1. **Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those? [relevant rubric items: "data exploration", "outlier investigation"]**

The question that we need to answer in the project is, "Based on the public Enron financial and email dataset, which Enron employees may have committed fraud"

The dataset was from the federal investigation of Enron. It includes tens of thousands of emails and detailed financial data for top executives.

Dataset Info:

a. total number of data point.

   There are 144 data points in the dataset

b. poi vs non-poi

   poi = 18, non-poi: 144-18=126

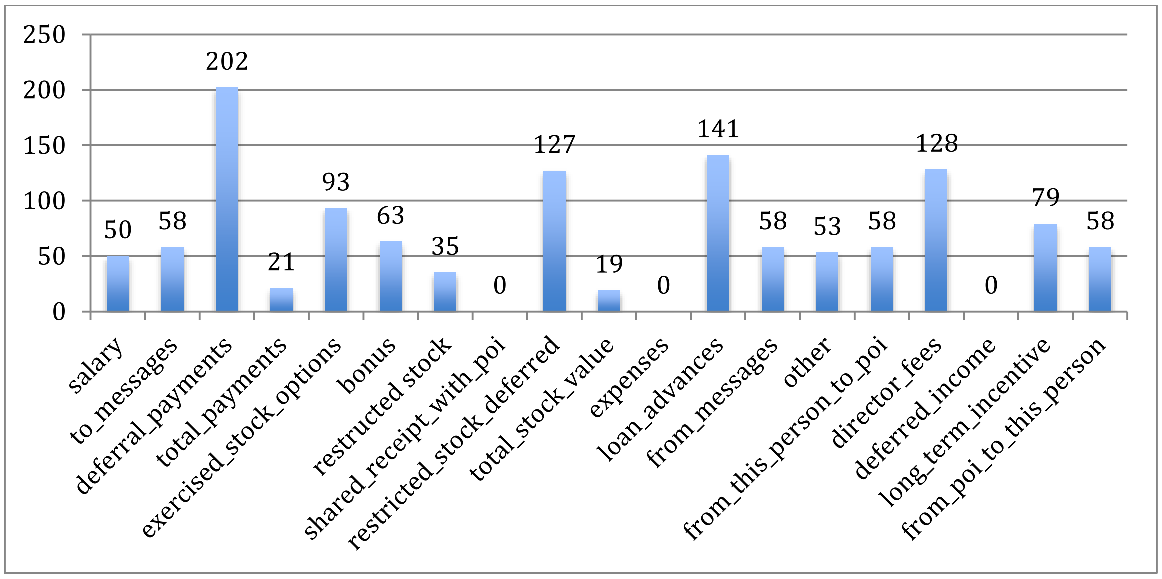
c. Number of feature used

   In my final analysis, I ended up using 9 features.

   features_list = ['poi', 'deferral_payments', 'total_payments',
   'exercised_stock_options',
   'restricted_stock','restricted_stock_deferred', 'expenses', 'director_fees',
   'deferred_income']

   Using the above 9 features gave me the best precision/recall rate. See more detail on the feature selection process below

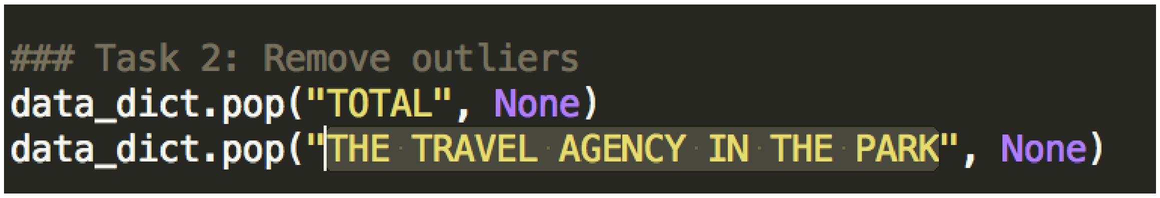
d. Features with missing values



There were two areas of outliers that I found.

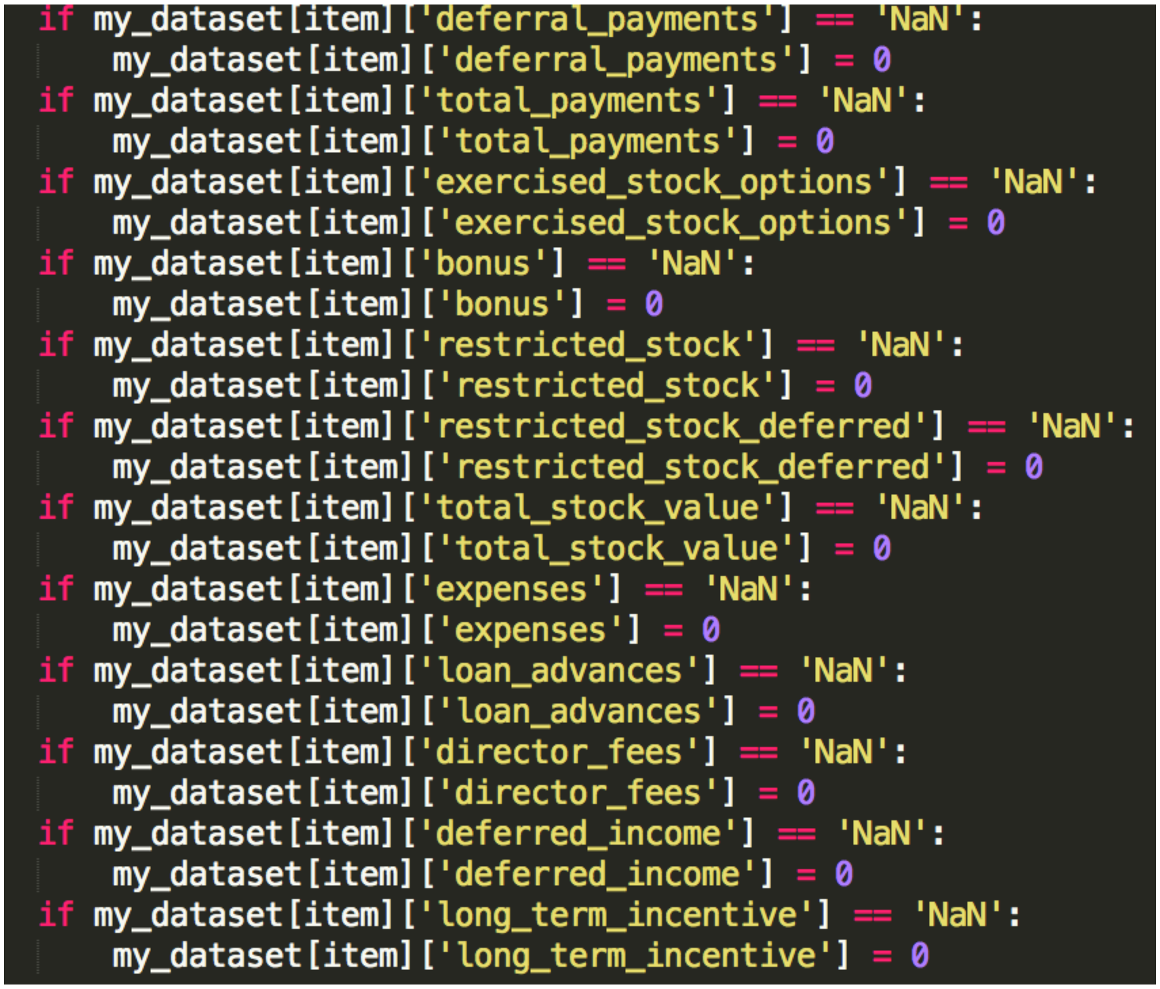
a. dictionary keys

The following keys were found in the dictionary, 'TOTAL' and 'THE TRAVEL AGENCY IN THE PARK'. There were later removed from the dictionary


```
### Task 2: Remove outliers
data_dict.pop("TOTAL", None)
data_dict.pop("THE TRAVEL AGENCY IN THE PARK", None)
```

b. 'NaN' values

Many values were assigned with 'NaN' values, including values that should be in numerical values. For numerical values, I then reassigned them to 0 instead of 'NaN'



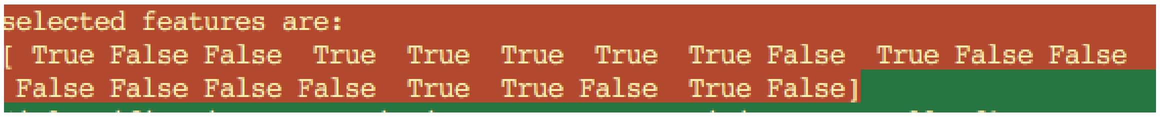
```
if my_dataset[item]['deferral_payments'] == 'NaN':
    my_dataset[item]['deferral_payments'] = 0
if my_dataset[item]['total_payments'] == 'NaN':
    my_dataset[item]['total_payments'] = 0
if my_dataset[item]['exercised_stock_options'] == 'NaN':
    my_dataset[item]['exercised_stock_options'] = 0
if my_dataset[item]['bonus'] == 'NaN':
    my_dataset[item]['bonus'] = 0
if my_dataset[item]['restricted_stock'] == 'NaN':
    my_dataset[item]['restricted_stock'] = 0
if my_dataset[item]['restricted_stock_deferred'] == 'NaN':
    my_dataset[item]['restricted_stock_deferred'] = 0
if my_dataset[item]['total_stock_value'] == 'NaN':
    my_dataset[item]['total_stock_value'] = 0
if my_dataset[item]['expenses'] == 'NaN':
    my_dataset[item]['expenses'] = 0
if my_dataset[item]['loan_advances'] == 'NaN':
    my_dataset[item]['loan_advances'] = 0
if my_dataset[item]['director_fees'] == 'NaN':
    my_dataset[item]['director_fees'] = 0
if my_dataset[item]['deferred_income'] == 'NaN':
    my_dataset[item]['deferred_income'] = 0
if my_dataset[item]['long_term_incentive'] == 'NaN':
    my_dataset[item]['long_term_incentive'] = 0
```

2. What features did you end up using in your POI identifier, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not? As part of the assignment, you should attempt to engineer your own feature that does not come ready-made in the dataset -- explain what feature you tried to make, and the rationale behind it. (You do not necessarily have to use it in the final analysis, only engineer and test it.) In your feature selection step, if you used an algorithm like a decision tree, please also give the feature importances of the features that you use, and if you used an automated feature selection function like SelectKBest, please report the feature scores and reasons for your choice of parameter values. [relevant rubric items: "create new features", "properly scale features", "intelligently select feature"]

<u>Feature Selection Process</u>:

SelectKBest was used to pick the most important features to use.

By calling selectkbest "get_support" method, the following list is returned:



```
selected features are:
[ True False False  True  True  True  True  True False  True False False
 False False False False  True  True False  True False]
```

Mapping this back to my feature list implies that the following features were selected:

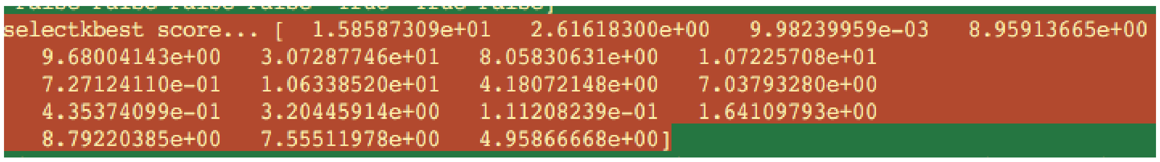
['**salary**', '**total_payments**', '**exercised_stock_options**', '**bonus**', '**restricted_stock**', '**shared_receipt_with_poi**', '**total_stock_value**', '**loan_advances**'   '**deferred_income**', '**long_term_incentive**',]

provide performance scores returned for the features and justify the choice of K (number of features):

I used "pipe.named_steps['selectkbest']" to pick the best stage. K = 10 was returned . Therefore, 10 is the best number picked .

	Precision	**Recall**
K = 14	0.361	0.314
K = 13	0.361	0.314
K = 12	0.365	0.313
K = 10	0.379	0.308

Selectkbest scores:



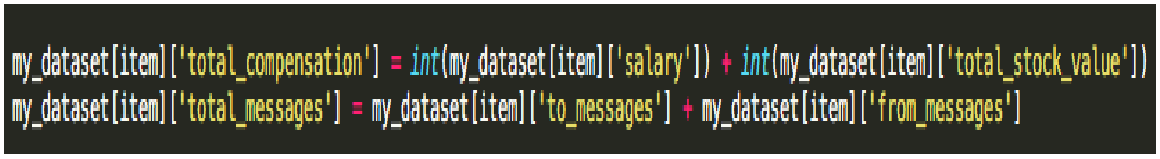
```
selectkbest score... [  1.58587309e+01   2.61618300e+00   9.98239959e-03   8.95913665e+00
   9.68004143e+00   3.07287746e+01   8.05830631e+00   1.07225708e+01
   7.27124110e-01   1.06338520e+01   4.18072148e+00   7.03793280e+00
   4.35374099e-01   3.20445914e+00   1.11208239e-01   1.64109793e+00
   8.79220385e+00   7.55511978e+00   4.95866668e+00]
```

Salary	1.58587309e+01
To_messages	2.61618300e+00
Deferral_payments	9.98239959e-03
Total_payments	8.95913665e+00
Exercised_stock_options	9.68004143e+00
Bonus	3.07287746e+01
Restricted_stock	8.05830631e+00
Shared_receipt_with_poi	1.07225708e+01
Restricted_stock_deferred	7.27124110e-01
Total_stock_value	1.06338520e+01
Expenses	4.18072148e+00
Loan_advances	7.03793280e+00
From_messages	4.35374099e-01
other	3.20445914e+00

From_this_person_to_poi	1.11208239e-01
Director_fees	1.64109793e+00
Deferred_income	8.79220385e+00
Long_term_incentive	7.55511978e+00
From_poi_to_this_person	4.95866668e+00

I did not use any scaling because I ended up using AdaBoost and not SVC. For SVC, MinMaxScaler() is very important to get the features scaled before feeding it into the SVC classifier.

For New Features, I created two additional features, total_messages and total_compensation



```
my_dataset[item]['total_compensation'] = int(my_dataset[item]['salary']) + int(my_dataset[item]['total_stock_value'])
my_dataset[item]['total_messages'] = my_dataset[item]['to_messages'] + my_dataset[item]['from_messages']
```

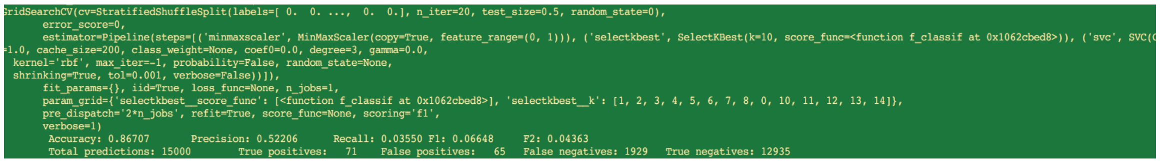
the rationale behind this is to gauge if total value for compensation and messages would be a better indicator than the separated values. I did not use the new features at the end because the algorithm I used could achieve precision and recall over 0.3, without using the new features

	Precision	**recall**
With total_compensation and total_messages added	0.34	0.28
without	**0.35**	**0.30**

3. What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms? [relevant rubric item: “pick an algorithm”]

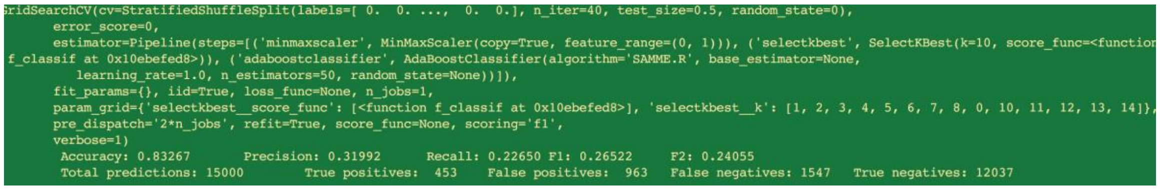
I ended up using GuassianNB classifier since it had the best Precision and recall result (**Precision 0.365 recall 0.312).** I’ve also tried several different others with worse performance. Here are the results of different models:

a. SVC – **Precision 0.32 recall 0.22**



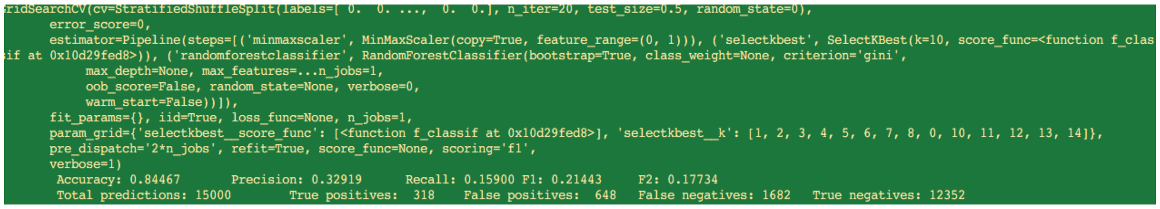
```
GridSearchCV(cv=StratifiedShuffleSplit(labels=[ 0.  0. ...,  0.  0.], n_iter=20, test_size=0.5, random_state=0),
       error_score=0,
       estimator=Pipeline(steps=[('minmaxscaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('selectkbest', SelectKBest(k=10, score_func=<function f_classif at 0x1062cbed8>)), ('svc', SVC(C
=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0,
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False))]),
       fit_params={}, iid=True, loss_func=None, n_jobs=1,
       param_grid={'selectkbest__score_func': [<function f_classif at 0x1062cbed8>], 'selectkbest__k': [1, 2, 3, 4, 5, 6, 7, 8, 0, 10, 11, 12, 13, 14]},
       pre_dispatch='2*n_jobs', refit=True, score_func=None, scoring='f1',
       verbose=1)
        Accuracy: 0.86707       Precision: 0.52206      Recall: 0.03550 F1: 0.06648     F2: 0.04363
        Total predictions: 15000        True positives:   71    False positives:   65   False negatives: 1929   True negatives: 12935
```

b. AdaBoost - **Precision 0.32 recall 0.22**



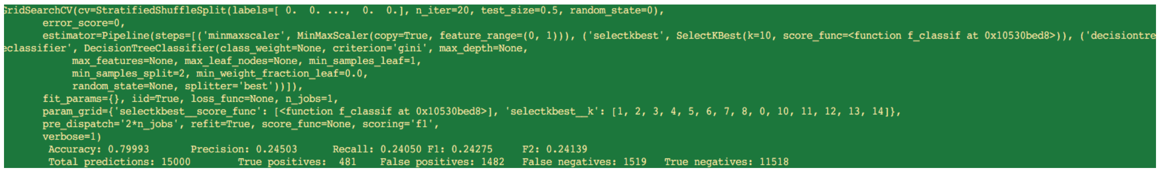
```
GridSearchCV(cv=StratifiedShuffleSplit(labels=[ 0.  0. ...,  0.  0.], n_iter=40, test_size=0.5, random_state=0),
       error_score=0,
       estimator=Pipeline(steps=[('minmaxscaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('selectkbest', SelectKBest(k=10, score_func=<function
 f_classif at 0x10ebefed8>)), ('adaboostclassifier', AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=50, random_state=None))]),
       fit_params={}, iid=True, loss_func=None, n_jobs=1,
       param_grid={'selectkbest__score_func': [<function f_classif at 0x10ebefed8>], 'selectkbest__k': [1, 2, 3, 4, 5, 6, 7, 8, 0, 10, 11, 12, 13, 14]},
       pre_dispatch='2*n_jobs', refit=True, score_func=None, scoring='f1',
       verbose=1)
        Accuracy: 0.83267       Precision: 0.31992      Recall: 0.22650 F1: 0.26522     F2: 0.24055
        Total predictions: 15000        True positives:  453    False positives:  963   False negatives: 1547   True negatives: 12037
```

c. RandomForest – **Precision 0.33, recall 0.16**



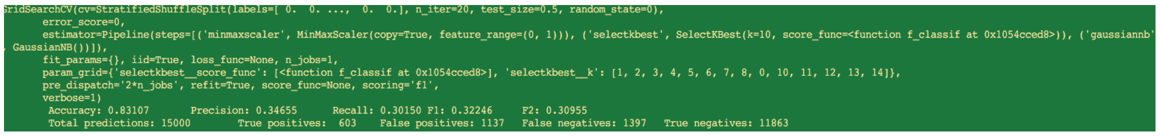
```
GridSearchCV(cv=StratifiedShuffleSplit(labels=[ 0.  0. ...,  0.  0.], n_iter=20, test_size=0.5, random_state=0),
       error_score=0,
       estimator=Pipeline(steps=[('minmaxscaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('selectkbest', SelectKBest(k=10, score_func=<function f_clas
sif at 0x10d29fed8>)), ('randomforestclassifier', RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features=...n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False))]),
       fit_params={}, iid=True, loss_func=None, n_jobs=1,
       param_grid={'selectkbest__score_func': [<function f_classif at 0x10d29fed8>], 'selectkbest__k': [1, 2, 3, 4, 5, 6, 7, 8, 0, 10, 11, 12, 13, 14]},
       pre_dispatch='2*n_jobs', refit=True, score_func=None, scoring='f1',
       verbose=1)
        Accuracy: 0.84467       Precision: 0.32919      Recall: 0.15900 F1: 0.21443     F2: 0.17734
        Total predictions: 15000        True positives:  318    False positives:  648   False negatives: 1682   True negatives: 12352
```

d. DecisionTree – **Precision 0.24, Recall 0.24**



```
GridSearchCV(cv=StratifiedShuffleSplit(labels=[ 0.  0. ...,  0.  0.], n_iter=20, test_size=0.5, random_state=0),
       error_score=0,
       estimator=Pipeline(steps=[('minmaxscaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('selectkbest', SelectKBest(k=10, score_func=<function f_classif at 0x10530bed8>)), ('decisiontre
eclassifier', DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            random_state=None, splitter='best'))]),
       fit_params={}, iid=True, loss_func=None, n_jobs=1,
       param_grid={'selectkbest__score_func': [<function f_classif at 0x10530bed8>], 'selectkbest__k': [1, 2, 3, 4, 5, 6, 7, 8, 0, 10, 11, 12, 13, 14]},
       pre_dispatch='2*n_jobs', refit=True, score_func=None, scoring='f1',
       verbose=1)
        Accuracy: 0.79993       Precision: 0.24503      Recall: 0.24050 F1: 0.24275     F2: 0.24139
        Total predictions: 15000        True positives:  481    False positives: 1482   False negatives: 1519   True negatives: 11518
```

e. GuassianNB - **Precision 0.35, Recall 0.3**



```
GridSearchCV(cv=StratifiedShuffleSplit(labels=[ 0.  0. ...,  0.  0.], n_iter=20, test_size=0.5, random_state=0),
       error_score=0,
       estimator=Pipeline(steps=[('minmaxscaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('selectkbest', SelectKBest(k=10, score_func=<function f_classif at 0x1054cced8>)), ('gaussiannb'
, GaussianNB())]),
       fit_params={}, iid=True, loss_func=None, n_jobs=1,
       param_grid={'selectkbest__score_func': [<function f_classif at 0x1054cced8>], 'selectkbest__k': [1, 2, 3, 4, 5, 6, 7, 8, 0, 10, 11, 12, 13, 14]},
       pre_dispatch='2*n_jobs', refit=True, score_func=None, scoring='f1',
       verbose=1)
        Accuracy: 0.83107       Precision: 0.34655      Recall: 0.30150 F1: 0.32246     F2: 0.30955
        Total predictions: 15000        True positives:  603    False positives: 1137   False negatives: 1397   True negatives: 11863
```

4. What does it mean to tune the parameters of an algorithm, and what can happen if you don't do this well? How did you tune the parameters of your particular algorithm? (Some algorithms do not have parameters that you need to tune -- if this is the case for the one you picked, identify and briefly explain how you would have done it for the model that was not your final choice or a different model that does utilize parameter tuning, e.g. a decision tree classifier). [relevant rubric item: "tune the algorithm"]

The goal of tuning the parameters of an algorithm is to optimize the algorithm so that it can perform the "best". For example, if we "fine tune" too much ("overfit"), the algorithm would need to go thru too much details and performance would be a disaster. On the other hand, if we "tune' the algorithm too "general", the precision/accuracy of the prediction would not be ideal.

Classifier's performance with different parameter values:

	Precision	**Recall**
n_jobs = 1	0.35	0.3
n_jobs = -1	Getting error on cross_validation.py: 1470: FitFailedWarning: Calssifier fit failed. ValueError('Found array with 0 features while minimum of 1 is required"	
Params – selectkbest__k = 14	0.35	0.3
Params – selectkbest__k = 10	0.37	0.298
Params – selectkbest__k = 12	**0.36**	**0.3**
Selectkbest__score_func = f_classif	0.36	0.3
Selectkbest__score_func = chi2	0.27	0.29
Refit = false	Per sckit-learn doc, if refit is set to False, it is impossible to make predictions using this GridSearchCV insance after fitting. By default it is set to true	
Pre-dispatch = 2*n_jobs	0.35	0.3

n_jobs -> GridSearchCV evaluates each parameter setting independently. By setting n_jobs = -1, then we can force computations to be run in parallel.

For parameter "**scoring**", "f1" was chosen since it is for binary targets.

For "**cv**", I used StratifiedShuffleSplit

"f_classif" is ANOVA F-value between label/feature for classification tasks.

"chi2" is chi-squared stats of non-negative features for classification tasks. Please see results on table above

for cross-validaiton (cv) parameters:

	Precision	**Recall**
Test_size = 0.5	0.36	0.3

Test_size = 0.9	0.36	0.26
Test_size = 0.2	0.35	0.3
Test_size = 0.6	**0.365**	**0.312**
Test_size = 0.7	0.363	0.293

5. What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis? [relevant rubric item: "validation strategy"]

Validation is to measure how effective the algorithm is. One classic mistake could be not having "random" data. In order to have good representation data during training phase and "randomness", "**Cross validation**" was used.

Why is validation important and necessary?

We need to perform validation to make sure "**overfitting**" does not occur. "Overfiting" means the algorithm is too "fine-tuned" with the training set data, and it doesn't make good prediction on the real data.

How is validation performed?

One common mistake is to reuse the same data for prediction which was used for training. Therefore, we need to hold out a "test" set from the rest of the data set, which is used for prediction. However, due to the limited size of our data set, we need to use cross validation to achieve randomness in our data sets. What cross-validation (cv) does is to split the data in k smaller sets. Each time the data set is split, a model is trained using k-1 of the folds as training data. Then the resulting model is validated on the remaining part of the data.

What does "effective" mean in this context?

"Effective" in this context means to have separate sets of train and test data and also to have randomness, and good representative of the data.

The StratifiedShuffleSplit was used to evaluate the performance of the algorithms. StratifiedShuffleSplit is to perform k random splits, train and test splits each have class distributions that reflect the overall data. The reason we choose StratifiedShuffleSplit is the **limited size of our data set**, while keeping the class distributions when the data is split. If we use another method such as .KFold, then we will not be able to maintain the class distributions.

	Precision	**Recall**
StratifiedShuffleSplit n_iter = 20	**0.365**	**0.312**
k-fold - per scikit learn documention, StratifiedKFold should be used when y is binary.		

In our case of determining for poi, therefore, StratifiedKFold should be used		
Stratified k-fold		
fold = 3	0.34	0.289
fold = 5	0.34	0.287
fold = 10	0.347	0.293
fold = 20	0.329	0.305

6. Give at least 2 evaluation metrics and your average performance for each of them. Explain an interpretation of your metrics that says something human-understandable about your algorithm's performance. [relevant rubric item: "usage of evaluation metrics"]

A common evaluation metric of the algorithm performance would be to measure the accuracy. In our context, accuracy would be the number of correctly identified POIs and non-POIs divided by total number of observations predicted. However, accuracy would not be a good measure of our algorithm in this case because our data set is very "unbalanced", having only 18 POIs, and 126 non-POIs. Since our data set is very unbalanced, if we guess everyone to be non-POI, accuracy would still be 87%.

Therefore, we use other metrics to evaluate our algorithm performance, precision and recall.

In "human understandable" terms,

<u>Precision</u> **is the ratio of how often the algorithm model correctly identifies a positive match to total times it guesses a positive label**

In our context, this would mean how often the people the algorithm identifies as POI are actual POI

<u>Recall</u> is **the ratio of how often the algorithm model correctly identifies a label as positive to how many total positives labels**

In our context, this would mean how many POI the algorithm identifies out of the total number of real POI

<u>In math equations form:</u>

Precision = true positive / (true positive + false positive)

Recall = true positive / (true positive + false negative)