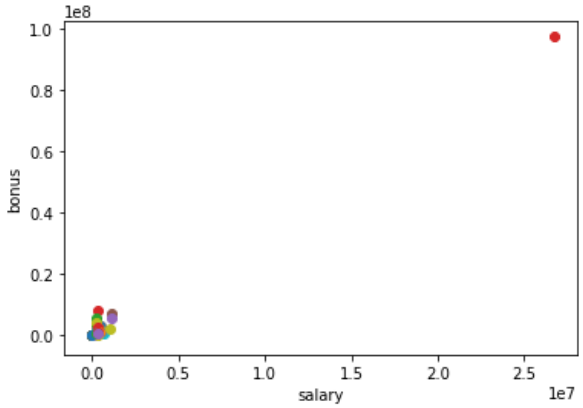
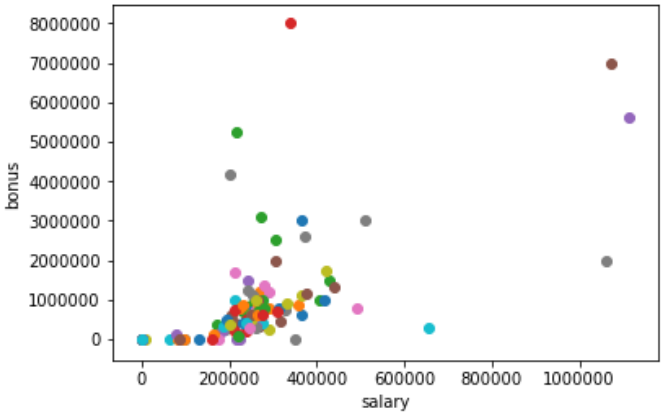
Question 1

The purpose of this machine learning project was to use different machine learning algorithms to create predictive models that would correctly identify the persons of interest. Our dataset had an imbalance between persons of interest and the total sample of employees. Our overall sample size was initially 146 but there was an outlier that did not correspond to an actual employee so it was removed below. The actual sample size was 145 employees with only 18 classified as persons of interest and 127 employees that were not classified as a POI. There were 21 features that came along within this dataset. Within the project, I ended up creating an additional feature that summed up the salary and bonus of each employee. The only feature that did not have any missing values corresponded to POI. The remainder features had plenty of NaN's scattered accross the different employees.

The financial dataset had one initial outlier that was shown within the scatter plot. This data point was shown past the 25 million mark. Further exploration revealed that it corresponded to the key named “Total” that had a corresponding value of $26,704,229. Total represented the summation of all the employees’ salaries within the dataset. This Total key was then removed by using the pop() method to remove this key-value pair from the dictionary. Another plot was created and now three outliers emerged from the dataset. These outliers corresponded to individuals that were classified as persons of interest and would not be convenient to dispose of these points. The first chart demonstrates the initial outlier that corresponded to “Total”. The second chart shows once the pop method was used to remove “Total”.





Question 2

The methodology of deciding which features to use was accomplished by SelectKBest for its univariate feature selection capabilities. For SelectKBest, I initially used all of the features except the email addresses. Once the scores were given for each feature, I went through the list and removed features that got below a score of 10 for their initial output. The final features that were used for my list of seven features for the POI identifier were salary, bonus, salary plus bonus, deferred income, exercised stock options, long term incentive, and total stock value. Feature scaling was not used as I did not incorporate SVM's nor K-means as my classifiers. The feature that I created took the total sum between salary and bonus for each employee. The rationale was to have a key that took the total yearly payment that did not take into consideration of any unearned income or investments. For this newly created feature, the selectKBest score was highly favorable to include it in the final feature list. The feature that I created, salary plus bonus, received an initial score of 22.888. The remaining scores are listed below and only the scores that were 10 or above were used while choosing seven features. This was used as an initial point of reference to observe how many classifiers would achieve to have both the precision and recall above .3.

```
9 new = SelectKBest(k=2).fit(features, labels)
10 print new.scores_
```

```
18.57570327  21.06000171  22.8879507    0.21705893  11.59554766
 2.10765594  25.09754153   6.23420114   0.1641645    5.34494152
 2.42650813   7.2427304   10.07245453   4.20497086   9.34670079
 0.06498431   8.74648553   1.69882435   8.86672154  24.46765405]
```

SelectKBest for Seven Features:

Features	Score
Salary	17.15
Bonus	19.72
Salary Plus Bonus	21.40
Deferred Income	10.91
Exercise Stock Options	23.69
Long Term Incentive	9.32
Total Stock Value	23.05

After choosing seven features, I tried using six features to see how each of the classifiers used would perform. To get to six features, I chose to remove the lowest performing score from the above seven features which belonged to Long Term Incentive. I reran the SelectKBest for the following six features and the results can be seen below. After having these six features, I also reran the classifiers to observe what type of performance they had.

SelectKBest for Six Features:

Features	Score
Salary	17.15
Bonus	19.72
Salary Plus Bonus	21.40
Deferred Income	10.91
Exercise Stock Options	23.69
Total Stock Value	23.05

```
new = SelectKBest(k=2).fit(features, labels)
print new.scores_
```

```
[ 17.14638411  19.72272095  21.40137978  10.9110254   23.68630462
  23.04507278]
```

To get to five features, I chose to remove the lowest performing score from the above six features which belonged to Deferred Income. I reran the SelectKBest for the following five features and the results can be seen below. After having these six features, I also reran the classifiers to observe what type of performance they had.

Features	Score
Salary	14.58
Bonus	17.33
Salary Plus Bonus	18.74
Exercise Stock Options	21.15
Total Stock Value	20.49

```
new = SelectKBest(k=2).fit(features, labels)
print new.scores_
```

```
[ 14.57930747  17.32607465  18.73555197  21.15364654  20.49288835]
```

Question 3

While working on the project, I experimented with 7, 6, and 5 features with Gaussian Naïve Bayes, Decision Tree, and KNeighbors classifiers. As shown below, the recall decreases for Gaussian Naïve Bayes as less features are used for the features that I actually chose. As less features that I chose were used, this actually increases recall for KNeighbors. Recall for the Decision Tree initially decreases when given six features but ends up increasing as it gets to five features. For precision, the KNeighbors classifier increased its precision as less of the features that I chose were included.The average performance of the classifiers attempted increased for both recall and precision as less of the features that I chose were used. The highest recall achieved belonged to the Decision Tree at .40 while using salary, bonus, salary plus bonus,exercised stock options, and total stock value features to identify all the poi. Its corresponding precision was of .38 correct at identifying the poi

Precision and Recall for Seven Features

Classifier	Precision	Recall
Gaussian Naïve Bayes	.44	.37
Decision Tree	.32	.33
KNeighbors	.67	.22

Average Recall: .3066
Average Precision: .4767

Precision and Recall for Six Features

Classifier	Precision	Recall
Gaussian Naïve Bayes	.46	.36
Decision Tree	.31	.32
KNeighbors	.79	.29

Average Recall: .3233
Average Precision: .52

Precision and Recall for Five Features

Classifier	Precision	Recall
Gaussian Naïve Bayes	.44	.299
Decision Tree	.38	.40
KNeighbors	.81	.31

Average Recall: .3363
Average Precision: .5433

Question 4

GridSearchCV was used for parameter tuning. Parameter tuning with GridSearchCV took a list of several parameters for a particular algorithm and cross validated with a test dataset to come up with a combination that produces the optimum solution. This will also help reduce the possibility of overfitting. StratifiedShuffleSplit was used within GridSearchCV because of the imbalance that existed in our data. The GridSearchCV was also tuned to focus on the recall of the algorithm.

The first tune consisted of only listing the various possible algorithms such as auto, ball tree, kd tree and brute to the GridSearchCV. The outcome did not change from the original value. It ended up choosing auto as the algorithm. On the second tune, I only gave various numerical options to the n_neighbors parameter and the output dropped down to .3100 for the recall. On the third tune, I was able to increase it to .31700 by running the weights parameter with either the uniform or distance option. The third tune ended up choosing the uniform option to achieve this result.

Tuning with Grid Search with Five Features

Classifier	Precision	Recall
KNeighbors Original	.80513	.31400
KNeighbors 1st Tune	.80513	.31400
KNeighbors 2nd Tune	.71101	.31000
KNeighbors 3rd Tune	.78758	.31700

First Tune:

```
#KNeighbors GridSearch  Tune 1 with Five Features
from sklearn.model_selection import GridSearchCV
from sklearn import cross_validation
parameters = {'algorithm': ('auto','ball_tree','kd_tree','brute')}

KN = KNeighborsClassifier()
clf = GridSearchCV(KN, parameters)
cv = cross_validation.StratifiedShuffleSplit(labels, 100, random_state = 42)
a_grid_search = GridSearchCV(clf, parameters, cv = cv, scoring = 'recall')

test_classifier(clf, my_dataset, features_list)
```

```
GridSearchCV(cv=None, error_score='raise',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'algorithm': ('auto', 'ball_tree', 'kd_tree', 'brute')},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
        Accuracy: 0.88277       Precision: 0.80513      Recall: 0.31400 F1: 0.45180
```

Second Tune:

```
#KNeighbors GridSearch Tune 2 with Five Features
from sklearn.model_selection import GridSearchCV
from sklearn import cross_validation
parameters = {'n_neighbors':[3,4,5,6,8,10,12,13,14]}

KN = KNeighborsClassifier()
clf = GridSearchCV(KN, parameters)
cv = cross_validation.StratifiedShuffleSplit(labels, 100, random_state = 42)
a_grid_search = GridSearchCV(clf, parameters, cv = cv, scoring = 'recall')

test_classifier(clf, my_dataset, features_list)
```

```
GridSearchCV(cv=None, error_score='raise',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'n_neighbors': [3, 4, 5, 6, 8, 10, 12, 13, 14]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
        Accuracy: 0.87446       Precision: 0.71101      Recall: 0.31000 F1: 0.43175
        Total predictions: 13000        True positives:  620    False positives:  252
negatives: 10748
```

Third Tune:

```
#KNeighbors GridSearch Tune 3 with Five Features
from sklearn.model_selection import GridSearchCV
from sklearn import cross_validation
parameters = {'weights': ('uniform', 'distance')}

KN = KNeighborsClassifier()
clf = GridSearchCV(KN, parameters)
cv = cross_validation.StratifiedShuffleSplit(labels, 100, random_state = 42)
a_grid_search = GridSearchCV(clf, parameters, cv = cv, scoring = 'recall')

test_classifier(clf, my_dataset, features_list)
```

```
GridSearchCV(cv=None, error_score='raise',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'weights': ('uniform', 'distance')},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
        Accuracy: 0.88177       Precision: 0.78758      Recall: 0.31700 F1: 0.45205     F
        Total predictions: 13000        True positives:  634    False positives:  171   F
negatives: 10829
```

Question 5

Cross validation and StratifiedShufflesplit were used within GridsearchCV for validation purposes. Cross validation analyzes the results from statistical analysis and generalizes it to another independent dataset that model has not seen before. This ensures that the algorithm I did cross validation on does what it is supposed to do. StratifiedShuffleSplit was used because of the imbalance between poi and overall sample. This method ensures that each subgroup gets the same percentage of samples. GridSearchCV then uses cross validation, StratifiedShuffleSplit, and parameters that I specified to come up with the optimal recall value.

Question 6

Precision is the fraction of true positives among the total amount of all of the true positives and false positives. Precision tells us the proportion of employees that were labeled as persons of interest that actually were.Recall is the fraction of true positives among the total amount of false negatives and true positives. Recall tells us the proportion of employees that actually were persons of interest by us as being persons of interest.

In our project, recall was much more important due to the imbalance that existed. The performance for precision and recall for each classifier that was attempted is shown below. While working on the project, I experimented with 7, 6, and 5 features with Gaussian Naïve Bayes, Decision Tree, and KNeighbors classifiers. As shown below, the recall decreases for Gaussian Naïve Bayes as less features are used for the features that I actually chose. As less features that I chose were used, this actually increases recall for KNeighbors. Recall for the Decision Tree initially decreases when given six features but ends up increasing as it gets to five features. For precision, the KNeighbors classifier increased its precision as less of the features that I chose were included.The average performance of the classifiers attempted increased for both recall and precision as less of the features that I chose were used. The highest recall achieved belonged to the Decision Tree at .40 while using salary, bonus, salary

plus bonus,exercised stock options, and total stock value features to identify all the poi. Its corresponding precision was .38.

Precision and Recall for Seven Features

Classifier	Precision	Recall
Gaussian Naïve Bayes	.44	.37
Decision Tree	.32	.33
KNeighbors	.67	.22

Average Recall: .3066
Average Precision: .4767

Precision and Recall for Six Features

Classifier	Precision	Recall
Gaussian Naïve Bayes	.46	.36
Decision Tree	.31	.32
KNeighbors	.79	.29

Average Recall: .3233
Average Precision: .52

Precision and Recall for Five Features

Classifier	Precision	Recall
Gaussian Naïve Bayes	.44	.299
Decision Tree	.38	.40
KNeighbors	.81	.31

Average Recall: .3363
Average Precision: .5433