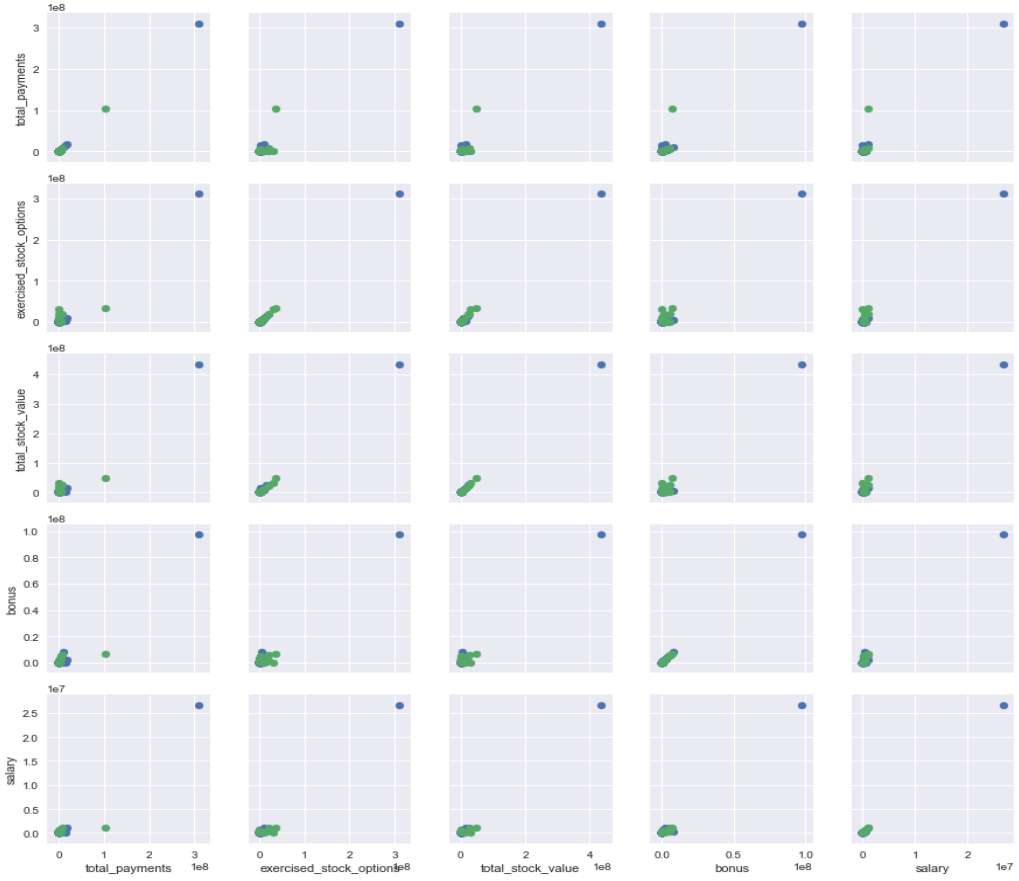
Introduction

The goal of this project is to see how Machine learning can help identify Persons of Interest (POI) in the Enron Fraud Scandal by building a person of interest identifier or model based on financial and email data made public as a result of the Enron scandal. Machine learning allows us to find hidden insights to data that are not immediately obvious. Additionally, it helps us build models that can learn and adapt to new data. In this project, we discover what financial data or email data can be used to identify POI. The POI's represent individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity. The dataset contains financial and E-mails data for a number of Enron employees; some of the financial data includes Salaries, bonus, stock options etc, I believe that some of these features can be mined by an algorithm to build a POI Identifier.

Data Set Investigation

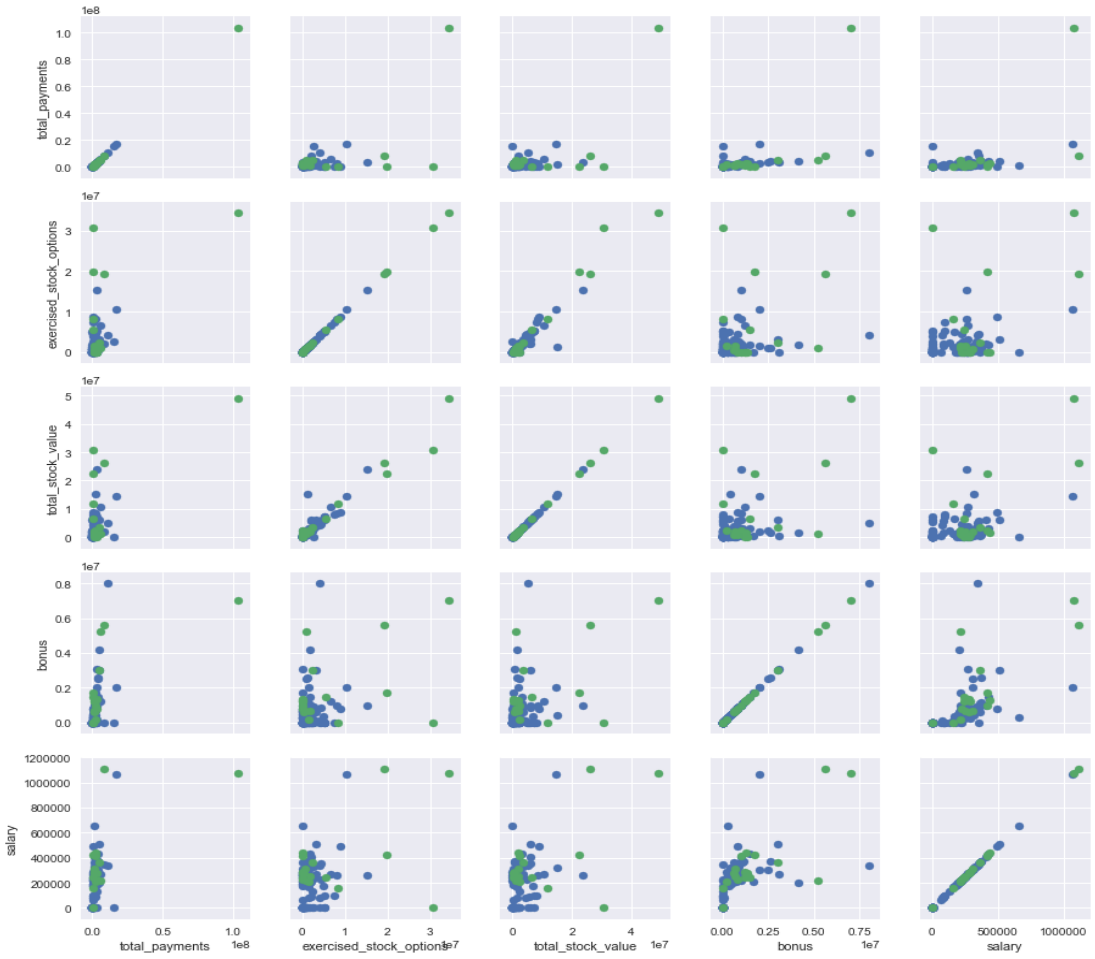
- The total number of data points in the dataset : 146
- The total number of features for each person in the dataset: 21
- The total number of POI = 18
- The total number of non-POI = 128

I had a look at financial features in the dataset to see any relations between the data.
Additionally, I had a look the min and max values of the financial data to look at the range of data and to check with features might need rescaling. I picked a few features that sounded interesting and had a quick look:



Outlier Investigation

The outlier that stands out is the Key "Total". It was the max value for each of the financial features and clearly was just the sum of the features and was removed it from the dictionary.



The remaining current maximum values for the financial data are expected as they are people from the POI list.

The other outliers that stood out were the min values for the "total_stock_value" and “restricted stock". These were negative values and belonged to ROBERT BELFAR and BHATNAGAR SANJAY respectively. It was the only negative value for the feature "total_stock_value" and ROBERT BELFAR had a lot of "Nan" values for other features as well so I removed ROBERT BELFAR from the dictionary. For restricted_stock feature, I wanted to keep other values for BHATNAGAR SANJAY; so I replaced the restricted_stock value with a NaN.

Feature selection and Engineering

Financial Features

Visually, it appeared that some features looked better than others in predicting POI for example: Exercised Stock options, Total_stock_value and bonus seemed to have good combinations that separated some POI's.

I checked the amount of Nan values for the financial Features (some are shown below) and also checked the amount of POI's that had Nan's for these values. A feature where most of the values are missing or most of the POI's don't have values won't be a very good feature.

Feature	General_Nan(%)	(POI_Nan%)
Exercised Stock options	34	33
total_stock_value	13	0
bonus	43	11.1
Salary	34	5.5
total_payments	14	0
director_fees	89	100%

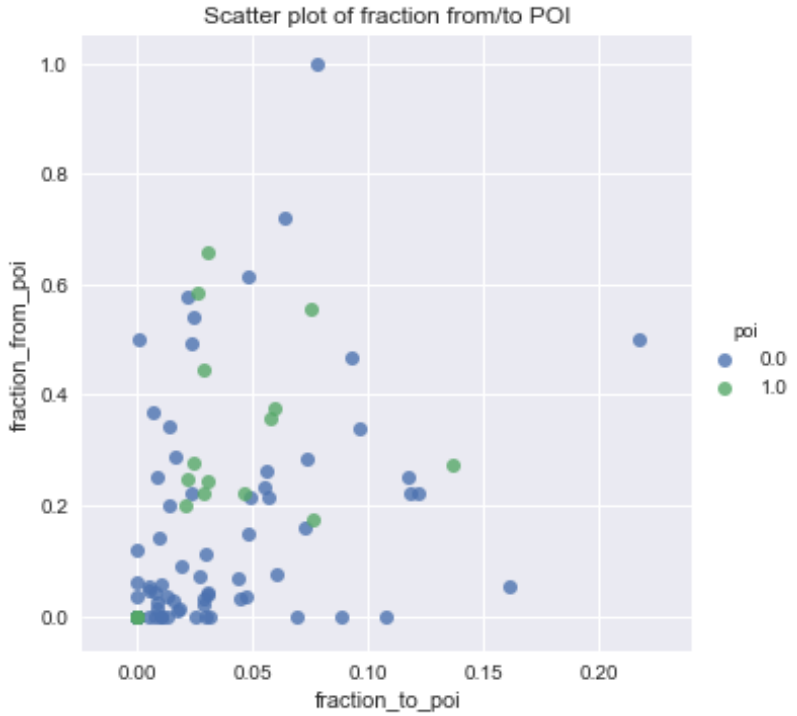
The feature which stood out was director fees, all the POI's didn't have a value for this feature, and so I won't expect to see this feature in any automated feature selection algorithms.

Mail Features

The mail features seemed the most likely place to create a new feature from. The numbers of messages from and to POI could be misleading in itself because they could merely be sharing departments or teams or projects etc. So I created two new features, which were:

'fraction_to_poi', -> from_poi_to_this_person/to_messages
'fraction_from_poi -> from this_person_to_poi/from_messages.

Looking below; most POI's have a value of 0.2 and above for fraction from POI. This will be a very good feature for identifying POI's.



I'll use automatic feature selection methods to check if any of my suppositions are correct.

Validation and Evaluation

During this Algorithm selection and tuning task, I will be trying several algorithms with different feature selection methods and different parameters. I will be using GridSearhCV to tune different parameters to get the set of parameters that produce the best estimator and I would like the parameters to be run across different sets of the data. To test the accuracy of the various models created, I will need to split the data into Training and Test sets. The test data should be 'different' from the training to ensure that the model can deal with new unseen data. Also, because we have a small dataset and about 7 times more NON_POI's than POI's, I will use the stratifiedShuffleSplit to ensure that I have a similar POI's and NON_POI's in each training and test validation set.

Furthermore, due to the high number of NON_POI's relative to POI's, accuracy would not be a good evaluation metric for the models created. For example, simply classifying everyone as NON_POI would give an accuracy of: 87% - when in fact, it has not classified any POI's correctly. A better way to evaluate the models would be to use precision and recall.

Precision: Of all the people classed as POIs, what fraction were actually POI's?
Recall: Of all the actual POI's, what fraction were classified correctly as POI's?

I think in the context of our project and for a real life situation, I would be very interested in a high recall, without sacrificing precision. The higher recall would enable us to identify POI's correctly or

people likely to commit fraud. On the other hand, we don't want to too low a precision value i.e. mislabelling people as POI when they aren't!

Feature Selection, Algorithm Selection and parameter Tuning.

I tried two automated feature_selection methods: selectKbest and feature_importances; the results are below:

Feature Selection: SelectKBest

Feature selection Method	Algorithm	Parameter	Preci	Recall	F1	Features selected	parameters
SelectKBest	GuassianNB	GridCV	0.4	0.29	0.33	bonus salary fraction_from_poi shared_receipt_with_poi	Pipeline(steps=[('feature_selection', SelectKBest(k=4, score_func=<function f_classif at 0x000000000D25B588>)), ('naive_bayes', GaussianNB(priors=None))]) Accuracy: 0.84447 Precision: 0.38937 Recall: 0.29300 F1: 0.33438 F2: 0.30826 Total predictions: 15000 True positives: 586 False positives: 919 False negatives: 1414 True negatives: 12081
SelectKBest	SVC	GridCV	0.1	0.99	0.25	bonus salary fraction_from_poi shared_receipt_with_poi	Pipeline(steps=[('feature_selection', SelectKBest(k=4, score_func=<function f_classif at 0x000000000D1D9588>)), ('my_classifier', SVC(C=0.1, cache_size=200, class_weight='balanced', coefo=0.0, decision_function_shape=None, degree=3, gamma=0.01, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))]) Accuracy: 0.24033 Precision: 0.14894 Recall: 0.99650 F1: 0.25915 F2: 0.46607 Total predictions: 15000 True positives: 1993 False positives: 11388 False negatives: 7 True negatives: 1612
SlectKBest	Decision Tree	GridCV	0.3	0.26	0.28	exercised_stock_options total_stock_value bonus salary fraction_from_poi shared_receipt_with_poi	Pipeline(steps=[('feature_selection', SelectKBest(k=6, score_func=<function f_classif at 0x000000000D204588>)), ('my_classifier', DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=5, max_features=None, max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=5, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best'))]) Accuracy: 0.82353 Precision: 0.31225 Recall: 0.26900 F1: 0.28901 F2: 0.27666 Total predictions: 15000 True positives: 538 False positives: 1185 False negatives: 1462 True negatives: 11815
SlectKBest	Decision Tree	GridCV	0.3	0.56	0.35	bonus salary fraction_from_poi shared_receipt_with_poi	Pipeline(steps=[('feature_selection', SelectKBest(k=4, score_func=<function f_classif at 0x000000000D204588>)), ('my_classifier', DecisionTreeClassifier(class_weight='balanced', criterion='entropy', max_depth=2, max_features=None, max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best'))]) Accuracy: 0.72347 Precision: 0.25558 Recall: 0.56150 F1: 0.35127 F2: 0.45304 Total predictions: 15000 True positives: 1123 False positives: 3271 False negatives: 877 True negatives: 9729
SelectKbest, Scaler	Decision Tree	GridVC	0.3	0.3	0.31	exercised_stock_options total_stock_value bonus salary fraction_from_poi shared_receipt_with_poi	Pipeline(steps=[('feature_selection', SelectKBest(k=6, score_func=<function f_classif at 0x000000000D224748>)), ('scaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('my_classifier', DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=10, max_features=None, max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=4, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best'))]) Accuracy: 0.82267 Precision: 0.32613 Recall: 0.30950 F1: 0.31760 F2: 0.31269 Total predictions: 15000 True positives: 619 False positives: 1279 False negatives: 1381 True negatives: 11721

Algorithms:

- **GaussianNB**

```
parameters = dict(feature_selection__k=[3,4,5,6],
                  naive_bayes__priors = [None])
```

Summary:

Using SelectKbest with GuassianNB and GridCV to get the best parameters gave me a good precision value but a low recall value (just below 0.3). The features selected by SelectKBest are shown in the comments section.

- **Decision Tree**

```
Parameters  = dict(feature_selection__k=[3,4,5,6],
              my_classifier__max_depth= [1 ,2, 5, 10],
              my_classifier__min_samples_split=[2, 3, 4, 5, 10],
              my_classifier__criterion = ["gini", "entropy"])
              #my_classifier__class_weight =["balanced"])
```

Summary:

Using SelectKbest with Decision Tree algorithm gave me different results depending on the class_weight parameter. Setting class_weight = balanced gave me a lower precision but higher recall value : 0.25 and 0.56 respectively than class_weight = None. Is this because it's multiplying the POI cases during training? And as such classifying more cases as POI? I did notice that the min_sample_Split was set to 2 when the class_weight was set to balanced and this might account for the low precision values due to overfitting.

Using the Decision Tree with and without the scaler and without the scaler functions gave me very similar results.

- **Support Vector Classifier**

```
parameters = dict(feature_selection__k=[3,4,5,6,7,8,9,10],
              my_classifier__C= [0.1, 1, 2, 4, 6, 8, 10],
              my_classifier__kernel=["rbf"],
              my_classifier__gamma = [0.01, 0.1, 1, 10.0, 50.0, 100.0],
              my_classifier__class_weight =["balanced"])
```

Summary:

This model got an incredibly high recall value: 0.99 but very low precision. It identified most of the POI's correctly but also mislabelled a lot of Non-POI's as POIs.

Feature Selection: Feature_importances

Feature selection Me	Algorithm	Parameter	Preci	Recall	F1	Features selected	parameters
Feaure_importances	Decision Tree	GridCV	0.4	0.32	0.34	all	DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=None, max_features=None, max_leaf_nodes=20, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best') Accuracy: 0.83653 Precision: 0.37203 Recall: 0.32850 F1: 0.34891 F2: 0.33637 Total predictions: 15000 True positives: 657 False positives: 1109 False negatives: 1343 True negatives: 11891
Feaure_importances	Decision Tree	GridCV	0.3	0.25	0.29	all	DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None, max_features=None, max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=5, min_samples_split=20, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best') Accuracy: 0.83680 Precision: 0.34926 Recall: 0.25950 F1: 0.29776 F2: 0.27356 Total predictions: 15000 True positives: 519 False positives: 967 False negatives: 1481 True negatives: 12033

Algorithms:

- **Decision Tree**

```
param_grid = {"criterion": ["gini", "entropy"],
              "min_samples_split": [2, 5,10, 20],
              "max_depth": [None, 2, 5, 10],
              "min_samples_leaf": [1, 5, 10],
              "max_leaf_nodes": [None, 5, 10, 20],
              }
```

Summary:

The feature_importances for both Models are seen below; when class weight = none, the recall values were below the required 0.3. I get better precision values with class_weight = balanced with more features being used. With class_weight set to none, only 3 features are scored as important. Please see below for selected features and feature scores.

Class weight: Balanced			Class Weight: None	
Rank	Score	Features	Rank	Score
1	0.3959423	fraction_from_poi	1	0.429905
2	0.20451553	shared_receipt_with_poi	2	0.464876
3	0.14045542	restricted_stock	3	0.105219
4	0.1015523	expenses	4	0
5	0.09980796	salary	5	0
6	0.05658157	total_payments	6	0
7	0.00114493	fraction_to_poi	7	0
8	0	bonus	8	0
9	0	long_term_incentive	9	0
10	0	exercised_stock_options	10	0
11	0	total_stock_value	11	0
12	0	director_fees	12	0

Final Algorithm: Decision Tree

I decided to take the first 4 important features from above (and other runs) and use them as my features using Decision tree as the algorithm.

Feature selection Method	Algorithm	Paramete	Prec	Recal	F1	Features selected	parameters
Manual_Best_feature_importance	Decision Tree	GridCV	0.4	0.49	0.45	"restricted_stock", "expenses", "fraction_from_poi","shared_receipt_with_poi"	DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=5, max_features=None, max_leaf_nodes=20, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best') Accuracy: 0.83960 Precision: 0.41478 Recall: 0.49400 F1: 0.45094 F2: 0.47582 Total predictions: 15000 True positives: 988 False positives: 1394 False negatives: 1012 True negatives: 11606
Manual_Best_feature_importance	Decision Tree	GridCV	0.5	0.65	0.56	"fraction_from_poi","shared_receipt_with_poi"	DecisionTreeClassifier(class_weight='balanced', criterion='entropy', max_depth=2, max_features=None, max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best') Accuracy: 0.86573 Precision: 0.49733 Recall: 0.65300 F1: 0.56463 F2: 0.61453 Total predictions: 15000 True positives: 1306 False positives: 1320 False negatives: 694 True negatives: 11680

Summary:

I get the better precision, recall and F1 values with these features and using GridCV to get the best parameters. The final features with the feature_importances are below:

Features	Rank	Score
fraction_from_poi	1	0.435942296
shared_receipt_with_poi	2	0.2279847
expenses	3	0.172646396
restricted_stock	4	0.163426608

Using the first 2 features gives a real boast to both precision, recall and F1 values. This would be my final Model chosen.