1) Summarize 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 dataset consists of 146 samples, each containing 21 feature, including the target feature (POI).
Total number of target classes is 2, being True (18) and False(128).

The dataset contains lots of missing values for each feature except the target. The table of numbers of missing values for each feature:

Feature	Number of missing values
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

The goal of the project is to create end-to-end machine learning model that predicts fraudulent entries in the list of persons of interest.
Machine Learning is extremely useful in this case as it helps drastically reduce the amount of manual work required to detect frauds.

The dataset contains missing values, which is normally close to real world machine learning problems. It also contains outlier which in this case is the “Total” row. This row is not very useful for the particular task so can be removed from the dataset.

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”, “properly scale features”, “intelligently select feature”]

I’ve used scikit-learn’s SelectKBest to identify the most useful features in this case.

Person’s messaging with POI and their wealth indicators empirically shown to be the most useful, so I’ve introduced 3 new features to the dataset based on those:

- “interaction_with_poi” - ratio of person's correspondence with poi to their total correspondence
- “payments_ratio” - ratio of deferral payments to total payments
- “salary_payments_ratio” - ratio of income to total payments

One of the artificial features - salary_payments_ratio feature appeared to be useful and took a 2nd place in the SelectKBest’s rating of all features, so it was used in the final analysis. Below is the table containing impact for all of the analyzed features:

Feature	Score
exercised_stock_options	25.09754153
total_stock_value	24.46765405
salary_payments_ratio	21.0600139
bonus	21.06000171
salary	18.57570327
deferred_income	11.59554766
long_term_incentive	10.07245453
restricted_stock	9.346700791
total_payments	8.866721537
shared_receipt_with_poi	8.746485532
loan_advances	7.242730397
expenses	6.234201141
interaction_with_poi	5.518505544
from_poi_to_this_person	5.344941523
other	4.204970858
from_this_person_to_poi	2.426508127
director_fees	2.107655943
to_messages	1.698824349
payments_ratio	1.338116689
deferral_payments	0.21705893
from_messages	0.164164498
restricted_stock_deferred	0.064984312

Manual tests have shown the best results with 4 as a number of features with the following scores (default classifier hyperparameters): Accuracy: 0.84208 Precision: 0.48191 Recall: 0.35300.

The below table shows complete relation between number of features tested (based on SelectKBest) and resulting metrics:

#of features	Accuracy	Precision	Recall
2	0.84069	0.46889	0.2675
3	0.843	0.48581	0.351
4	0.84208	0.48191	0.353
5	0.83315	0.43819	0.2995
6	0.84693	0.45449	0.357
7	0.84214	0.43772	0.369
8	0.83829	0.4244	0.3705
9	0.84373	0.3937	0.3185
10	0.84107	0.38378	0.317

Note: normalization (sklearn.preprocessing.scale) was applied to the training dataset prior learning.

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 classifiers tested (default hyperparameters) and their respective precision and recall:

Classifier	Precision	Recall
GaussianNB	0.48191	0.353
LogisticRegres	0.02265	0.033
RandomFores	0.51077	0.32
MLPClassifier	0.14664	0.3515
KneighborsCla	0.80172	0.279
GradientBoost	0.38493	0.3295
AdaBoostClas	0.37784	0.3325

I've chosen to stick with the GaussianNB algorithm since it produced relatively stable precision-recall ratio and took relatively short time to compute – not really an issue with a dataset of that size but makes a model more "future-proof".

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 item: "tune the algorithm"]

Each classifier has it's own set of hyperparameters, which are parameters that are not directly learnt within estimators (http://scikit-learn.org). Depending on parameter number for a particular classifier and amount of values these parameters can take tuning hyperparameters may become a time- and resource-consuming procedure, so number of hyperparameter set should be limited to the most reasonable values, which may be deducted intuitively or through other means. However, skipping some hyperparameter sets or using step too large when performing a grid search may result in UNDER-tuning the classifier, leaving it in a state where some significant improvement of metrics might be achieved through hyperparameters.

The main goal of tuning a classifier is reaching it's state where some selected metric(s) reach the highest for the particular classifier values.

It's important to remember that tuning a classifier should be performed on a test dataset, unexposed to the classifier during training, ideally – through cross-validation (a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set). Such approach helps prevent over-FITTING, which is basically when a model becomes too complex and learns properties of a particular dataset but not the properties of the data. This may result in outstanding results on training data but poor results on the test data that was not exposed to the model during training.

I've used sklearn's built-in GridSearchCV to go through a range of possible parameters and to pick the best ones based on given scores (precision and recall). The parameter to be tuned in this case was "priors", which is Prior probabilities of the classes, as sklearn's documentation explains.
The list of probabilities I've used have length of 19 and goes from 0.05 to 0.95 with a step of .05.

Other algorithms often have more parameters and therefore require more time to cross-validate each possible parameter set.

GridSearchCV implements cross-validation based on StratifiedKFold **stratified sampling method** ***which is extremely useful in this particular case as large imbalance in the distribution of the target classes is present in the dataset. Therefore it's extremely important to maintain relative class frequencies in each train and validation fold.***

According to scikit-learn's documentation StratifiedKFold is a variation of k-fold which returns stratified folds: each set contains approximately the same percentage of samples of each target class as the complete set.

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 item: "validation strategy"]

I've used GridSearchCV to both tune parameters and validate at the same time, the possibility that makes such method so popular. It also helps avoiding some common mistakes like training and testing on the same data: cross validation assures the dataset is split into training and testing data, while using all available samples as part of a training and a testing set at a different iteration. This also helps to get a more statistically accurate results, which is extremely important while working with small 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"]

I've used precision and recall to evaluate the model as recommended in the assignment. The reason for this is priority of getting positives right. For the particular dataset, labeling fraud as positive is much more important than labeling non-fraud negative.

Precision score is intuitively classifier's ability not to produce false positives, or in other words – not positive a sample that is negative. The higher the score – the less people will be incorrectly "suspected".
Recall score is the opposite of that, meaning it represents ability of the classifier to mark positive samples as positive, correctly identifying persons of interest, which is what the task is about.