In [8]:
```
#!/usr/bin/python


"""
    Starter code for the evaluation mini-project.
    Start by copying your trained/tested POI identifier from
    that which you built in the validation mini-project.

    This is the second step toward building your POI identifier!

    Start by loading/formatting the data...
"""

import pickle
import sys
sys.path.append("../tools/")
from feature_format import featureFormat, targetFeatureSplit

data_dict = pickle.load(open("../final_project/final_project_dataset.pkl", "r") )

### add more features to features_list!
features_list = ["poi", "salary"]

data = featureFormat(data_dict, features_list)
labels, features = targetFeatureSplit(data)
```

Go back to your code from the last lesson, where you built a simple first iteration of a POI identifier using a decision tree and one feature. Copy the POI identifier that you built into the skeleton code in evaluation/evaluate_poi_identifier.py. Recall that at the end of that project, your identifier had an accuracy (on the test set) of 0.724. Not too bad, right? Let's dig into your predictions a little more carefully.

From Python 3.3 forward, a change to the order in which dictionary keys are processed was made such that the orders are randomized each time the code is run. This will cause some compatibility problems with the graders and project code, which were run under Python 2.7. To correct for this, add the following argument to the featureFormat call on line 25 of evaluate_poi_identifier.py:

sort_keys = '../tools/python2_lesson14_keys.pkl'

This will open up a file in the tools folder with the Python 2 key order.

In [9]:
```
### your code goes here
from sklearn.tree import DecisionTreeClassifier as dtc
from sklearn import cross_validation

features_train, features_test, labels_train, labels_test = cross_validation.train
clf = dtc().fit(features_train, labels_train)
print "Accuracy: ",clf.score(features_test,labels_test)
```

```
Accuracy:  0.7241379310344828
```

How many POIs are predicted for the test set for your POI identifier?

(Note that we said test set! We are not looking for the number of POIs in the whole dataset.)

In [10]:
```
count = 0
for x in clf.predict(features_test):
    count += x
print "POIs: ", count
```

```
POIs:  4.0
```

How many people total are in your test set?

In [11]:
```
print len(features_test)
```

```
29
```

If your identifier predicted 0. (not POI) for everyone in the test set, what would its accuracy be?

.86

Look at the predictions of your model and compare them to the true test labels.

Do you get any true positives?

(In this case, we define a true positive as a case where both the actual label and the predicted label are 1)

In [14]:
```
pred = clf.predict(features_test)
count = 0
for x in range(len(features_test)):
    if (pred[x]==labels_test[x]) and (pred[x]==1):
        count += 1
print count
```

```
0
```

As you may now see, having imbalanced classes like we have in the Enron dataset (many more non-POIs than POIs) introduces some special challenges, namely that you can just guess the more common class label for every point, not a very insightful strategy, and still get pretty good accuracy!

Precision and recall can help illuminate your performance better. Use the *precision_score* and *recall_score* available in *sklearn.metrics* to compute those quantities.

What's the precision?

```
In [15]: from sklearn.metrics import precision_score, recall_score
         print "Precision: ",precision_score(labels_test,pred)
         print "Recall: ",recall_score(labels_test,pred)
```

```
Precision:  0.0
Recall:  0.0
```

In the final project you'll work on optimizing your POI identifier, using many of the tools learned in this course. Hopefully one result will be that your precision and/or recall will go up, but then you'll have to be able to interpret them.

Here are some made-up predictions and true labels for a hypothetical test set; fill in the following boxes to practice identifying true positives, false positives, true negatives, and false negatives. Let's use the convention that "1" signifies a positive result, and "0" a negative.

predictions = [0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1]

true labels = [0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0]

How many true positives are there?

6

How many true negatives are there in this example?

9

How many false positives are there?

3

How many false negatives are there?

2

What's the precision of this classifier?

.667

What's the recall of this classifier?

.75

Fill in the blank:

“My true positive rate is high, which means that when a ***POI*** is present in the test data, I am good at flagging him or her.”

“My identifier doesn’t have great ***precision***, but it does have good ***recall***. That means that, nearly every time a POI shows up in my test set, I am able to identify him or her. The cost of this is that I sometimes get some false positives, where non-POIs get flagged.”

“My identifier doesn’t have great ***recall***, but it does have good ***precision***. That means that whenever a POI gets flagged in my test set, I know with a lot of confidence that it’s very likely to be a real POI and not a false alarm. On the other hand, the price I pay for this is that I sometimes miss real POIs, since I’m effectively reluctant to pull the trigger on edge cases.”

“My identifier has a really great ***F1 score***.

This is the best of both worlds. Both my false positive and false negative rates are ***low***, which means that I can identify POI’s reliably and accurately. If my identifier finds a POI then the person is almost certainly a POI, and if the identifier does not flag someone, then they are almost certainly not a POI.”

```
In [ ]:
```