set, we 'caught' or identified about 78.1% of them.

Resources:

Enron information:
https://en.wikipedia.org/wiki/Enron
https://en.wikipedia.org/wiki/Enron_scandal
https://www.investopedia.com/updates/enron-scandal-summary/
https://www.cnn.com/2013/07/02/us/enron-fast-facts/index.html
https://www.businesstravelnews.com/More-News/Enron-s-Agency-Changes-Name-Reaffirms-Corp-Commitment

General:
https://stackoverflow.com/questions/38015181/accuracy-score-valueerror-cant-handle-mix-of-binary-and-continuous
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Radius Neighbors Classifier:
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.RadiusNeighborsClassifier.html#sklearn.neighbors.RadiusNeighborsClassifier

Ridge Classifier:
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RidgeClassifier.html

Pipelines:
https://scikit-learn.org/stable/auto_examples/compose/plot_digits_pipe.html#sphx-glr-auto-examples-compose-plot-digits-pipe-py
https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
https://scikit-learn.org/stable/tutorial/statistical_inference/putting_together.html

PCA:
https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues/2700#2700

GridSearchCV:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
https://scikit-learn.org/stable/modules/model_evaluation.html#scoring

KNN:

https://scikit-learn.org/stable/modules/neighbors.html
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.RadiusNeighborsClassifier.html#sklearn.neighbors.RadiusNeighborsClassifier

SVC:
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Evaluation:
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

Gaussian NB:
https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB

Decision Tree Classifier:
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier