Identify Fraud from Enron Data
By Abdulrahman Alemadi

GOAL OF THE PROJECT:

The goal is to build a machine learning algorithm to identify persons of interest (suspects) of the Enron scandal. The algorithm uses employees' information as features, such as salary and stock options to identify suspects. The dataset has 146 points (persons) and each point has 21 features, mostly financial features such as salary and stock options, the other features are email related such as email from/to POI. We have 18 POI's out of 146 in the dataset.

Below represents the percentage of missing values in the dataset.

```
poi                      0.000000
total_stock_value        0.136986
total_payments           0.143836
email_address            0.239726
restricted_stock         0.246575
exercised_stock_options  0.301370
salary                   0.349315
expenses                 0.349315
other                    0.363014
from_messages            0.410959
from_this_person_to_poi  0.410959
to_messages              0.410959
shared_receipt_with_poi  0.410959
from_poi_to_this_person  0.410959
bonus                    0.438356
long_term_incentive      0.547945
deferred_income          0.664384
deferral_payments        0.732877
restricted_stock_deferred  0.876712
director_fees            0.883562
loan_advances            0.972603
```

I decided to remove the features that have more than 50% of data missing, and for the rest of the features I have decided to impute them with the mean of the feature. There was one outlier in the dataset, it was the total amount, I simply decided to remove it.

FEATURE SELECTION & ENGINEERING

I decided to feed these features into the Principal Component Analysis function. The PCA decided the most effective features that would trigger a POI prediction, then it would combine the most effective features into 1 feature, which is the first principal component (PC1) as shown in figure 1 below. The figure shows that PC1 explains 95% of the variance in the data, so we can just use PC1 in our model and drop the rest, as they do not contain much information.

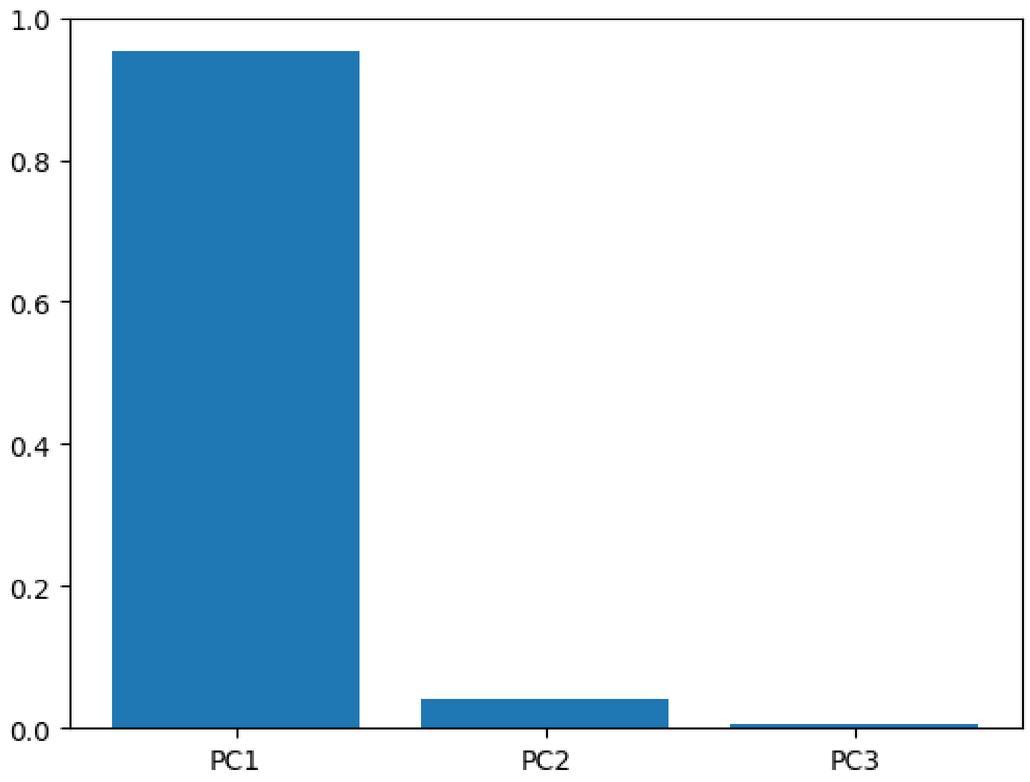

Figure 1: PCA Explained Variance Ratio

I decided not to scale the features because it has significantly decreased the AUC score of the Logistic Regression Model.

```
features_list = ['poi_contact_ratio1', 'poi_contact_ratio2','salary',
                 'to_messages', 'total_payments', 'bonus',
                 'total_stock_value', 'shared_receipt_with_poi',
                 'exercised_stock_options', 'from_messages',
                 'other', 'from_this_person_to_poi', 'expenses',
                 'restricted_stock', 'from_poi_to_this_person',
                'total_worth', 'exp/net']
```

The new features I have made were the following:

- **poi_contact_ratio1**: the number of emails sent from the person to POI **divided** by the total emails sent. I made this feature because it is important to account for how frequently does the person contact a POI, out of all emails sent.
- **poi_contact_ratio2**: the number of emails received from the POI to person **divided** by the total emails received.
- **total_worth**: the **sum** of these features 'salary', 'total_payments', 'bonus', 'total_stock_value', exercised_stock_options', 'restricted_stock'. I made this feature because I am assuming that the POI's of the scandal has a large net worth
- **exp/net**: simply, expenses **divided** by total_worth. I picked this feature from the assumption that POI's have a large proportion of spending from their net worth.

Below is a graph that indicates the importance scores of the features, the scores were obtained by using the SelectKBest function:

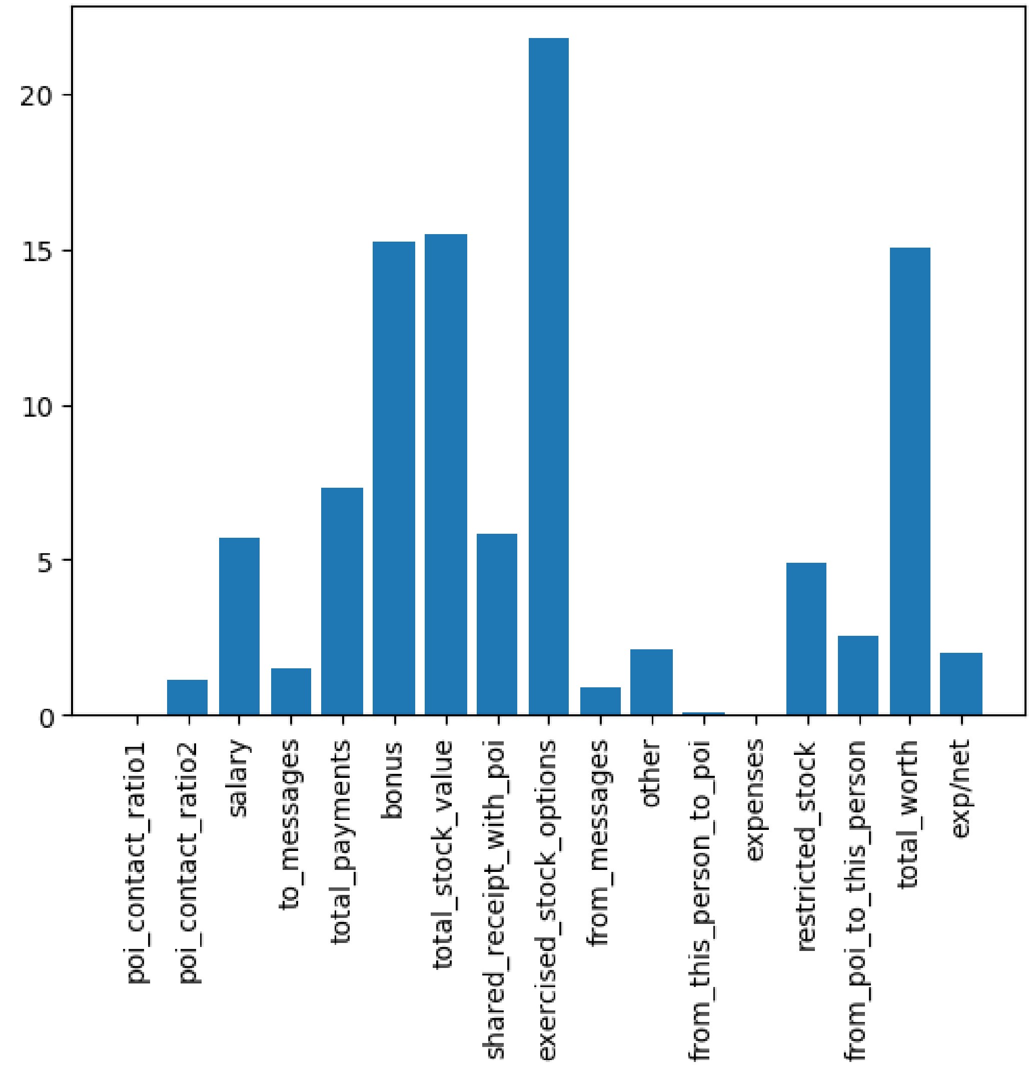

Figure 2: Feature Importance Scores

I tried to integrate SelectKBest into PCA -> LogisticRegression, but the AUC has dropped significantly to .62. Therefore, I left it out of the pipeline. I am assuming because the more features PCA has, the better.

ALGORITHM SELECTION

I decided to use logistic regression to fit and train the data. The performance metric of the models is the following:

Models	AUC	AVG F1-Score
PCA -> LogisticRegression	0.84	0.90
StandardScaler -> RandomForestClassifier	0.61	0.90
MinMaxScaler -> SelectKBest-> SVC	0.5	0.83

I had the best Area Under the Curve (AUC) results with Logistic Regression, so I have decided to use it for my final analysis.

PARAMETER TUNING

Tuning an algorithm involves trying different combination of parameters to obtain the highest yield of accuracy of the model. Tuning an algorithm is the last step before validation. Tuning is important because we want to produce a model that is fit, yet somewhat generalized to unseen data. So, it can predict most accurately throughout different datasets.

I applied GridSearchCV to fine tune my algorithm, with the following parameters:

```
params = {'LR__C': np.logspace(-10, 10, 20),
     'LR__penalty': ['l1', 'l2'],
     'LR__intercept_scaling': [1, 10 , 100, 1000,10000,1000000]}
```

The best combination of parameters is the following:

```
{'LR__intercept_scaling': 100, 'LR__penalty': 'l1', 'LR__C': 0.2976351441631313}
```

MODEL VALIDATION

Validation is used to be sure that we have a good model, a model that is not prone to overfitting. If we had overfit our model, it would only have good results on the training data which is a classic mistake of validation.

To validate my model, I have utilized tester.py, which uses the StratifiedShuffleSplit function. The StratifiedShuffleSplit function shuffles the whole data then it splits it to training and testing subsets. Then the tester fits the training data then it predicts the outcome on the testing data. The tester iterates this process 1000 times, so it minimizes the element of chance. Finally, it aggregates the performance of each iteration to give the validation stats of the 1000 iterations.

VALIDATION METRICS

Below is the validation metrics for the testing subset.

```
precision  recall  f1-score   support

  0.0      0.97     0.90     0.94       40
  1.0      0.43     0.75     0.55        4

avg / total      0.92     0.89     0.90       44

```

Precision in our case is the accuracy of predicting a POI that is truly a POI, having a low precision score means that the model has predicted innocent employees as POI’s (think false alarms). **Recall** is the percentage of the POIs are correctly identified. Having a low recall score would mean that the model has not detected the POI’s. Recall and precision are tradeoffs, in our case it would be preferable to have a higher recall score than a precision score. That way we can investigate further if the predicted persons are innocent, instead of overlooking the real POI’s. **F1-score** is the weighted average of the precision and recall, a higher f1-score simply means a better model.

When using the Tester code, I came up with these metrics:

```
Accuracy: 0.85080   Precision: 0.41793  Recall: 0.30300 F1: 0.35130 F2: 0.32063
Total predictions: 15000    True positives:  606    False positives:  844          False negatives: 1394   True negatives: 12
156
```