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

"""
    Starter code for the regression mini-project.

    Loads up/formats a modified version of the dataset
    (why modified?  we've removed some trouble points
    that you'll find yourself in the outliers mini-project).
    Draws a little scatterplot of the training/testing data
    You fill in the regression code where indicated:
"""



import sys
import pickle
sys.path.append("../tools/")
from feature_format import featureFormat, targetFeatureSplit
dictionary = pickle.load( open("../final_project/final_project_dataset_modified.p

### list the features you want to look at--first item in the
### list will be the "target" feature
features_list = ["bonus", "salary"]
data = featureFormat( dictionary, features_list, remove_any_zeroes=True)
target, features = targetFeatureSplit( data )

### training-testing split needed in regression, just like classification
from sklearn.cross_validation import train_test_split
feature_train, feature_test, target_train, target_test = train_test_split(feature
train_color = "b"
test_color = "b"
```

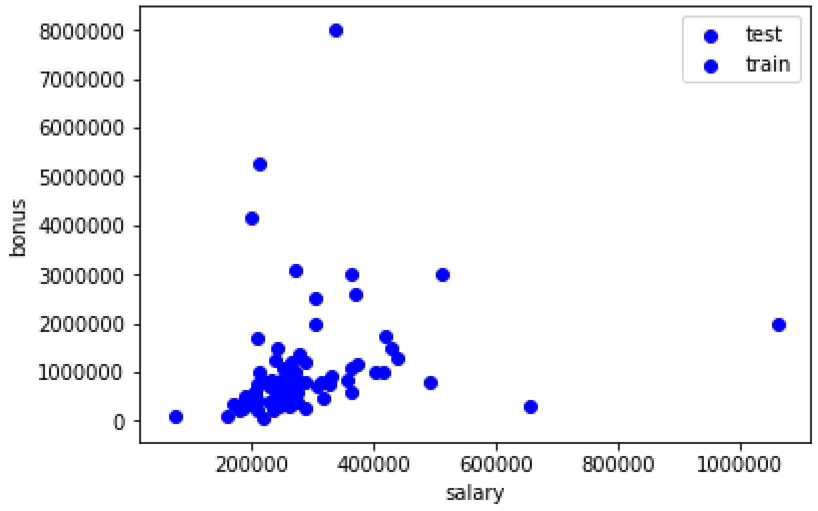
Run the starter code found in regression/finance_regression.py. This will draw a scatterplot, with all the data points drawn in. What target are you trying to predict? What is the input feature being used to predict it?

Target: bonus

Input: salary

Mentally (or better yet, print out the scatterplot and use paper and pencil) sketch out the regression line that you roughly predict.

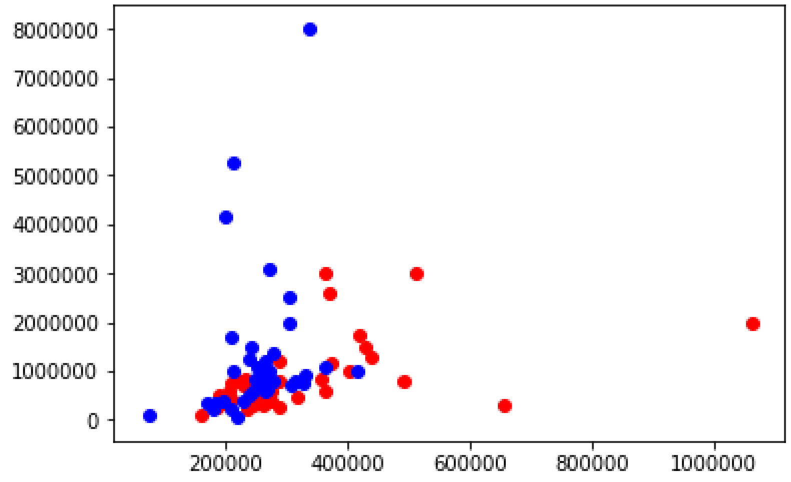
In [2]:
```
### draw the scatterplot, with color-coded training and testing points
import matplotlib.pyplot as plt
for feature, target in zip(feature_test, target_test):
    plt.scatter( feature, target, color=test_color )
for feature, target in zip(feature_train, target_train):
    plt.scatter( feature, target, color=train_color )

### labels for the legend
plt.scatter(feature_test[0], target_test[0], color=test_color, label="test")
plt.scatter(feature_test[0], target_test[0], color=train_color, label="train")
```



In regression, you need training and testing data, just like in classification. This has already been set up in the starter code. Change the value of test_color from "b" to “r” (for “red”), and rerun. Note: For those students converting Python 2 code to Python 3, see below for some important remarks regarding compatibility.

You will be fitting your regression using the blue (training) points only. (You may have noticed that instead of the standard 10%, we’ve put 50% of our data into the test set--that’s because in Part 5, we will switch the training and testing datasets around and splitting the data evenly makes this more straightforward.)

In [4]:
```
test_color = "r"
### draw the scatterplot, with color-coded training and testing points
import matplotlib.pyplot as plt
for feature, target in zip(feature_test, target_test):
    plt.scatter( feature, target, color=test_color )
for feature, target in zip(feature_train, target_train):
    plt.scatter( feature, target, color=train_color )
```

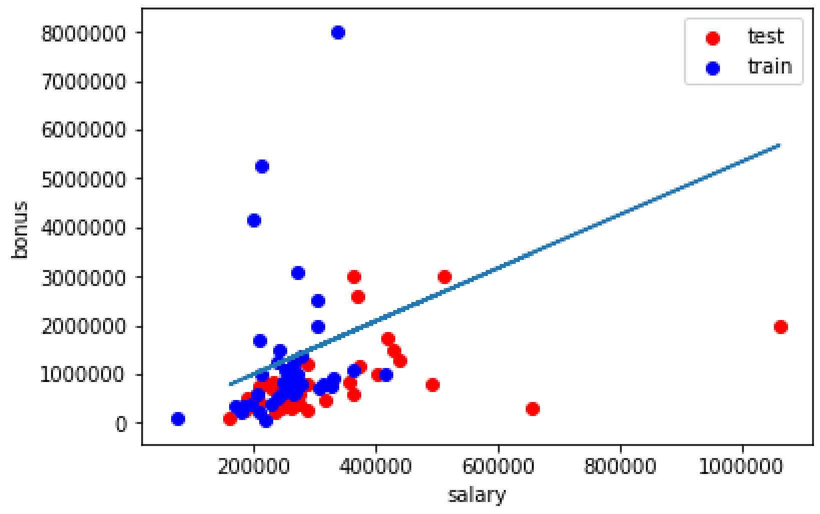


Import LinearRegression from sklearn, and create/fit your regression. Name it reg so that the plotting code will show it overlaid on the scatterplot. Does it fall approximately where you expected it?

Extract the slope (stored in the reg.coef_ attribute) and the intercept. What are the slope and intercept?

In [6]:
```
from sklearn.linear_model import LinearRegression as lr
reg = lr()
reg.fit(feature_train,target_train)

print "coef ",reg.coef_
print "inte ",reg.intercept_

### draw the scatterplot, with color-coded training and testing points
import matplotlib.pyplot as plt
for feature, target in zip(feature_test, target_test):
    plt.scatter( feature, target, color=test_color )
for feature, target in zip(feature_train, target_train):
    plt.scatter( feature, target, color=train_color )

### labels for the legend
plt.scatter(feature_test[0], target_test[0], color=test_color, label="test")
plt.scatter(feature_test[0], target_test[0], color=train_color, label="train")

### draw the regression line, once it's coded
try:
    plt.plot( feature_test, reg.predict(feature_test) )
except NameError:
    pass
plt.xlabel(features_list[1])
plt.ylabel(features_list[0])
plt.legend()
plt.show()
```

```
coef  [5.44814029]
inte  -102360.5432938796
```



Imagine you were a less savvy machine learner, and didn’t know to test on a holdout test set. Instead, you tested on the same data that you used to train, by comparing the regression predictions to the target values (i.e. bonuses) in the training data. What score do you find? You may not have an intuition yet for what a “good” score is; this score isn’t very good (but it could be a lot worse).

In [8]:
```
reg.score(feature_train,target_train)
```
Out[8]:
```
0.04550919269952436
```

Now compute the score for your regression on the test data, like you know you should. What's that score on the testing data? If you made the mistake of only assessing on the training data, would you overestimate or underestimate the performance of your regression?

In [9]:
```
reg.score(feature_test,target_test)
```
Out[9]:
```
-1.48499241736851
```

There are lots of finance features available, some of which might be more powerful than others in terms of predicting a person's bonus. For example, suppose you thought about the data a bit and guess that the "long_term_incentive" feature, which is supposed to reward employees for contributing to the long-term health of the company, might be more closely related to a person's bonus than their salary is.

A way to confirm that you're right in this hypothesis is to regress the bonus against the long term incentive, and see if the regression score is significantly higher than regressing the bonus against the salary. Perform the regression of bonus against long term incentive--what's the score on the test data?

In [10]:
```
features_list = ["bonus", "long_term_incentive"]
data = featureFormat( dictionary, features_list, remove_any_zeroes=True)
target, features = targetFeatureSplit( data )

feature_train, feature_test, target_train, target_test = train_test_split(feature

from sklearn.linear_model import LinearRegression as lr
reg = lr()
reg.fit(feature_train,target_train)

reg.score(feature_test,target_test)
```
Out[10]:
```
-0.5927128999498639
```

If you had to predict someone's bonus and you could only have one piece of information about them, would you rather know their salary or the long term incentive that they received?

LTI

In [ ]:
```
```