Bryony Miles - Enron Project Questions Answered 2 (more details in the report**)**

Question:	Which Enron Employees may have committed fraud based on the public Enron financial and email dataset.
Data:	146 entries, 21 features (financial, email related, **poi**) **poi** = individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity

Machine learning is particularly useful in this case as the entries fall into two categories: guilty (poi) or not guilty (non-poi).

Outliers - during the data exploration I found the following:

- various NaN values which I analysed and converted to 0
- two invalid entries - TOTAL and THE TRAVEL AGENCY IN THE PARK which I deleted
- two data entry issues - BHANTNAGER SANJAY and BELFER ROBERT which I updated using the original data
- I also looked at two POIs with a lot of nulls and 20 POIs with very high entries but decided that the data was relevant and kept them in.

New Features - In the end I creating 9 new features:

- *key_payments:* salary + bonus + other
- *deferral_balance:* deferral_payments + deferred_ income
- r*etention incentives:* long_term_incentive + total_stock_value
- *total_of_totals:* total_payments + total_stock_value
- *bonus/salary*
- *exercised_stock_options/salary*
- *retention_incentives/key_payments*
- *odd_payments - retention_incentives > 28,000,000* and *deferral_balance < -1,500,000*
- poi_emailratio - ratio of emails to or from POIs

Feature Selection - I used KBest** scores to decide which value of k to begin my initial algorithms with:

```
bonus/salary KBEST 10.7835847082
exercised_stock_options/salary KBEST 0.119303870047
poi_emailratio KBEST 5.39937028809
exercised_stock_options/salary KBEST 0.119303870047
odd_payments KBEST 47.4045185584
key_payments KBEST 17.4322739258
deferral_balance KBEST 13.5354825655
retention_incentives KBEST 23.8497535003
total_of_totals KBEST 16.9893364218
salary KBEST 18.2896840434
bonus KBEST 20.7922520472
other KBEST 4.20243630027
deferral_payments KBEST 0.228859619021
deferred_income KBEST 11.4248914854
loan_advances KBEST 7.18405565829
long_term_incentive KBEST 9.92218601319
exercised_stock_options KBEST 22.3489754073
restricted_stock_deferred KBEST 0.768146344787
restricted_stock KBEST 8.82544221992
```

The top 7 features look pretty good.

I played around with GridSearchCV and Kbest options for my chosen algorithms but in the end I concluded that this overkill. KBest scores look good for the first seven which is reasoned enough.

***First time round I went down the wrong path and decided to choose the features intuitively as KBest seemed to be throwing out inconsistent results. I was clearly not reading the results properly… All part of the learning curve!*

Scaling - As the features were all integers, potentially related and with a lot of outliers of interest with the numbers varying between features I decided to use a MinMaxScaler.

Evaluation Metrics - I decided to use *precision* and *recall:*

- *Precision* = likelihood that an identified POI is actually a POI. % false alarms.
- *Recall* = likelihood of identifying a POI in the dataset if there was one.
- As the rubric asks for above 0.3 in both I'm assuming they are of equal importance.

Cross Validation - I needed to split the data for cross validation - i.e. use a training set and a test set on the same algorithm and compare at the results. Initially I used train_test_split but then decided to take advantage of the validator in *tester.py* which uses Stratified Shuffle Split to create the train and test data and has a very detailed report.

Algorithm Choice: I tested out five algorithms using the MinMaxScaler, KBest k=7. As advised after submission 1 I used a pipeline. The GaussianNB result was pretty good: *Precision* 0.48193, *Recall* 0.28 which is an indication I'm on the right track.

Parameter Tuning: I started out using GridSearchCV to tune the parameters but the search time was excruciatingly slow and I decided it would be a better learning experience if I iterated through the parameters myself and got a better idea of their properties. Using the Sklearn website, I worked systematically on the remaining four algorithms.

Decision Tree			**Logistic Regression**			**LinearSVC**			**Random Forest**		
	p	**r**		**p**	**r**		**p**	**r**		**p**	**r**
initial result	0.40367	0.39600		0.80662	0.15850		0.74294	0.25000		0.51138	0.23600
criterion = entropy	0.39067	0.35600	c = 1000	0.51330	0.26050	c=1000	0.29516	0.29900	n_estimators	improves balance not speed.	
splitter = random	0.41445	0.4820	As C increases, precision decreases and recall increases			loss='hinge'	1	0.27550	criterion = 'entropy'	0.52625	0.22050
max features	best at 7 (as you'd expect)		class weight = balanced	**0.33021**	**0.45850**	fit intercept = false	0.31234	0.24300	bootstrap ='false'	0.51796	0.32450
min_samples_split = 5	0.46245	0.39100	parameters with no impact	fit intercept = false, intercept scaling verbose, njobs		class weight = balanced	0.30977	0.48850	oob score n_estimators = 100	0.59936	0.27900
class weight = balanced	0.31463	0.27750				parameters with no impact	verbose, max_iter,dual		class weight = balanced	0.49195	0.16800
random state = '42'	0.39731	0.38400				I then looked at the various combinations and optimum result was.			As you can't have out of bag estimation if bootstrap=False and that had the best result I went for		
min_samples_split = 5 *and* splitter = random	**0.50879**	**0.41950**				C=10, class *and* weight = balanced	**0.31288**	**0.56600**	n_estimators = 100 *and* bootstrap = False	**0.51608**	**0.33700**

Finally I tested my best algorithm (Decision Tree, min_sample_splits = 5, splitter = random) with different KBest values and it turned out that k=6 was marginally better.   Here are the final full results:

```
Pipeline(steps=[('Scaler', MinMaxScaler(copy=True, feature_range=(0, 1))), ('SKB', SelectKBest(k=6,
score_func=<function f_classif at 0x107b47e18>)), ('Decision Tree',
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=5, min_weight_fraction_leaf=0.0,
            presort=False, random_state=42, splitter='random'))])
      Accuracy: 0.88127 Precision: 0.56616     Recall: 0.46850   F1: 0.51272       F2: 0.48524
      Total predictions: 15000     True positives:  937    False positives:  718    False negatives:
1063  True negatives: 12282
```

Note that I've changed random state to 42 to maintain consistent results.

Conclusion

So the final results means that 57% of POIs identified are definitely POIs and 47% of POIs in the dataset are being identified.

I'm sure there are other things I could do to improve this score. I didn't perform PCA and I did not use a Parameter Tuning algorithm. However, the major drawback is the restriction of a very small dataset (144 after outlier removal) with a small % of POIs - 12.5%. Ideally more data would be released for the 18 POIs identified who had no financial records in the dataset. This however is unlikely after so many years.

The logical next step therefore would be to look at the actual emails and tracking links between POIs and other email addresses and identifying key words which are more common when POIs are involved.

This has been a fabulous project. It has been a learning curve twice over and I would like to come back to the data again and have a dig into the email content linked to POIs.

The code is found in four python files:
- *explore_data.py*
- *feature_selection.py (commented out in this case)*
- *algorithms.py*
- *poi_id.py*