Identify Fraud from Enron Email

James Dumville October 2020

Project Overview:

In 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for top executives. In this project, you will play detective, and put your new skills to use by building a person of interest identifier based on financial and email data made public as a result of the Enron scandal. To assist you in your detective work, we've combined this data with a hand-generated list of persons of interest in the fraud case, which means individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity.

Project Goal:

The goal of this project is to evaluate and select a machine learning algorithm that best identifies persons of interest in the historic fraud that occured at Enron. In 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for top executives. In this data, I will look for features that can be used in my algorithm to identify the persons of interest.

Data Exploration:

In [1]:
```
"""Import pickle and sklearn to get started.
Load the data as enron_dict"""

import pickle
import sklearn

enron_data = pickle.load(open("final_project_dataset_unix.pkl", "rb"))
```

The number of people in the dataset seems like a good place to start.

In [2]:
```
len(enron_data)
```

Out[2]:
```
146
```

Now let's take a look at the names we have.

In [3]:
```
sorted(enron_data.keys())
```

```
Out[3]: ['ALLEN PHILLIP K',
 'BADUM JAMES P',
 'BANNANTINE JAMES M',
 'BAXTER JOHN C',
 'BAY FRANKLIN R',
 'BAZELIDES PHILIP J',
 'BECK SALLY W',
 'BELDEN TIMOTHY N',
 'BELFER ROBERT',
 'BERBERIAN DAVID',
 'BERGSIEKER RICHARD P',
 'BHATNAGAR SANJAY',
 'BIBI PHILIPPE A',
 'BLACHMAN JEREMY M',
 'BLAKE JR. NORMAN P',
 'BOWEN JR RAYMOND M',
 'BROWN MICHAEL',
 'BUCHANAN HAROLD G',
 'BUTTS ROBERT H',
 'BUY RICHARD B',
 'CALGER CHRISTOPHER F',
 'CARTER REBECCA C',
 'CAUSEY RICHARD A',
 'CHAN RONNIE',
 'CHRISTODOULOU DIOMEDES',
 'CLINE KENNETH W',
 'COLWELL WESLEY',
 'CORDES WILLIAM R',
 'COX DAVID',
 'CUMBERLAND MICHAEL S',
 'DEFFNER JOSEPH M',
 'DELAINEY DAVID W',
 'DERRICK JR. JAMES V',
 'DETMERING TIMOTHY J',
 'DIETRICH JANET R',
 'DIMICHELE RICHARD G',
 'DODSON KEITH',
 'DONAHUE JR JEFFREY M',
 'DUNCAN JOHN H',
 'DURAN WILLIAM D',
 'ECHOLS JOHN B',
 'ELLIOTT STEVEN',
 'FALLON JAMES B',
 'FASTOW ANDREW S',
 'FITZGERALD JAY L',
 'FOWLER PEGGY',
 'FOY JOE',
 'FREVERT MARK A',
 'FUGH JOHN L',
 'GAHN ROBERT S',
 'GARLAND C KEVIN',
 'GATHMANN WILLIAM D',
 'GIBBS DANA R',
 'GILLIS JOHN',
 'GLISAN JR BEN F',
 'GOLD JOSEPH',
 'GRAMM WENDY L',
```

```
'GRAY RODNEY',
'HAEDICKE MARK E',
'HANNON KEVIN P',
'HAUG DAVID L',
'HAYES ROBERT E',
'HAYSLETT RODERICK J',
'HERMANN ROBERT J',
'HICKERSON GARY J',
'HIRKO JOSEPH',
'HORTON STANLEY C',
'HUGHES JAMES A',
'HUMPHREY GENE E',
'IZZO LAWRENCE L',
'JACKSON CHARLENE R',
'JAEDICKE ROBERT',
'KAMINSKI WINCENTY J',
'KEAN STEVEN J',
'KISHKILL JOSEPH G',
'KITCHEN LOUISE',
'KOENIG MARK E',
'KOPPER MICHAEL J',
'LAVORATO JOHN J',
'LAY KENNETH L',
'LEFF DANIEL P',
'LEMAISTRE CHARLES',
'LEWIS RICHARD',
'LINDHOLM TOD A',
'LOCKHART EUGENE E',
'LOWRY CHARLES P',
'MARTIN AMANDA K',
'MCCARTY DANNY J',
'MCCLELLAN GEORGE',
'MCCONNELL MICHAEL S',
'MCDONALD REBECCA',
'MCMAHON JEFFREY',
'MENDELSOHN JOHN',
'METTS MARK',
'MEYER JEROME J',
'MEYER ROCKFORD G',
'MORAN MICHAEL P',
'MORDAUNT KRISTINA M',
'MULLER MARK S',
'MURRAY JULIA H',
'NOLES JAMES L',
'OLSON CINDY K',
'OVERDYKE JR JERE C',
'PAI LOU L',
'PEREIRA PAULO V. FERRAZ',
'PICKERING MARK R',
'PIPER GREGORY F',
'PIRO JIM',
'POWERS WILLIAM',
'PRENTICE JAMES',
'REDMOND BRIAN L',
'REYNOLDS LAWRENCE',
'RICE KENNETH D',
'RIEKER PAULA H',
```

```
 'SAVAGE FRANK',
 'SCRIMSHAW MATTHEW',
 'SHANKMAN JEFFREY A',
 'SHAPIRO RICHARD S',
 'SHARP VICTORIA T',
 'SHELBY REX',
 'SHERRICK JEFFREY B',
 'SHERRIFF JOHN R',
 'SKILLING JEFFREY K',
 'STABLER FRANK',
 'SULLIVAN-SHAKLOVITZ COLLEEN',
 'SUNDE MARTIN',
 'TAYLOR MITCHELL S',
 'THE TRAVEL AGENCY IN THE PARK',
 'THORN TERENCE H',
 'TILNEY ELIZABETH A',
 'TOTAL',
 'UMANOFF ADAM S',
 'URQUHART JOHN A',
 'WAKEHAM JOHN',
 'WALLS JR ROBERT H',
 'WALTERS GARETH W',
 'WASAFF GEORGE',
 'WESTFAHL RICHARD K',
 'WHALEY DAVID A',
 'WHALLEY LAWRENCE G',
 'WHITE JR THOMAS E',
 'WINOKUR JR. HERBERT S',
 'WODRASKA JOHN',
 'WROBEL BRUCE',
 'YEAGER F SCOTT',
 'YEAP SOON']
```

There's a name I remember from the news.

In [4]:
```
field = list(enron_data['LAY KENNETH L'])

print(enron_data['LAY KENNETH L'])
```


```
{'salary': 1072321, 'to_messages': 4273, 'deferral_payments': 202911, 'total_
payments': 103559793, 'loan_advances': 81525000, 'bonus': 7000000, 'email_add
ress': 'kenneth.lay@enron.com', 'restricted_stock_deferred': 'NaN', 'deferred
_income': -300000, 'total_stock_value': 49110078, 'expenses': 99832, 'from_po
i_to_this_person': 123, 'exercised_stock_options': 34348384, 'from_messages':
36, 'other': 10359729, 'from_this_person_to_poi': 16, 'poi': True, 'long_term
_incentive': 3600000, 'shared_receipt_with_poi': 2411, 'restricted_stock': 14
761694, 'director_fees': 'NaN'}
```


Now I want to write the data to a CSV file to be read into pandas for further analysis.

In [5]:
```
import csv
fieldnames = ['name'] + field

# with open('enron.csv', 'w') as csvfile:
#     writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

#     writer.writeheader()
#     for name in enron_data.keys():
#         if name != 'TOTAL':
#             n = {'name':name}
#             n.update(enron_data[name])
#             writer.writerow(n)
```

Now read that file back in.

In [6]:
```
import pandas as pd
enron = pd.read_csv('enron.csv')
```

Now we can look a little more at the data, like how many null values we have per column.

In [7]:
```
enron.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145 entries, 0 to 144
Data columns (total 22 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   name                       145 non-null    object 
 1   salary                     94 non-null     float64
 2   to_messages                86 non-null     float64
 3   deferral_payments          38 non-null     float64
 4   total_payments             124 non-null    float64
 5   loan_advances              3 non-null      float64
 6   bonus                      81 non-null     float64
 7   email_address              111 non-null    object 
 8   restricted_stock_deferred  17 non-null     float64
 9   deferred_income            48 non-null     float64
 10  total_stock_value          125 non-null    float64
 11  expenses                   94 non-null     float64
 12  from_poi_to_this_person    86 non-null     float64
 13  exercised_stock_options    101 non-null    float64
 14  from_messages              86 non-null     float64
 15  other                      92 non-null     float64
 16  from_this_person_to_poi    86 non-null     float64
 17  poi                        145 non-null    bool   
 18  long_term_incentive        65 non-null     float64
 19  shared_receipt_with_poi    86 non-null     float64
 20  restricted_stock           109 non-null    float64
 21  director_fees              16 non-null     float64
dtypes: bool(1), float64(19), object(2)
memory usage: 24.1+ KB
```

Here we can see where the nulls in the dataset reside, we only have all data points for name and the POI indicator. This tells me that this task may be a little more difficult than I initially thought.

In [8]:
```
enron['poi'].value_counts()
```

Out[8]:
```
False    127
True      18
Name: poi, dtype: int64
```

Here we see we have 18 of our POIs, and 127 non-POIs.
Let's take a look at the POIs and see if there are any trends that may indicate they could be good features for our model.

In [9]:
```
poi_data = enron[enron.poi.isin([True])]
```

In [10]:
```
poi_data.describe()
```

Out[10]:

	salary	to_messages	deferral_payments	total_payments	loan_advances	bo
count	1.700000e+01	14.000000	5.000000e+00	1.800000e+01	1.0	1.600000e
mean	3.834449e+05	2417.142857	5.198942e+05	7.913590e+06	81525000.0	2.075000e
std	2.783597e+05	1961.858101	9.128895e+05	2.396549e+07	NaN	2.047437e
min	1.584030e+05	225.000000	1.025900e+04	9.109300e+04	81525000.0	2.000000e
25%	2.401890e+05	1115.750000	2.761000e+04	1.142396e+06	81525000.0	7.750000e
50%	2.786010e+05	1875.000000	2.029110e+05	1.754028e+06	81525000.0	1.275000e
75%	4.151890e+05	2969.250000	2.146780e+05	2.665345e+06	81525000.0	2.062500e
max	1.111258e+06	7991.000000	2.144013e+06	1.035598e+08	81525000.0	7.000000e

Let's seperate the financial information from the email info.

In [11]:
```
poi_finances = poi_data[['salary','deferral_payments','total_payments','loan_a
dvances','bonus','restricted_stock_deferred','deferred_income','total_stock_va
lue','expenses','exercised_stock_options','other','long_term_incentive','restr
icted_stock','director_fees']]
poi_finances.describe()
```

Out[11]:

	salary	deferral_payments	total_payments	loan_advances	bonus	restricted
count	1.700000e+01	5.000000e+00	1.800000e+01	1.0	1.600000e+01	
mean	3.834449e+05	5.198942e+05	7.913590e+06	81525000.0	2.075000e+06	
std	2.783597e+05	9.128895e+05	2.396549e+07	NaN	2.047437e+06	
min	1.584030e+05	1.025900e+04	9.109300e+04	81525000.0	2.000000e+05	
25%	2.401890e+05	2.761000e+04	1.142396e+06	81525000.0	7.750000e+05	
50%	2.786010e+05	2.029110e+05	1.754028e+06	81525000.0	1.275000e+06	
75%	4.151890e+05	2.146780e+05	2.665345e+06	81525000.0	2.062500e+06	
max	1.111258e+06	2.144013e+06	1.035598e+08	81525000.0	7.000000e+06	

In [12]:
```
poi_email = poi_data[['to_messages','from_poi_to_this_person','from_messages',
'from_this_person_to_poi','shared_receipt_with_poi']]
poi_email.describe()
```

Out[12]:

	to_messages	from_poi_to_this_person	from_messages	from_this_person_to_poi	shared_
count	14.000000	14.000000	14.000000	14.000000	
mean	2417.142857	97.785714	300.357143	66.714286	
std	1961.858101	76.058862	805.844574	158.289622	
min	225.000000	13.000000	16.000000	4.000000	
25%	1115.750000	44.500000	33.000000	12.500000	
50%	1875.000000	62.000000	44.500000	15.500000	
75%	2969.250000	135.750000	101.500000	28.750000	
max	7991.000000	240.000000	3069.000000	609.000000	

Now we can do the same for the non-POIs so we can compare this data.

In [13]:
```
nonpoi_data = enron[enron.poi.isin([False])]
```

In [14]:
```
nonpoi_finances = nonpoi_data[['salary','deferral_payments','total_payments',
'loan_advances','bonus','restricted_stock_deferred','deferred_income','total_s
tock_value','expenses','exercised_stock_options','other','long_term_incentive'
,'restricted_stock','director_fees']]
nonpoi_finances.describe()
```

Out[14]:

	salary	deferral_payments	total_payments	loan_advances	bonus	restricted
count	7.700000e+01	3.300000e+01	1.060000e+02	2.000000e+00	6.500000e+01	
mean	2.621515e+05	8.903462e+05	1.725091e+06	1.200000e+06	9.868249e+05	
std	1.392317e+05	1.341381e+06	2.618288e+06	1.131371e+06	1.173880e+06	
min	4.770000e+02	-1.025000e+05	1.480000e+02	4.000000e+05	7.000000e+04	
25%	2.061210e+05	8.543000e+04	3.304798e+05	8.000000e+05	4.000000e+05	
50%	2.516540e+05	2.604550e+05	1.056092e+06	1.200000e+06	7.000000e+05	
75%	2.885890e+05	8.753070e+05	2.006025e+06	1.600000e+06	1.000000e+06	
max	1.060932e+06	6.426990e+06	1.725253e+07	2.000000e+06	8.000000e+06	

In [15]:
```
nonpoi_email = nonpoi_data[['to_messages','from_poi_to_this_person','from_mess
ages','from_this_person_to_poi','shared_receipt_with_poi']]
nonpoi_email.describe()
```

Out[15]:

	to_messages	from_poi_to_this_person	from_messages	from_this_person_to_poi	shared_
count	72.000000	72.000000	72.000000	72.000000	
mean	2007.111111	58.500000	668.763889	36.277778	
std	2693.165955	87.995198	1978.997801	85.139690	
min	57.000000	0.000000	12.000000	0.000000	
25%	513.750000	10.000000	20.500000	0.000000	
50%	944.000000	26.500000	41.000000	6.000000	
75%	2590.750000	61.750000	216.500000	23.250000	
max	15149.000000	528.000000	14368.000000	411.000000	

Let's look at the differences between these two sets.

In [16]:
```
poi_finances.describe() - nonpoi_finances.describe()
```

Out[16]:

	salary	deferral_payments	total_payments	loan_advances	bonus	restricte
count	-60.000000	-2.800000e+01	-8.800000e+01	-1.0	-4.900000e+01	
mean	121293.375859	-3.704520e+05	6.188499e+06	80325000.0	1.088175e+06	
std	139128.027285	-4.284915e+05	2.134720e+07	NaN	8.735576e+05	
min	157926.000000	1.127590e+05	9.094500e+04	81125000.0	1.300000e+05	
25%	34068.000000	-5.782000e+04	8.119162e+05	80725000.0	3.750000e+05	
50%	26947.000000	-5.754400e+04	6.979350e+05	80325000.0	5.750000e+05	
75%	126600.000000	-6.606290e+05	6.593195e+05	79925000.0	1.062500e+06	
max	50326.000000	-4.282977e+06	8.630726e+07	79525000.0	-1.000000e+06	

Here we can see that there are a few differences in the financials for these two groups, these may turn out to be very impactful features.

Let's do the same for our email data.

In [17]:
```
poi_email.describe() - nonpoi_email.describe()
```

Out[17]:

	to_messages	from_poi_to_this_person	from_messages	from_this_person_to_poi	shared_
count	-58.000000	-58.000000	-58.000000	-58.000000	
mean	410.031746	39.285714	-368.406746	30.436508	
std	-731.307854	-11.936336	-1173.153226	73.149932	
min	168.000000	13.000000	4.000000	4.000000	
25%	602.000000	34.500000	12.500000	12.500000	
50%	931.000000	35.500000	3.500000	9.500000	
75%	378.500000	74.000000	-115.000000	5.500000	
max	-7158.000000	-288.000000	-11299.000000	198.000000	

Here we may get a feature or two, but I think the financials will be more telling.

Now let's graph some of these features to explore them a little bit.

In [18]:
```
import matplotlib.pyplot as plt
%matplotlib inline
%pylab inline
import seaborn as sns
```

```
Populating the interactive namespace from numpy and matplotlib
```

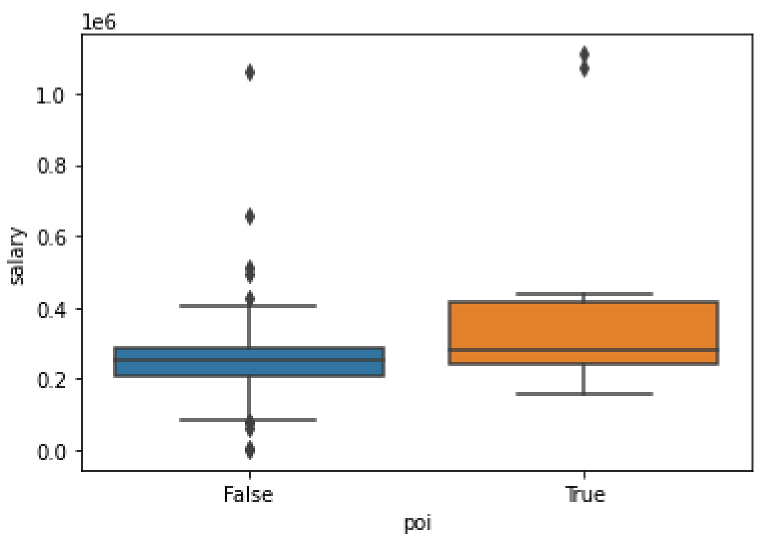
Salary:

In [19]:
```
avg_salary = enron.groupby('poi').mean()['salary']
avg_salary
```

Out[19]:
```
poi
False    262151.506494
True     383444.882353
Name: salary, dtype: float64
```

In [20]:
```
sns.boxplot(x='poi',y='salary',data=enron)
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x15b1b5ca250>
```



Looks like the salary is generally higher for our POIs than for our Non-POIs.

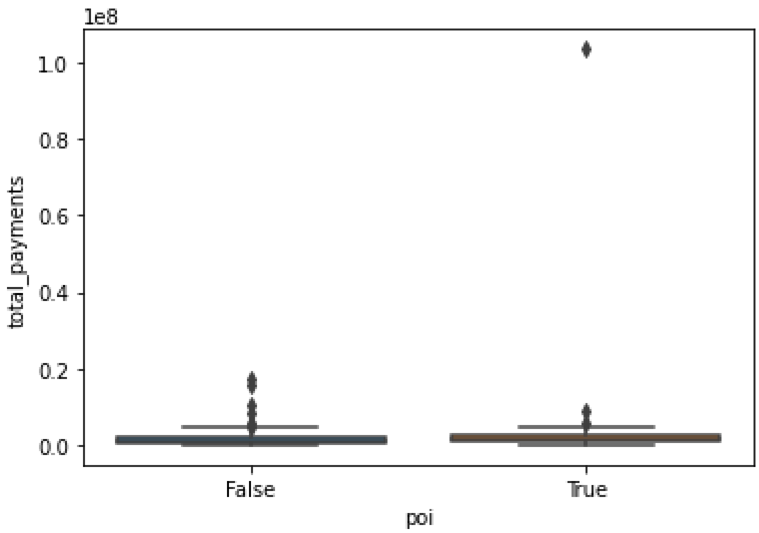
Total Payments:

In [21]:
```
avg_tot_payments = enron.groupby('poi').mean()['total_payments']
avg_tot_payments
```

Out[21]:
```
poi
False    1.725091e+06
True     7.913590e+06
Name: total_payments, dtype: float64
```

In [22]:
```
sns.boxplot(x='poi',y='total_payments',data=enron)
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x15b1bcf6700>

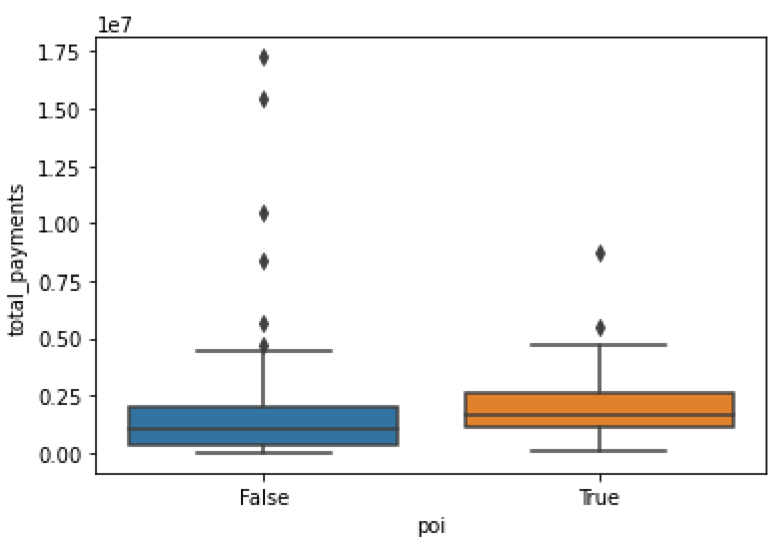


We have a big outlier here, our greediest fraudster, which we can take out in the next graph to get a better visual of this field.

In [23]:
```
enron2 = enron[(enron['total_payments']<30000000)]

sns.boxplot(x='poi',y='total_payments',data=enron2)
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x15b1972cf10>



I would have expected a larger gap here given what was seen in our salary section.

I'm going to add a boolean field here as an additional feature to test called has_tpayment which returns a true if there is a value in this column, or a false if there is a NAN.

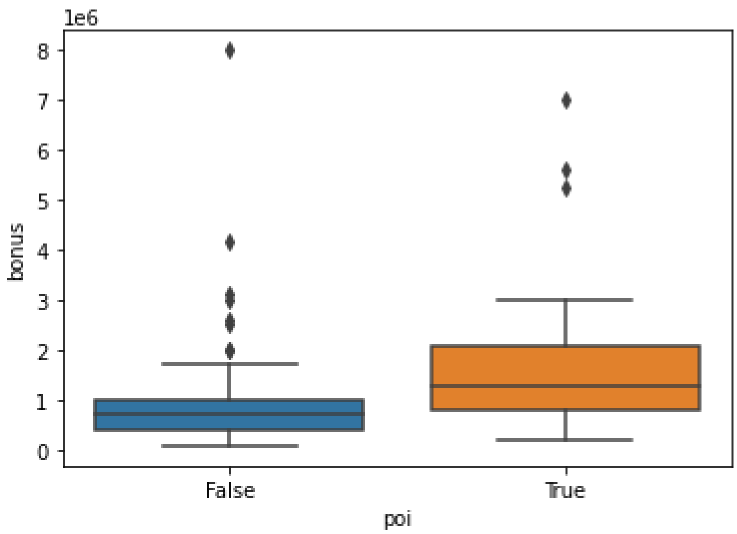
Bonus:

In [24]:
```
avg_bonus = enron.groupby('poi').mean()['bonus']
avg_bonus
```

Out[24]:
```
poi
False    9.868249e+05
True     2.075000e+06
Name: bonus, dtype: float64
```

In [25]:
```
sns.boxplot(x='poi',y='bonus',data=enron)
```

Out[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x15b1be32c40>
```



This could be a usefeal feature, but there are some outliers among the non-POIs that may be problematic.

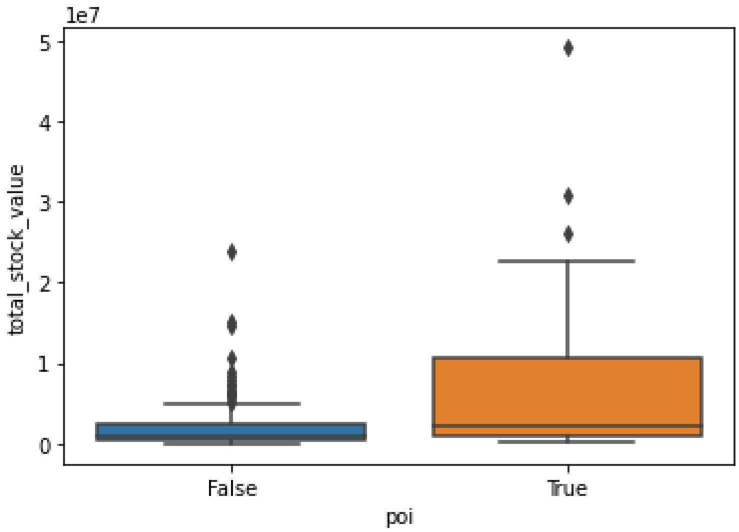
Total Stock Value:

In [26]:
```
avg_tot_sv = enron.groupby('poi').mean()['total_stock_value']
avg_tot_sv
```

Out[26]:
```
poi
False    2.374085e+06
True     9.165671e+06
Name: total_stock_value, dtype: float64
```

In [27]:
```
sns.boxplot(x='poi',y='total_stock_value',data=enron)
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x15b1be97eb0>



This field may turn out to be a benefical feature.

I'm going to add a boolean field here as an additional feature to test called has_tsv which returns a true if there is a value in this column, or a false if there is a NAN.

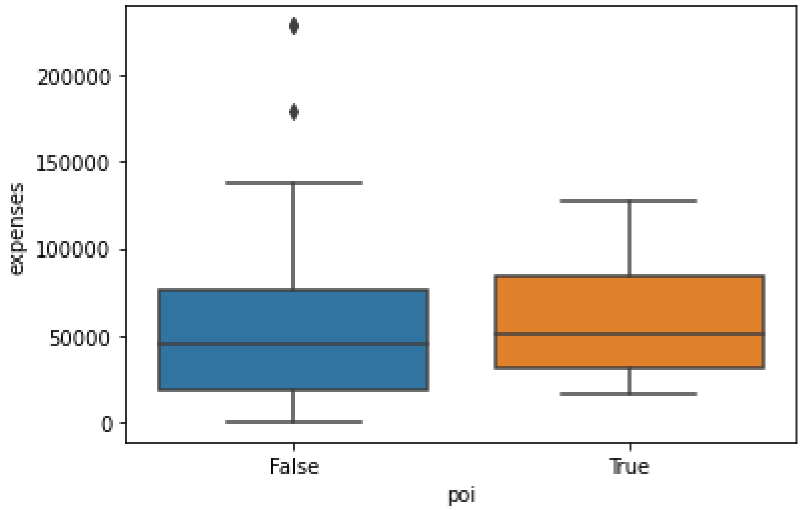
Expenses:

In [28]:
```
avg_expenses = enron.groupby('poi').mean()['expenses']
avg_expenses
```

Out[28]:
```
poi
False    52846.315789
True     59873.833333
Name: expenses, dtype: float64
```

In [29]:
```
sns.boxplot(x='poi',y='expenses',data=enron)
```

Out[29]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x15b1bf04d30>
```



Here, there is not much difference between the expenses recorded by either our POIs and non-POIs.

I'm going to add a boolean field here as an additional feature to test called has_expenses which returns a true if there is a value there, or a false if there is a NAN.

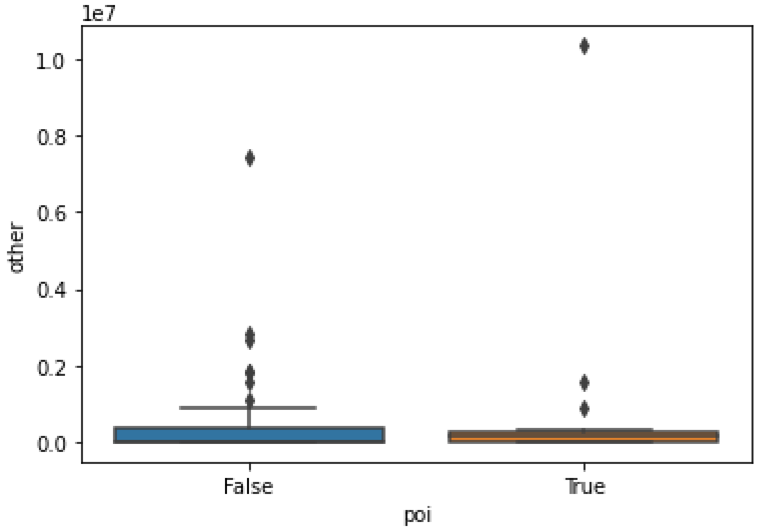
Other:

In [30]:
```
avg_other = enron.groupby('poi').mean()['other']
avg_other
```

Out[30]:
```
poi
False    383128.378378
True     802997.388889
Name: other, dtype: float64
```

In [31]:
```
sns.boxplot(x='poi',y='other',data=enron)
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x15b1bf71a00>



This field doesn't look like it will provide much value as is with our outliers.

I'm going to add a boolean field here as an additional feature to test called has_other which returns a true if there is a value in this column, or a false if there is a NAN.

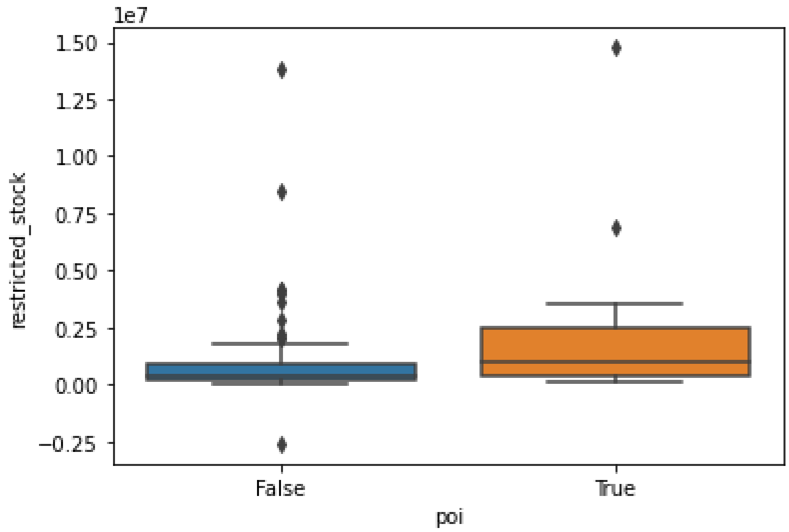
Restricted Stock:

In [32]:
```
avg_res_stock = enron.groupby('poi').mean()['restricted_stock']
avg_res_stock
```

Out[32]:
```
poi
False    9.310073e+05
True     2.318621e+06
Name: restricted_stock, dtype: float64
```

In [33]:
```
sns.boxplot(x='poi',y='restricted_stock',data=enron)
```

Out[33]: <matplotlib.axes._subplots.AxesSubplot at 0x15b1bfd0880>



This looks like a field that could be useful as a feature.

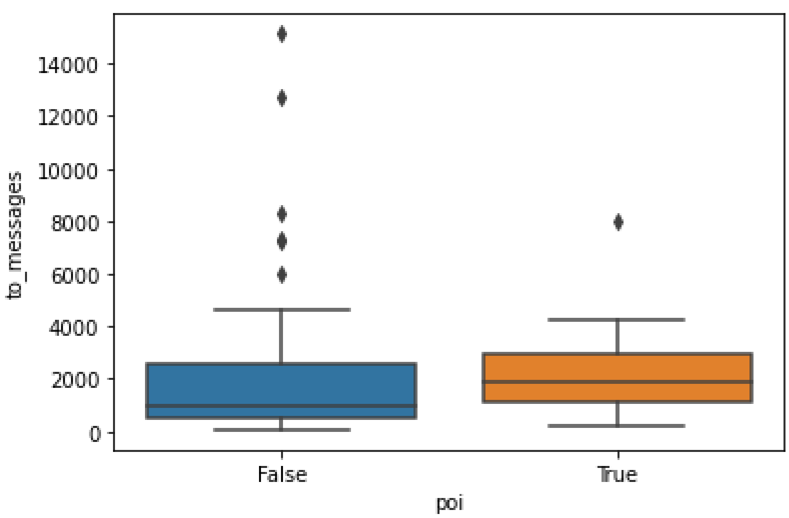
To Messages:

In [36]:
```
avg_to_msg = enron.groupby('poi').mean()['to_messages']
avg_to_msg
```

Out[36]:
```
poi
False    2007.111111
True     2417.142857
Name: to_messages, dtype: float64
```

In [37]:
```
sns.boxplot(x='poi',y='to_messages',data=enron)
```

Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x15b1c03d040>



There doesn't appear to be anything here.

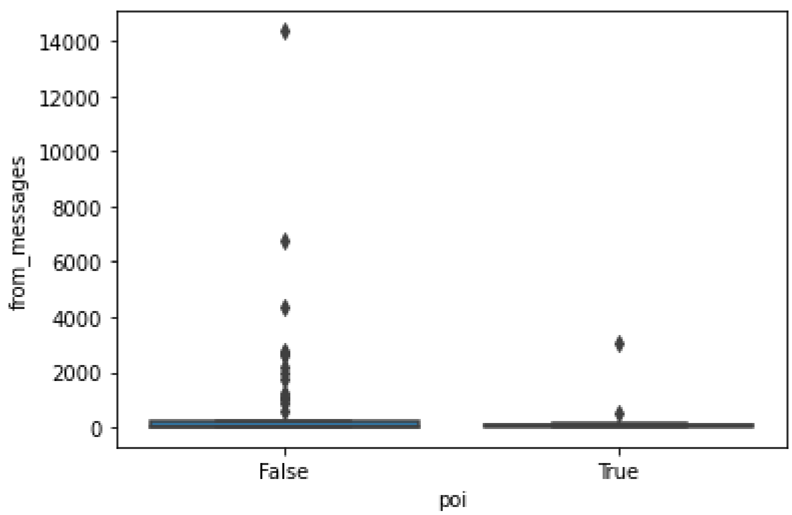
From Messages:

In [38]:
```
avg_from_msg = enron.groupby('poi').mean()['from_messages']
avg_from_msg
```

Out[38]:
```
poi
False    668.763889
True     300.357143
Name: from_messages, dtype: float64
```

In [39]:
```
sns.boxplot(x='poi',y='from_messages',data=enron)
```

Out[39]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x15b1c2b26a0>
```



Outliers on each side for this field, not useful.

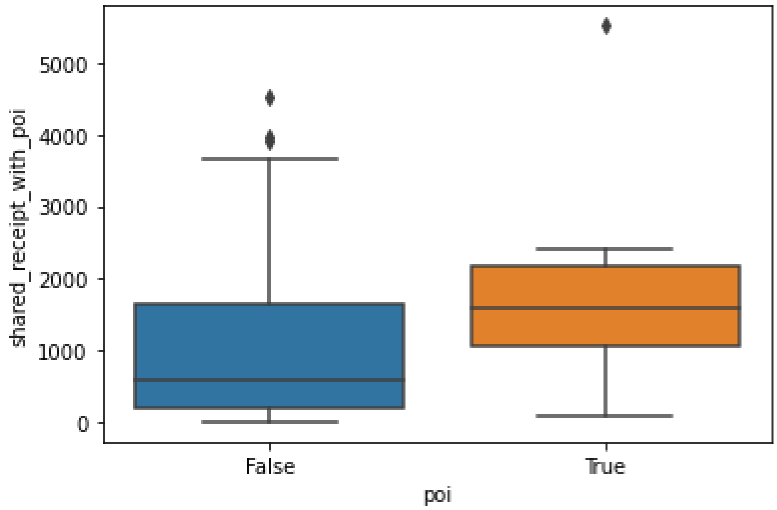
Shared Receipt with POI:

In [42]:
```
avg_sr_poi = enron.groupby('poi').mean()['shared_receipt_with_poi']
avg_sr_poi
```

Out[42]:
```
poi
False    1058.527778
True     1783.000000
Name: shared_receipt_with_poi, dtype: float64
```

In [43]:
```
sns.boxplot(x='poi',y='shared_receipt_with_poi',data=enron)
```

Out[43]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x15b1c323610>
```



This field may also be useful. I think that is the end of our features.

Outliers

Most outliers are datapoints that must be considered such as Ken Lay's extreme salary, however, there were only a few in our data that need to be addressed.

The first of which is the total column as it provides no additional value and should be removed.

In [44]:
```
enron_data.pop('TOTAL')
sorted(enron_data.keys())
```

Out[44]:
```
['ALLEN PHILLIP K',
 'BADUM JAMES P',
 'BANNANTINE JAMES M',
 'BAXTER JOHN C',
 'BAY FRANKLIN R',
 'BAZELIDES PHILIP J',
 'BECK SALLY W',
 'BELDEN TIMOTHY N',
 'BELFER ROBERT',
 'BERBERIAN DAVID',
 'BERGSIEKER RICHARD P',
 'BHATNAGAR SANJAY',
 'BIBI PHILIPPE A',
 'BLACHMAN JEREMY M',
 'BLAKE JR. NORMAN P',
 'BOWEN JR RAYMOND M',
 'BROWN MICHAEL',
 'BUCHANAN HAROLD G',
 'BUTTS ROBERT H',
 'BUY RICHARD B',
 'CALGER CHRISTOPHER F',
 'CARTER REBECCA C',
 'CAUSEY RICHARD A',
 'CHAN RONNIE',
 'CHRISTODOULOU DIOMEDES',
 'CLINE KENNETH W',
 'COLWELL WESLEY',
 'CORDES WILLIAM R',
 'COX DAVID',
 'CUMBERLAND MICHAEL S',
 'DEFFNER JOSEPH M',
 'DELAINEY DAVID W',
 'DERRICK JR. JAMES V',
 'DETMERING TIMOTHY J',
 'DIETRICH JANET R',
 'DIMICHELE RICHARD G',
 'DODSON KEITH',
 'DONAHUE JR JEFFREY M',
 'DUNCAN JOHN H',
 'DURAN WILLIAM D',
 'ECHOLS JOHN B',
 'ELLIOTT STEVEN',
 'FALLON JAMES B',
 'FASTOW ANDREW S',
 'FITZGERALD JAY L',
 'FOWLER PEGGY',
 'FOY JOE',
 'FREVERT MARK A',
 'FUGH JOHN L',
 'GAHN ROBERT S',
 'GARLAND C KEVIN',
 'GATHMANN WILLIAM D',
 'GIBBS DANA R',
 'GILLIS JOHN',
 'GLISAN JR BEN F',
 'GOLD JOSEPH',
 'GRAMM WENDY L',
```

```
'GRAY RODNEY',
'HAEDICKE MARK E',
'HANNON KEVIN P',
'HAUG DAVID L',
'HAYES ROBERT E',
'HAYSLETT RODERICK J',
'HERMANN ROBERT J',
'HICKERSON GARY J',
'HIRKO JOSEPH',
'HORTON STANLEY C',
'HUGHES JAMES A',
'HUMPHREY GENE E',
'IZZO LAWRENCE L',
'JACKSON CHARLENE R',
'JAEDICKE ROBERT',
'KAMINSKI WINCENTY J',
'KEAN STEVEN J',
'KISHKILL JOSEPH G',
'KITCHEN LOUISE',
'KOENIG MARK E',
'KOPPER MICHAEL J',
'LAVORATO JOHN J',
'LAY KENNETH L',
'LEFF DANIEL P',
'LEMAISTRE CHARLES',
'LEWIS RICHARD',
'LINDHOLM TOD A',
'LOCKHART EUGENE E',
'LOWRY CHARLES P',
'MARTIN AMANDA K',
'MCCARTY DANNY J',
'MCCLELLAN GEORGE',
'MCCONNELL MICHAEL S',
'MCDONALD REBECCA',
'MCMAHON JEFFREY',
'MENDELSOHN JOHN',
'METTS MARK',
'MEYER JEROME J',
'MEYER ROCKFORD G',
'MORAN MICHAEL P',
'MORDAUNT KRISTINA M',
'MULLER MARK S',
'MURRAY JULIA H',
'NOLES JAMES L',
'OLSON CINDY K',
'OVERDYKE JR JERE C',
'PAI LOU L',
'PEREIRA PAULO V. FERRAZ',
'PICKERING MARK R',
'PIPER GREGORY F',
'PIRO JIM',
'POWERS WILLIAM',
'PRENTICE JAMES',
'REDMOND BRIAN L',
'REYNOLDS LAWRENCE',
'RICE KENNETH D',
'RIEKER PAULA H',
```

```
 'SAVAGE FRANK',
 'SCRIMSHAW MATTHEW',
 'SHANKMAN JEFFREY A',
 'SHAPIRO RICHARD S',
 'SHARP VICTORIA T',
 'SHELBY REX',
 'SHERRICK JEFFREY B',
 'SHERRIFF JOHN R',
 'SKILLING JEFFREY K',
 'STABLER FRANK',
 'SULLIVAN-SHAKLOVITZ COLLEEN',
 'SUNDE MARTIN',
 'TAYLOR MITCHELL S',
 'THE TRAVEL AGENCY IN THE PARK',
 'THORN TERENCE H',
 'TILNEY ELIZABETH A',
 'UMANOFF ADAM S',
 'URQUHART JOHN A',
 'WAKEHAM JOHN',
 'WALLS JR ROBERT H',
 'WALTERS GARETH W',
 'WASAFF GEORGE',
 'WESTFAHL RICHARD K',
 'WHALEY DAVID A',
 'WHALLEY LAWRENCE G',
 'WHITE JR THOMAS E',
 'WINOKUR JR. HERBERT S',
 'WODRASKA JOHN',
 'WROBEL BRUCE',
 'YEAGER F SCOTT',
 'YEAP SOON']
```

Next we can get rid of Eugene, as he doens't have any values, all fields are null.

In [45]:
```
enron_data.pop('LOCKHART EUGENE E')
```

Out[45]:
```
{'salary': 'NaN',
 'to_messages': 'NaN',
 'deferral_payments': 'NaN',
 'total_payments': 'NaN',
 'loan_advances': 'NaN',
 'bonus': 'NaN',
 'email_address': 'NaN',
 'restricted_stock_deferred': 'NaN',
 'deferred_income': 'NaN',
 'total_stock_value': 'NaN',
 'expenses': 'NaN',
 'from_poi_to_this_person': 'NaN',
 'exercised_stock_options': 'NaN',
 'from_messages': 'NaN',
 'other': 'NaN',
 'from_this_person_to_poi': 'NaN',
 'poi': False,
 'long_term_incentive': 'NaN',
 'shared_receipt_with_poi': 'NaN',
 'restricted_stock': 'NaN',
 'director_fees': 'NaN'}
```

Clean, Select, and Scale Features:

In [ ]:

In [46]:
```
import sys
import pickle

from feature_format import featureFormat, targetFeatureSplit
from tester import dump_classifier_and_data
```

In [47]:
```
features_full_list = enron.columns.tolist()
features_full_list.pop(0) # remove 'name'
features_full_list.pop(19) # remove 'email_address'
features_full_list.pop(14) # remove 'poi' and add to beginning
features_list = ['poi']
for n in features_full_list:
    features_list.append(n)
features_list
```

Out[47]:
```
['poi',
 'salary',
 'to_messages',
 'deferral_payments',
 'total_payments',
 'loan_advances',
 'bonus',
 'email_address',
 'restricted_stock_deferred',
 'deferred_income',
 'total_stock_value',
 'expenses',
 'from_poi_to_this_person',
 'exercised_stock_options',
 'from_messages',
 'from_this_person_to_poi',
 'poi',
 'long_term_incentive',
 'shared_receipt_with_poi',
 'director_fees']
```

In [48]:
```
features_list
```

Out[48]:
```
['poi',
 'salary',
 'to_messages',
 'deferral_payments',
 'total_payments',
 'loan_advances',
 'bonus',
 'email_address',
 'restricted_stock_deferred',
 'deferred_income',
 'total_stock_value',
 'expenses',
 'from_poi_to_this_person',
 'exercised_stock_options',
 'from_messages',
 'from_this_person_to_poi',
 'poi',
 'long_term_incentive',
 'shared_receipt_with_poi',
 'director_fees']
```

Now to add the additional boolean features.
Total Payments:

In [49]:
```
for name in enron_data:
    data = enron_data[name]
    payment = data["total_payments"]
    if payment != 'NaN':
        has_tpayment = True
    else:
        has_tpayment = False
    enron_data[name]["has_tpayment"] = has_tpayment
```

Total Stock Value:

In [50]:
```
for name in enron_data:
    data = enron_data[name]
    payment = data["total_stock_value"]
    if payment != 'NaN':
        has_tsv = True
    else:
        has_tsv = False
    enron_data[name]["has_tsv"] = has_tsv
```

Expenses:

In [51]:
```
for name in enron_data:
    data = enron_data[name]
    payment = data["expenses"]
    if payment != 'NaN':
        has_expenses = True
    else:
        has_expenses = False
    enron_data[name]["has_expenses"] = has_expenses
```

Other:

In [52]:
```
for name in enron_data:
    data = enron_data[name]
    payment = data["other"]
    if payment != 'NaN':
        has_other = True
    else:
        has_other = False
    enron_data[name]["has_other"] = has_other
```

Email Features:

In [55]:
```
def computeFraction( poi_messages, all_messages ):
    """ given a number messages to/from POI (numerator)
        and number of all messages to/from a person (denominator),
        return the fraction of messages to/from that person
        that are from/to a POI
   """
    fraction = 0.
    if poi_messages != 'NaN' and all_messages != 'NaN':
        fraction = float(poi_messages)/all_messages


    return fraction


for name in enron_data:

    data_point = enron_data[name]

    from_poi_to_this_person = data_point["from_poi_to_this_person"]
    to_messages = data_point["to_messages"]
    fraction_from_poi = computeFraction( from_poi_to_this_person, to_messages
)

    enron_data[name]["fraction_from_poi"] = fraction_from_poi

    from_this_person_to_poi = data_point["from_this_person_to_poi"]
    from_messages = data_point["from_messages"]
    fraction_to_poi = computeFraction( from_this_person_to_poi, from_messages
)

    enron_data[name]["fraction_to_poi"] = fraction_to_poi

for name in enron_data:
    data_point = enron_data[name]

    bonus = data_point['bonus']
    if bonus == 'NaN':
        bonus = 0.0
    options = data_point['exercised_stock_options']
    if options == 'NaN':
        options = 0.0
    total = bonus+options

    enron_data[name]['total_beso'] = total

for name in enron_data:
    data_point = enron_data[name]

    total_payments = data_point['total_payments']
    if total_payments == 'NaN':
        total_payments = 0.0
    total_stock = data_point['total_stock_value']
    if total_stock == 'NaN':
        total_stock = 0.0
    total = (total_payments + total_stock)/1000000
```

```
    enron_data[name]['num_millions'] = total
```

Let's make sure we have our expected features.

In [56]:
```
enron_data['LAY KENNETH L']
```

Out[56]:
```
{'salary': 1072321,
 'to_messages': 4273,
 'deferral_payments': 202911,
 'total_payments': 103559793,
 'loan_advances': 81525000,
 'bonus': 7000000,
 'email_address': 'kenneth.lay@enron.com',
 'restricted_stock_deferred': 'NaN',
 'deferred_income': -300000,
 'total_stock_value': 49110078,
 'expenses': 99832,
 'from_poi_to_this_person': 123,
 'exercised_stock_options': 34348384,
 'from_messages': 36,
 'other': 10359729,
 'from_this_person_to_poi': 16,
 'poi': True,
 'long_term_incentive': 3600000,
 'shared_receipt_with_poi': 2411,
 'restricted_stock': 14761694,
 'director_fees': 'NaN',
 'has_tpayment': True,
 'has_tsv': True,
 'has_expenses': True,
 'has_other': True,
 'fraction_from_poi': 0.028785396676807865,
 'fraction_to_poi': 0.4444444444444444,
 'total_beso': 41348384,
 'num_millions': 152.669871}
```

Now the features look good. We can get into feature selection.

In [57]:
```
from feature_format import featureFormat, targetFeatureSplit
features_list1 = ['poi','salary','to_messages','deferral_payments','total_paym
ents','loan_advances','bonus','restricted_stock_deferred','deferred_income','t
otal_stock_value','expenses','exercised_stock_options','from_messages','other'
,'long_term_incentive','shared_receipt_with_poi', 'director_fees'] # You will
 need to use more features

features_list2 = ['poi', 'salary', 'to_messages', 'deferral_payments', 'total_
payments', 'loan_advances', 'bonus', 'restricted_stock_deferred', 'deferred_in
come', 'total_stock_value', 'expenses', 'exercised_stock_options', 'from_messa
ges', 'other', 'long_term_incentive', 'shared_receipt_with_poi', 'director_fee
s', 'has_tpayment', 'has_tsv', 'has_expenses', 'has_other', 'fraction_from_po
i', 'fraction_to_poi', 'total_beso', 'num_millions']

my_dataset = enron_data

data1 = featureFormat(my_dataset, features_list1, sort_keys = True)
labels1, features1 = targetFeatureSplit(data1)
data2 = featureFormat(my_dataset, features_list2, sort_keys = True)
labels2, features2 = targetFeatureSplit(data2)
```

In [58]:
```
from sklearn.naive_bayes import GaussianNB
gnb_clf = GaussianNB()

gnb_clf.fit(features1, labels1)
print("Accuracy without new features:", gnb_clf.score(features1, labels1))
gnb_clf.fit(features2, labels2)
print("Accuracy with new features:", gnb_clf.score(features2, labels2))
gnb_clf.fit(features2, labels2)
```

```
Accuracy without new features: 0.8125
Accuracy with new features: 0.8402777777777778
```

Out[58]:
```
GaussianNB()
```

In [272]:
```
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.model_selection import GridSearchCV

skb = SelectKBest(k=11)
selected_features = skb.fit_transform(features2,labels2)
features_selected=[features_list2[i+1] for i in skb.get_support(indices=True)]

print('Features selected by SelectKBest:')
#print(selected_features)
print(features_selected)
features_list = features_selected
```

```
Features selected by SelectKBest:
['salary', 'bonus', 'deferred_income', 'total_stock_value', 'exercised_stock_
options', 'long_term_incentive', 'has_expenses', 'has_other', 'fraction_to_po
i', 'total_beso', 'num_millions']
```

In [273]:
```
#features_list = ['poi', 'salary', 'bonus', 'deferred_income', 'total_stock_value', 'exercised_stock_options', 'has_expenses', 'has_other', 'fraction_to_poi', 'total_beso', 'num_millions']
features_list.insert(0,'poi')
data = featureFormat(my_dataset, features_list, sort_keys = True)
labels, features = targetFeatureSplit(data)
```

In [274]:
```
from sklearn.model_selection import train_test_split
features_train, features_test, labels_train, labels_test = train_test_split(features, labels, test_size = 0.3, random_state=42)
```

In [275]:
```
#GaussianNB
from sklearn.naive_bayes import GaussianNB
clf1 = GaussianNB()
clf1.fit(features_train, labels_train)
pred1 = clf1.predict(features_test)
```

In [276]:
```
#RandomForest
from sklearn.ensemble import RandomForestClassifier
from sklearn import model_selection
#clf = RandomForestClassifier(n_estimators=100, min_samples_split=3)
clf2 = RandomForestClassifier(n_estimators=30, min_samples_split=5)
parameters = {'min_samples_split':[2,3,4,5,6], 'n_estimators': [10,20,30,40, 50, 60, 70, 80, 90, 100]}
random = RandomForestClassifier()
#clf2 = model_selection.GridSearchCV(random, parameters)
clf2.fit(features_train, labels_train)
pred2 = clf2.predict(features_test)
```

In [277]:
```
#adaboost
from sklearn.ensemble import AdaBoostClassifier
from sklearn import model_selection
ada = AdaBoostClassifier()
parameters = {'n_estimators':[10,50,100], 'random_state': [None, 0, 42, 138]}
#clf3 = model_selection.GridSearchCV(ada, parameters)
clf3 = AdaBoostClassifier(n_estimators=50, random_state=138)
clf3.fit(features_train, labels_train)
pred3 = clf3.predict(features_test)
```

In [278]:
```
#DecisionTree
from sklearn import model_selection, tree
parameters = {'min_samples_split':[2,3,4,5,6,7,8,9], 'min_samples_leaf':[1,2,3], 'random_state':[None, 0, 42] }
#clf4 = model_selection.GridSearchCV(tree, parameters)
clf4 = tree.DecisionTreeClassifier(min_samples_split = 5, random_state=42)
clf4 = clf4.fit(features_train, labels_train)
pred4 = clf4.predict(features_test)
```

In [279]:
```
from tester import dump_classifier_and_data, test_classifier

test_classifier(clf1, my_dataset, features_list)
test_classifier(clf2, my_dataset, features_list)
test_classifier(clf3, my_dataset, features_list)
test_classifier(clf4, my_dataset, features_list)
```

```
GaussianNB()
        Accuracy: 0.87333       Precision: 0.52874      Recall: 0.46000 F1: 
0.49198 F2: 0.47228
        Total predictions: 1500 True positives:   92    False positives:   82
False negatives:  108   True negatives: 1218

RandomForestClassifier(min_samples_split=5, n_estimators=30)
        Accuracy: 0.87800       Precision: 0.60494      Recall: 0.24500 F1: 
0.34875 F2: 0.27809
        Total predictions: 1500 True positives:   49    False positives:   32
False negatives:  151   True negatives: 1268

AdaBoostClassifier(random_state=138)
        Accuracy: 0.87000       Precision: 0.51572      Recall: 0.41000 F1: 
0.45682 F2: 0.42753
        Total predictions: 1500 True positives:   82    False positives:   77
False negatives:  118   True negatives: 1223

DecisionTreeClassifier(min_samples_split=5, random_state=42)
        Accuracy: 0.80933       Precision: 0.26630      Recall: 0.24500 F1: 
0.25521 F2: 0.24898
        Total predictions: 1500 True positives:   49    False positives:  135
False negatives:  151   True negatives: 1165
```

Based on this, the Naive Bayes is the algorithm I'll go with.

I iterated through testing my Naive Bayes model against different K values to optimize my KSelectBest, the results is below.

k	Acc	Prec	Recall
1	84	47	22
3	836	45	30
5	853	54	295
7	846	40	31
9	846	40	31
10	873	528	46
11	873	528	46
12	864	486	37
13	858	459	37
14	833	362	33
15	834	364	33

Questions:

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 goal of this project is to come up with a machine learning algorithm that can help an investigator identify "Persons of Interest" in the Enron fraud case. Machine learning is useful in trying to accomplish this task because it allows for the classification of multiple data points quickly and efficiently. The dataset I am using is from the fraud case against Enron, as it was entered into public record, which includes financial and email information for many Enron employees. There were a couple of outliers in my data, the first of which was the total entry, which was removed. There was also an individual(LOCKHART EUGENE E) with all NaN values, so he was removed as well. There was also a company(TRAVEL AGENCY), which was left alone in case the company was implicated somehow. A few of our POIs presented as outliers due to the vast financial reward they were reaping, but they were left in as well.

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”, “intelligently select features”, “properly scale features”]

-['poi', 'salary', 'bonus', 'deferred_income', 'total_stock_value', 'exercised_stock_options', 'long_term_incentive', 'has_expenses', 'has_other', 'fraction_to_poi', 'total_beso', 'num_millions'] were the features I chose to use based on my SelectKBest output. The final 5 features were all ones that I created, the first 2 were binary fields based on whether a numeric value existed in that field, there were a total of 4 features like that I created, my thought process for creating these features was that while I was investigating my data, I saw that my POIs all had values for these fields, while some non-POIs did not have data here, so at the very least these features should be able to be used to reduce false positives. The other 3 were pretty close to what I had done in the lessons leading up to this project. I did not need to do any scaling, as the models I chose (naive bayes, decision trees, random forrest, and adaBoost) do not require any feature scaling.
To select my K value, I methodically ran through different values with my chosen model (Naive Bayes) from 2 -15, and found that 11 was what I wanted to move forward with. (table above)

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 going with the naive bayes. I tried a naive bayes, random forest, adaBoost, and decision tree. The differences were more apparent than I thought they would be, I ended up with the highest in terms of accuracy, but it was the recall and precision that really put me over the top in chosing this one.

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? What parameters did you tune? (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 items: “discuss parameter tuning”, “tune the algorithm”]

-Tuning the parameters of an algorithm is the process of predefining the parameters that can't be learned directly from the training process. My tuning process was done through my model selection process, where I modified parameters in the random forest such as min_sample_split, n_estimator, min_sample_leaf and random_state to try get the best performance before my model was selected. My final model did not need to be tuned as there were no parameters to tune.

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”] -The evaluation metrics I used were precision and recall. The precision represents true positives(correctly identified POIs)/(true positives(correctly identified POIs)+false positives(non-POIs incorrectly labeled as POIs)).
The recall represents true positives(correctly identified POIs)/(true positives(correctly identified POIs)+false negatives(POIs incorrectly labeled as non-POIs))

“I hereby confirm that this submission is my work. I have cited above the origins of any parts of the submission that were taken from Websites, books, forums, blog posts, github repositories, etc

In [ ]:

```

```