Introduction

Once upon a time Enron, "America's Most Innovative Company" filed for bankruptcy in response to a $40 billion lawsuit involving financial fraud. Today their actions have resulted in SOX requirements that every publicly traded company must abide by.

The crimes committed by Enron resulted in a trove of valuable information that I used to investigate via machine learning.

Project Goal

The outcome of this project is to identify persons of interests (POI's) that are associated to the scandal via predictive machine learning techniques; and to find unique sets of identifiable features that set these people apart from the others that were not POI's (Persons of Interest).

Parameters:

1. 3% precision and recall score.
2. Apply exploratory analysis on the financial attributes defined later.

Why Machine Learning?

Machine learning is a technique that is effective at parsing through any data-type and any size data volume. It utilizes proven statistical theory for modern day problems.

Exploring the Dataset

Question Target: Student response addresses the most important characteristics of the dataset and uses these characteristics to inform their analysis. Important characteristics include:

- total number of data points
- allocation across classes (POI/non-POI)
- number of features
- are there features with many missing values? etc.

Answer: There are 146 data points in the dataset.

Out of the 146 data points, there are 18 of them are identified as persons of interest, and the rest 128 are non-POIs.

Identify Fraud from Enron Email
Travis Seal

Each data point consists of the name of the person and 21 features. All financial features have actual number values, and all the email features except email_address feature (variable character string) have integer values.

The dataset is a dictionary of dictionaries, with the keys being the Enron employees names, and their values being a dictionary of attributes that describe them. The features available for each person are:

['salary', 'to_messages', 'deferral_payments', 'total_payments', 'exercised_stock_options', 'bonus', 'restricted_stock', 'shared_receipt_with_poi', 'restricted_stock_deferred', 'total_stock_value', 'expenses', 'loan_advances', 'from_messages', 'other', 'from_this_person_to_poi', 'poi', 'director_fees', 'deferred_income', 'long_term_incentive', 'email_address', 'from_poi_to_this_person']

In the features, the boolean variable 'POI' determines whether the person is a person of interest or not. The 'POI_COUNT' function shows that there are 18 such people in the dataset.

Outliers

Problems with the data: People in the data have missing/unknown features, and is denoted by 'NaN' (Not a number, aka null). The following issues exist:

- LOCKHART EUGENE E: No data available on this person.
- THE TRAVEL AGENCY IN THE PARK: Not an employee associated with Enron.
- TOTAL: Summation of everyone's data - likely part of a spreadsheet. Found after visualizing financial features and finding an extreme outlier.

These data will be popped from the dictionary.

Identify Fraud from Enron Email
Travis Seal

Creating The Features

In featureCreater.py, the CreatePoiEmailRatio function creates the 'poi_email_ratio' feature and the CreateExercisedStockRatio function creates the 'exercised_stock_ratio' feature. **Reasoning:** Ratios provide a means to define the relationship between the value of the target, and the space by which it lives. I can refer to this value and it scales with the surroundings proportionally.

Intelligently Select Features

Select K Best and Feature Creation **(kbest.SelectKbest(data_dict,featuresList,10))**

Lets score the features. Using selectKbest.py; I am using default value of 10 for k.

Running the algorithm produces the following output. Notice in red, the most 'interesting' attributes. The stock related attributes seem to be the leading descriptors; with email as a moderate level of interest. It was surprising to see total_payments all the way at the bottom, i figured it would have more important.

The new feature **'poi_email_ratio'**, is the sum of the ratio of emails sent to the POI over total emails sent.

I will create **'exercised_stock_ratio'** since there are two leading attributes that seem to be the strongest. This will allow me to group them together by one value.

```
[('exercised_stock_options', 24.815079733218194),

 ('total_stock_value', 24.182898678566879),

 ('bonus', 20.792252047181535),

 ('salary', 18.289684043404513),

 ('poi_email_ratio', 15.988502438971485),

 ('deferred_income', 11.458476579280369),

 ('long_term_incentive', 9.9221860131898225),

 ('restricted_stock', 9.2128106219771002),
```

('total_payments', 8.7727777300916756),

('shared_receipt_with_poi', 8.589420731682381)]

Decision Tree Classifier

Here are the results of using the DecisionTreeClassifier while changing the min_samples_split parameter over the full feature list (9 features):

With PCA (although it was not very helpful in making it more accurate)

Min Samples Split	Precision	Recall
10	0.27668	0.21000
5	0.29480	0.26650
4	0.29142	0.27350
3	0.29600	0.27350
2	0.28520	0.27650

AdaBoost (Non PCA)

Results with full feature list changing number of estimators - n_estimators:

No. of Estimators	Precision	Recall

100	0.45647	0.30150
50	0.40070	0.28450
25	0.35342	0.25800

The Adaboost Classifier performs better than decision tree classifiers using the same set of inputs. Naturally Adaboost does consume more resources, however, it does produce better results. In both cases so far non-pca produces the more accurate results.

K-Nearest Neighbors

Results while changing the number of nearest neighbors:

No. of Neighbors	Precision	Recall
1	0.15713	0.13050
2	0.27615	0.06600
3	0.50308	0.24500
4	0.58953	0.10700
5	0.63878	0.16800
6	1.000	0.004
7	0.89744	0.01750

K-Nearest neighbors gives very accurate predictions byond 5n; however, it does cost recall. It is very conservative with making predictions and thus spends more time (CPU iterations) to make accurate predictions.

Properly Scale Features

Reduced data size:

When my dataset was smaller it appeared that I could increase my accuracy. So I limited my dataset to those features that were scored the highest:

```
[
    'poi', 'exercised_stock_options', 'total_stock_value', 'bonus',
    'salary', 'deferred_income', 'long_term_incentive',
    'poi_email_ratio'
]
```

Split	**Precision**	**Recall**
2	0.34873	0.3435
3	0.35089	0.3165
4	0.33651	0.3005
5	0.33641	0.292

As a general rule of thumb, the more data points you have, the more potential for accuracy in your model. This project is no exception. When I scaled my data with MinMaxScaler using the decision tree, the following results were produced:

Accuracy: 0.86850 **Precision: 0.65202** **Recall: 0.45250** F1: 0.53424 F2: 0.48200

Total predictions: 12000 True positives: 905 False positives: 483 False negatives: 1095

There is no consistent scale by which to observe the data. Thus, it is next to impossible to derive anything meaningful. Thus I am going to create ratios.

The newly created **'poi_email_ratio'** has a maximum of two. I am going to ensure scale by using sklearn's MinMaxScaler() over all the features.

Pick an Algorithm**:** Decision Tree

Objective: Infer who will be a POI.

Target value: . >= .3

Methods:

- DecisionTreeClassifier: A standard decision tree classifier
- AdaBoost: An ensemble decision tree classifier
- K Nearest Neighbors

AdaBoost (Non PCA) vs Decision Tree:

Both produce acceptable results. However, they do have their pros and cons as shown:

AdaBoost

Precision	**Recall**

0.45647	0.30150

Vs

Decision Tree

Precision	Recall
0.65337	0.45050

The Decision Tree has better precision and higher recall ability when properly tuned. K-Nearest Neighbors also produces great precision (after 7 neighbors)

0.89744	0.01750

However, the recall ability is greatly reduced. Decision Tree is more balanced in this respect.

Final Algorithm

Parameter Tuning: Our output is only as good as the input. Data is not the only input, parameters exist to adjust the algorithm to the environment. Data does not exist in a vacuum, and algorithms account for systemic influences via parameters. This is where exploratory data analysis pays off, the more you understand the intricacies the data, the greater degree you can tune your algorithm.

Tune the algorithm

Based on the evidence of what classifiers I have tested, I have chosen to use the Decision Tree Classifier, with sample split value of 8, and an input array of:

(The report doesn't clearly mention how the final feature set was chosen)

Using Select K Best and Feature I was able to identify the most valuable attributes:

'Exercised_stock_options' was flagged as being the most important (indicated by the highest value in the list). So this will be one of the attributes I will use to build the new smaller dataset. I will be looking for people who are labeled as POI's already, so that naturally will also be added to the dataset manually. The email ratio attribute worked wonderfully and I was surprised about it being really close the value of the salary. I thought that it was exceptionally interesting because it described the relationship between the communicators on the same scale. I understand the the curse of dimensionality was likely to be present, I wanted to include attributes that specifically described the relationship of the communicators. The function below generates a list of the most valuable attributes : `doGridSearchCV()`

[('exercised_stock_options', 24.815079733218194),

('total_stock_value', 24.182898678566879),

('bonus', 20.792252047181535),

('salary', 18.289684043404513),

('poi_email_ratio', 15.988502438971492),

('deferred_income', 11.458476579280369),

('long_term_incentive', 9.9221860131898225),

('restricted_stock', 9.2128106219771002)]

I used `GridSearchCV` with select_k_best.py to determine the final featureset. This allowed me to automate the feature selection so I did not have to filter out each one.

```
def doGridSearchCV():

    from sklearn.model_selection import GridSearchCV

    import sklearn

    n_features = pd.np.arange(1, len(feature_list))

    # Create a pipeline with feature selection and classification

    from sklearn.feature_selection import SelectKBest

    pipe = Pipeline([

        ('select_features', SelectKBest()),

        ('classify', DecisionTreeClassifier())

    ])

    param_grid = [

        {

            'select_features__k': n_features

        }

    ]

    # Use GridSearchCV to automate the process of finding the optimal number of features

    tree_clf = GridSearchCV(pipe, param_grid=param_grid)

    tree_clf.fit(features, labels)

    print('Best K value param: \n',tree_clf.best_params_)

    return tree_clf.best_params_['select_features__k']
```

This classifier maintains an exceptional level of accuracy with a well balanced recall ability when tuned properly.

Accuracy: 0.86858	**Precision**: 0.65337	**Recall**: 0.45050	**F1:** 0.53329	**F2**: 0.48033
Total predictions: 12000	**True positives**: 901	**False positives**: 478	**False negatives**: 1099	**True negatives**: 9522

```
precision      recall  f1-score   support

     0.0       0.90     0.97     0.94     38

     1.0       0.50     0.20     0.29     5

avg / total    0.86     0.88     0.86     43
```

Validation and Evaluation:

Usage of evaluation:

There are four attributes that are used to evaluate the model performance:

Accuracy measures the predictions made, and what the percentage of correctly identified POIs and non-POIs are. Accuracy measures our confidence level

1. Precision measures of all the predicted POIs and what percentage of them are real POIs. The higher precision the higher level of confidence we have about the correctness of the identified POIs in data.
2. Recall measures of all the POIs and what the percentage of them are being identified as POIs.The higher recall the more we can recover more POIs
3. F1 score determines the mean of precision and recall.

Importance:
What/Why/Who Cares

The performance of the resultant algorithm is evaluated through dividing the data into pools of training/testing data.

Validation is the systematic process of inferring the model performance on one pool of data and comparing it with the other. The objective is to prevent overfitting your data, which results in inaccurate results.

please explain what can happen if it isn't done well

To me the scariest thing about working with data is getting a result that is too perfect. Data is describes the world, and the world is imperfect. A major danger is overfitting your data. This will prevent you from the goal of generalizing data in the future.

Cross validation:
StratifiedShuffleSplit divides my data into two distinct sets. The first set is used for training my model, and the second is the data to generalize.

Why use StratifiedShuffleSplit over another validation method, such as KFold cross-validation?

The objective of splitting the data is so you can judge the model's predictive power on a data set. The degree my sets overlap, will tarnish the predictive power.

If my training and validation data overlap, the outcome will be an over optimistic result.

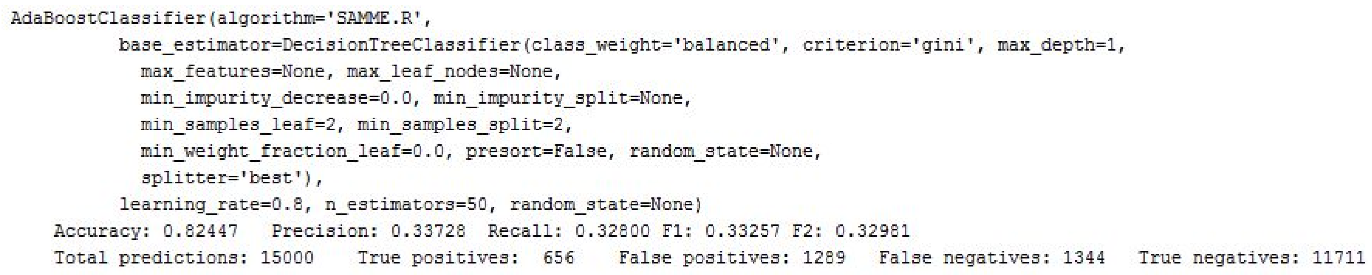
StratifiedShuffleSplit does not guarantee that there will be separation in by datasets.

Kfold does not (by default) shuffle the data.

So, it sounds counter intuitive that I should want to use a validation method that introduces some cross-pollination : the reason is to address biased results by over/under representation of the classes. It provides a more realistic representation.

Validation process (Split arrays or matrices into random train and test subsets):

Cool phrase: ensemble learning. Meaning using multiple learning are trained to solve the same problem. Using AdaBoostClassifier I can combine the results of many classifiers and produce a balanced result of **Precision: 0.33728   Recall: 0.32800**


```
AdaBoostClassifier(algorithm='SAMME.R',
          base_estimator=DecisionTreeClassifier(class_weight='balanced', criterion='gini', max_depth=1,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=2, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
          learning_rate=0.8, n_estimators=50, random_state=None)
    Accuracy: 0.82447   Precision: 0.33728  Recall: 0.32800 F1: 0.33257 F2: 0.32981
    Total predictions: 15000    True positives:  656    False positives: 1289   False negatives: 1344   True negatives: 11711
```

Evaluation ideas

An algorithm is only as useful as its accuracy; the accuracy could be expensive in terms of memory, or time. However, in order for it to be useful it must output accurate results. Thus, the total number of correct results (percentage wise) is the most critical component. The data set could have a large imbalance in labels and could easily produce too optimistic results; and mislead the interpretation.

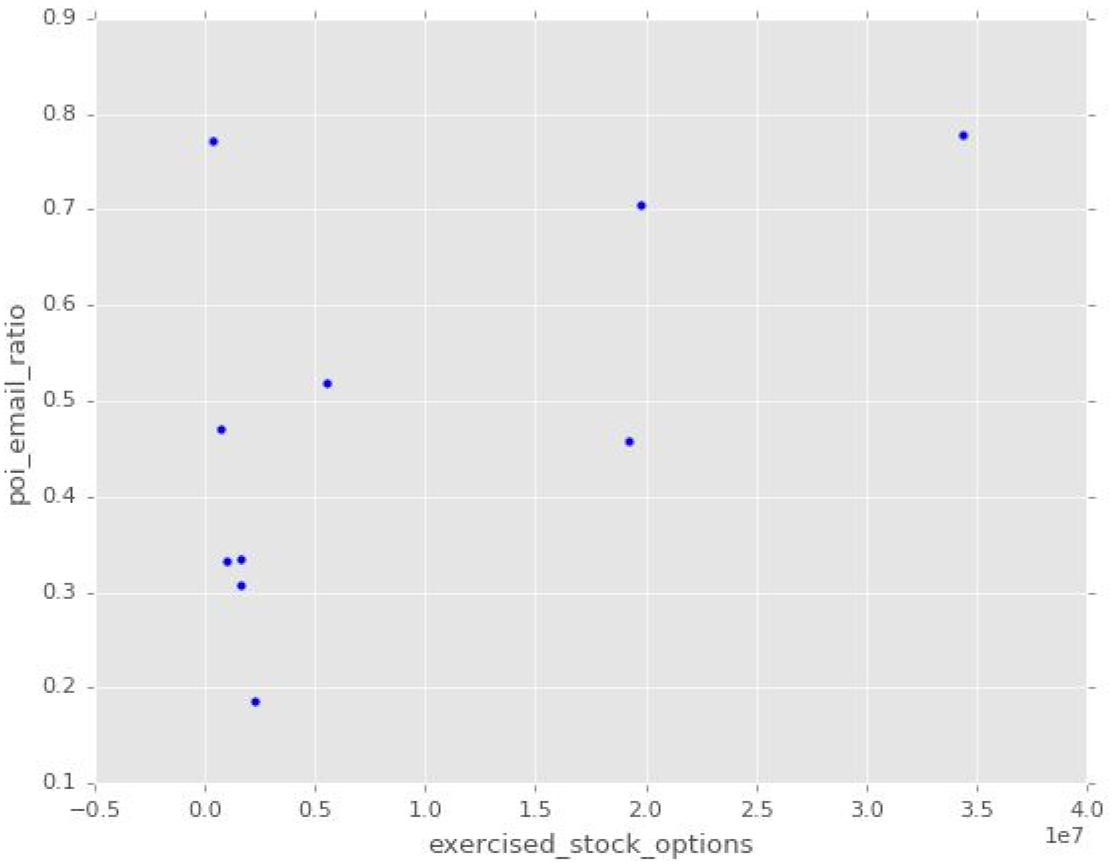
Situations like described, it is more accurate to assess the precision score / recall score.

- True positive: when a label is congruent with POI and the algorithm predicts it is a POI.
- False positive: when a label *lacks congruence* in POI and the algorithm infers it is a POI.
- True negative: when a label is congruent with POI and the algorithm inference a non-POI.
- False negative: when a label is not in agreement with the POI and the algorithms infurrance a POI.

Compared at: **Precision: 0.65130 Recall: 0.45200** when no scaling was involved. Scaling provides a means to detect more hidden patterns, but also increases the amount of ‘white noise’ in the data.

One aspect that would have been interesting to think about is natural language processing. It would be interesting to ask questions about what type language was used, what risk factors could have been identified, and build modules for Exchange Server that auto detects potential ‘fraud language’.

And here are some visualizations I generated:



Final Feature Selection

The final 9 features to be used for the machine learning process are:

[(**'exercised_stock_options'**, 24.815079733218194),

(**'total_stock_value'**, 24.182898678566879),

('bonus', 20.792252047181535),

('salary', 18.289684043404513),

(**'poi_email_ratio**', 15.988502438971492),

('deferred_income', 11.458476579280369),

('long_term_incentive', 9.9221860131898225),

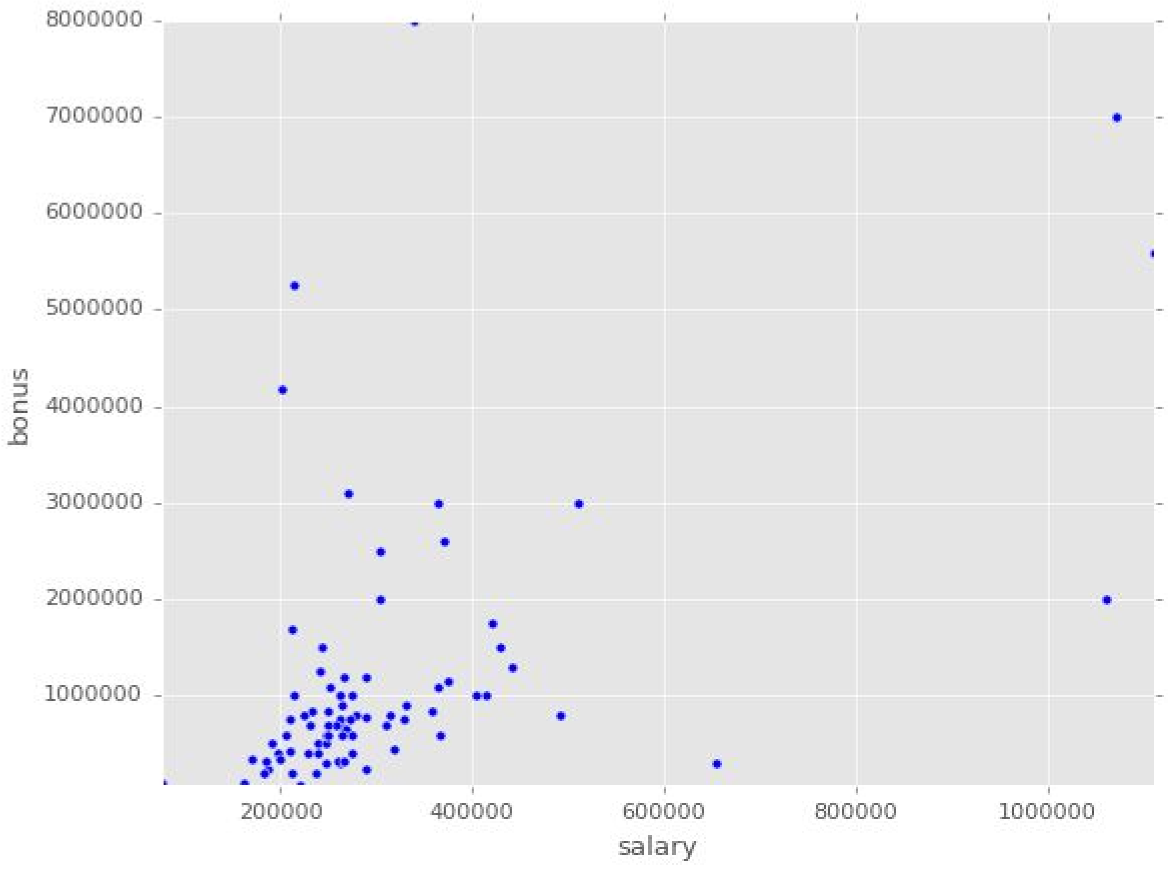
```
 ('restricted_stock', 9.2128106219771002),

 ('total_payments', 8.7727777300916827))]
```

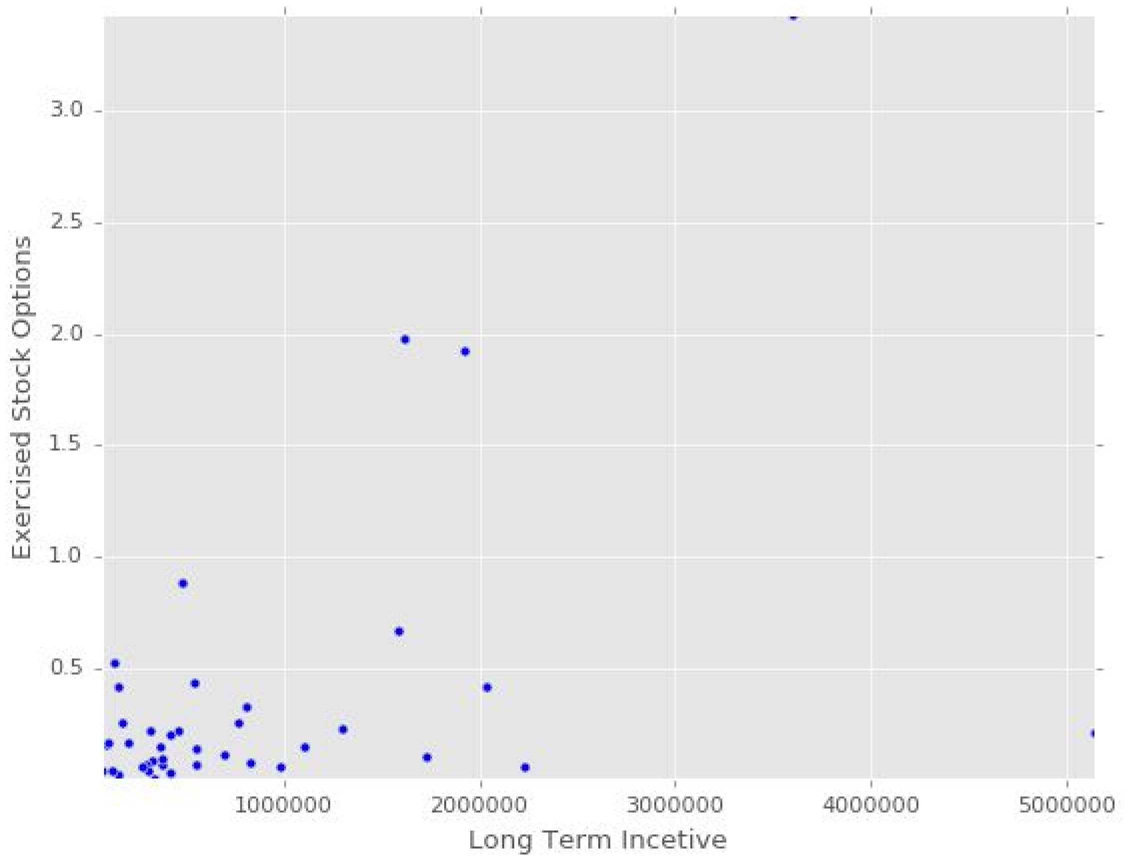
Visualizing Data

This graph show the relationship between the salary and the bonus amount. There are some really extreme values here, like

Salary vs Bonus (doPlotSalaryAndBonus function)



Exercised Stock Options vs Long-term Incentive (doExercisedStockOptionsLongTermIncentive())


Exercised Stock Options
Long Term Incetive