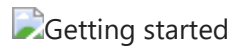


# Machine Learning to Identify Fraud in Enron Corpus

## Final Project for Python for Data Science Course

### Group Member:

- FATHIAH -
- MATHEW GEORGE Jithin
- TIATSE SOUOP Varesse
- SZEPEK Maria Sofie



## Context and Enron Corpus

In late 2001, Enron, an American energy company, filed for bankruptcy after one of the largest financial scandals in corporate history. After the company's collapse, over 600,000 emails generated by 158 Enron employees - now known as the Enron Corpus - were acquired by the Federal Energy Regulatory Commission during its investigation. The data was then uploaded online, and since then, a number of people and organizations have graciously prepared, cleaned and organized the dataset that is available to the public today (a few years later, financial data of top Enron executives were released following their trial).

## Project Description and Goal

The aim of this project is to apply machine learning techniques to build a predictive model that identifies Enron employees that may have committed fraud based on their financial and email data.

The dataset has:

- 14 financial features (salary, bonus, etc.),
- 6 email features (to and from messages, etc.)
- A Boolean label that denotes whether a person is a person-of-interest (POI) or not (established from credible news sources).

It is these features that will be explored, cleaned, and then put through various machine learning algorithms, before finally tuning them and checking its accuracy (precision and recall).

**The objective is to get a precision and recall score of at least 0.48**

First, the dataset will be manually explored to find outliers and trends and generally understand the data. Certain useful financial or email-based features will be chosen (manually and automatically using sklearn functions) and ensemble features created from those available, and then put through appropriate feature scaling. Then, numerous algorithms with parameter tuning will be trained and tested on the data.

The detailed results of the final algorithm, will be described in detail. The validation and evaluation metrics will be also shown and the reasoning behind their choice and its importance will be carefully explained. Finally, other ideas involving feature selection, feature scaling, other algorithms and usage of email texts will be discussed.

## Initial Preparation/Importing Libraries

```
In [1]: !pip install feature_format
```

Requirement already satisfied: feature\_format in /root/venv/lib/python3.7/site-packages (0.1)  
Requirement already satisfied: numpy in /shared-libs/python3.7/py/lib/python3.7/site-packages (from feature\_format) (1.19.5)  
WARNING: You are using pip version 20.1.1; however, version 21.3.1 is available.  
You should consider upgrading via the '/root/venv/bin/python -m pip install --upgrade pip' command.

In [2]:

```
import sys
import pickle
sys.path.append("../tools/")
from feature_format import featureFormat, targetFeatureSplit
from tester import dump_classifier_and_data
import pandas as pd
import sys
import pickle
import csv
import matplotlib.pyplot as plt

sys.path.append("../tools/")
from feature_format import featureFormat, targetFeatureSplit
#from poi_data import *
from sklearn.feature_selection import SelectKBest
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedShuffleSplit

from numpy import mean

from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate

from sklearn.metrics import accuracy_score, precision_score, recall_score

from sklearn.model_selection import train_test_split
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
plt.rc("font", size=14)
```

## TASK 1: Features

### Create features list and import necessary files

In [3]:

```
### Selecting features
target_label = 'poi'

email_features_list = [
    'from_messages',
    'from_poi_to_this_person',
    'from_this_person_to_poi',
    'shared_receipt_with_poi',
    'to_messages',
]

financial_features_list = [
    'bonus',
```

```

'defferral_payments',
'deferred_income',
'director_fees',
'exercised_stock_options',
'expenses',
'loan_advances',
'long_term_incentive',
'other',
'restricted_stock',
'restricted_stock_deferred',
'salary',
'total_payments',
'total_stock_value',
]

features_list = [target_label] + financial_features_list + email_features_list

```

```

In [6]: ### Load the dictionary containing the dataset
with open("final_project_dataset.pkl", "rb") as data_file:
    data_dict = pickle.load(data_file)

```

```

In [7]: ### 1.1.0 Explore csv file
def make_csv(data_dict):
    """ generates a csv file from a data set"""
    fieldnames = ['name'] + data_dict.itervalues().next().keys()
    with open('data.csv', 'w') as csvfile:
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        for record in data_dict:
            person = data_dict[record]
            person['name'] = record
            assert set(person.keys()) == set(fieldnames)
            writer.writerow(person)

```

## Initial data exploration

```

In [9]: ### 1.1.1 Dataset Exploration
print('# Exploratory Data Analysis #')
data_dict.keys()
print('Total number of data points: %d' % len(data_dict.keys()))
num_poi = 0
for name in data_dict.keys():
    if data_dict[name]['poi'] == True:
        num_poi += 1
print('Number of Persons of Interest: %d' % num_poi)
print('Number of people without Person of Interest label: %d' % (len(data_dict.keys()) - num_poi))

```

```

# Exploratory Data Analysis #
Total number of data points: 146
Number of Persons of Interest: 18
Number of people without Person of Interest label: 128

```

```

In [10]: ### 1.1.2 Feature Exploration
all_features = data_dict['ALLEN PHILLIP K'].keys()
print('Each person has %d features available' % len(all_features))
### Evaluate dataset for completeness
missing_values = {}
for feature in all_features:
    missing_values[feature] = 0
for person in data_dict.keys():

```

```

records = 0
for feature in all_features:
    if data_dict[person][feature] == 'NaN':
        missing_values[feature] += 1
    else:
        records += 1

```

Each person has 21 features available

```

In [11]: ### Print results of completeness analysis
print('Number of Missing Values for Each Feature:')
for feature in all_features:
    print("%s: %d" % (feature, missing_values[feature]))

```

Number of Missing Values for Each Feature:

```

salary: 51
to_messages: 60
deferral_payments: 107
total_payments: 21
loan_advances: 142
bonus: 64
email_address: 35
restricted_stock_deferred: 128
deferred_income: 97
total_stock_value: 20
expenses: 51
from_poi_to_this_person: 60
exercised_stock_options: 44
from_messages: 60
other: 53
from_this_person_to_poi: 60
poi: 0
long_term_incentive: 80
shared_receipt_with_poi: 60
restricted_stock: 36
director_fees: 129

```

In [ ]:

## TASK 2: Remove Outliers

We detect a few features with outliers and remove them to make the analysis more robust.

```

In [12]: def PlotOutlier(data_dict, feature_x, feature_y):
    """ Plot with flag = True in Red """
    data = featureFormat(data_dict, [feature_x, feature_y, 'poi'])
    for point in data:
        x = point[0]
        y = point[1]
        poi = point[2]
        if poi:
            color = 'red'
        else:
            color = 'blue'
        plt.scatter(x, y, color=color)
    plt.xlabel(feature_x)
    plt.ylabel(feature_y)
    plt.show()

# 2.1 Visualise outliers
print(PlotOutlier(data_dict, 'total_payments', 'total_stock_value'))

```

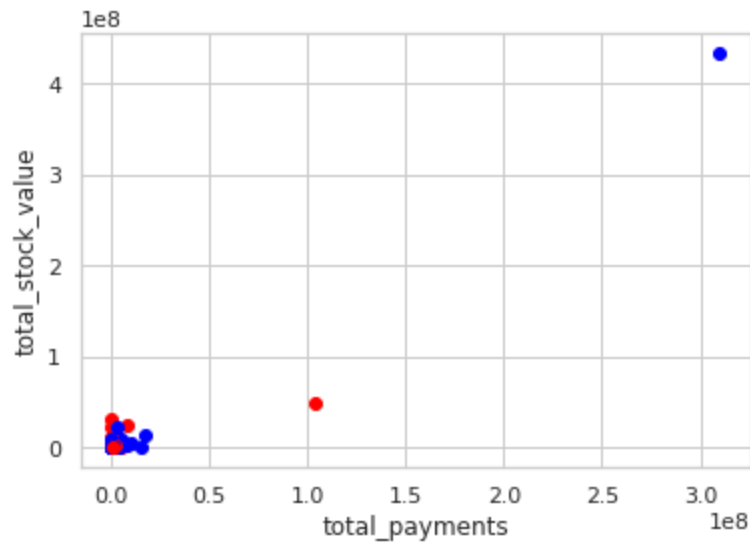
```

print(PlotOutlier(data_dict, 'from_poi_to_this_person', 'from_this_person_to_poi'))
print(PlotOutlier(data_dict, 'salary', 'bonus'))
#Remove outlier TOTAL line in pickle file.
data_dict.pop( 'TOTAL', 0 )

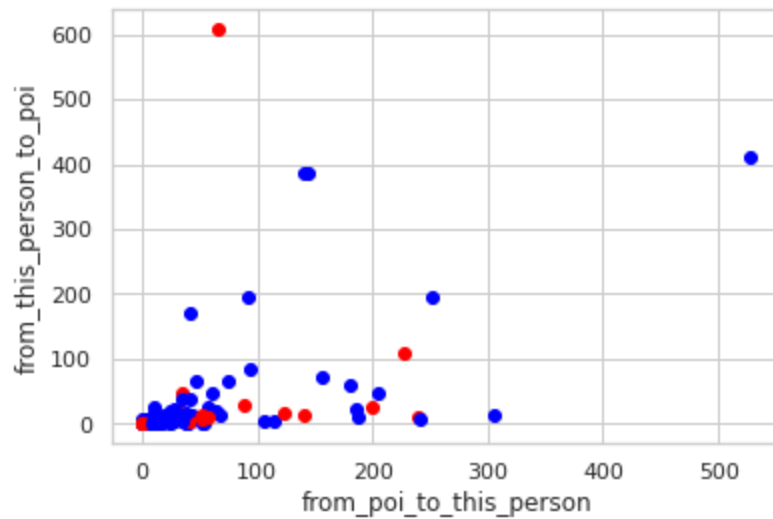
# 2.2 Function to remove outliers
def remove_outlier(dict_object, keys):
    """ removes list of outliers keys from dict object """
    for key in keys:
        dict_object.pop(key, 0)

outliers = ['TOTAL', 'THE TRAVEL AGENCY IN THE PARK', 'LOCKHART EUGENE E', 'FREVERT MARK A']
remove_outlier(data_dict, outliers)

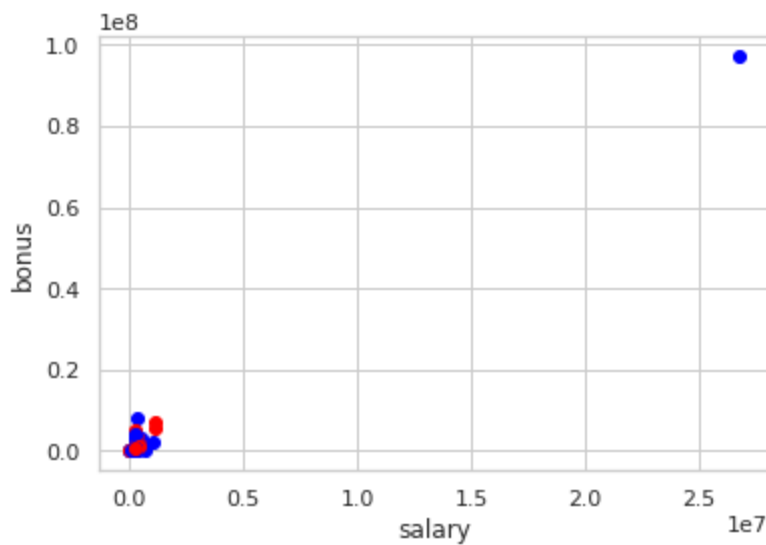
```



None



None



None

## Plotting after outliers removal

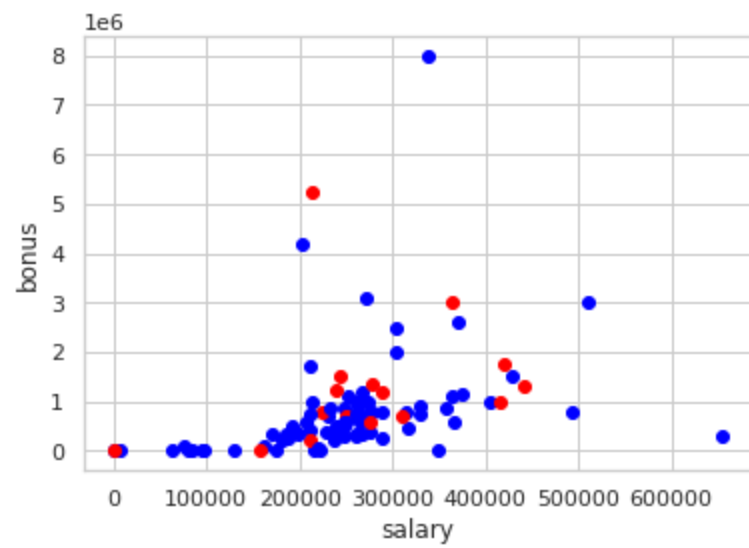
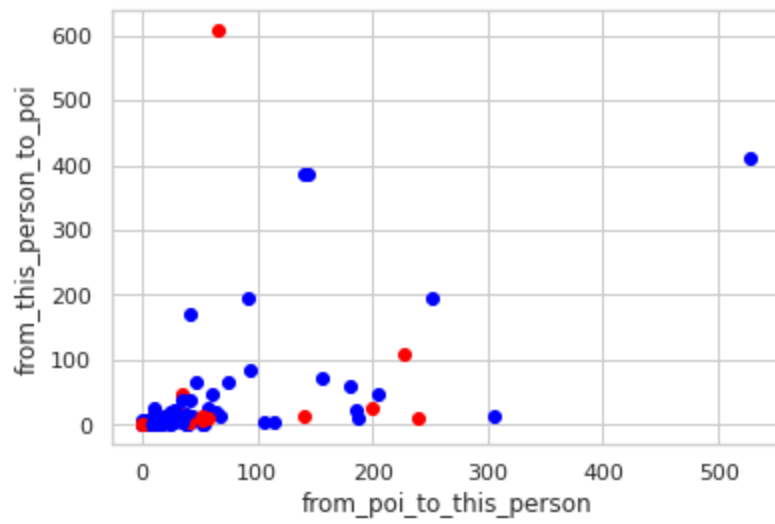
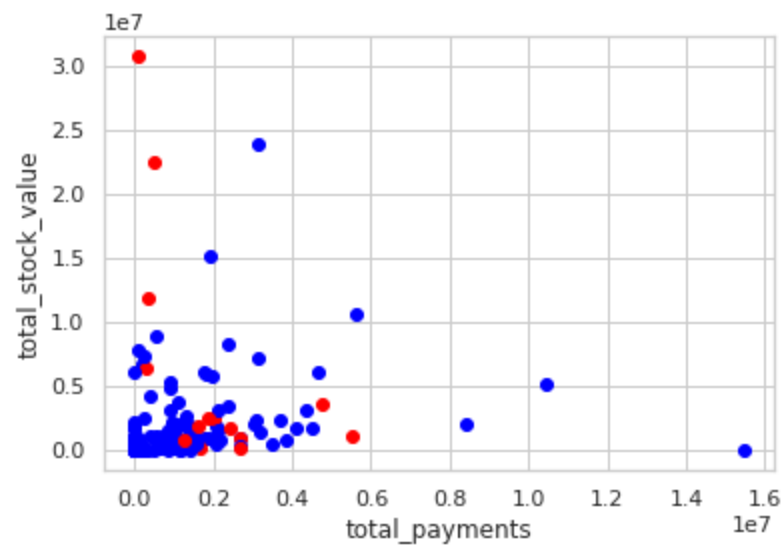
In [13]:

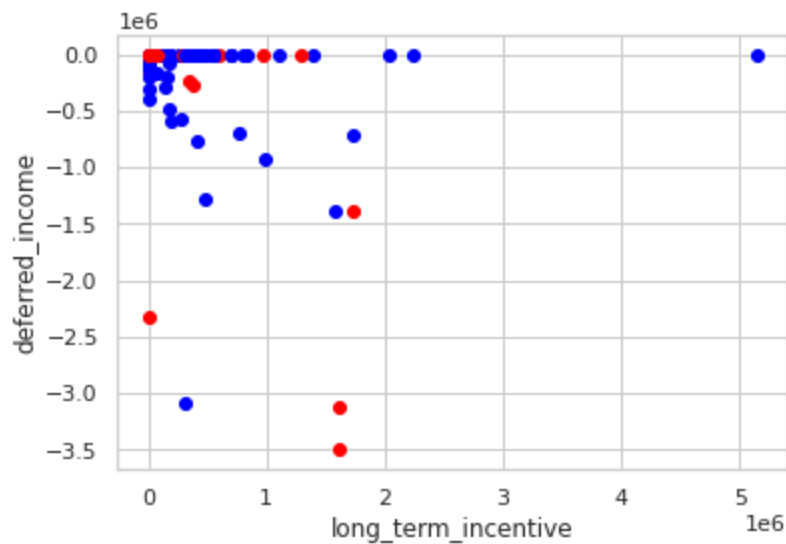
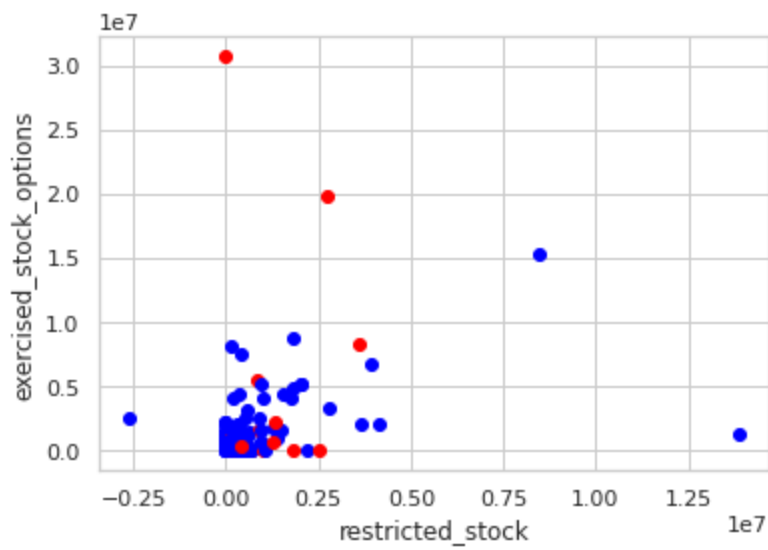
```
import matplotlib.pyplot
def showData(data_set, first_feature, second_feature):
    data = featureFormat(data_set, [first_feature, second_feature, 'poi'])
    for point in data:
        x = point[0]
        y = point[1]
        poi = point[2]
        if poi:
            color = 'red'
        else:
            color = 'blue'
        matplotlib.pyplot.scatter(x, y, color=color)

matplotlib.pyplot.xlabel(first_feature)
matplotlib.pyplot.ylabel(second_feature)
matplotlib.pyplot.show()
```

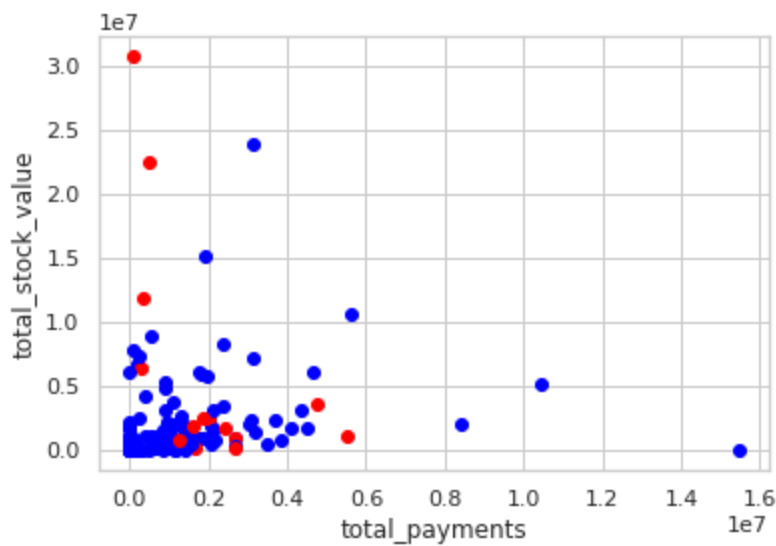
In [14]:

```
# Visualize data to identify outliers
# data_dict.T.to_dict
# data_dict.pop('LAY KENNETH L',0)
showData(data_dict, 'total_payments', 'total_stock_value')
showData(data_dict, 'from_poi_to_this_person', 'from_this_person_to_poi')
showData(data_dict, 'salary', 'bonus')
showData(data_dict, 'restricted_stock', 'exercised_stock_options')
showData(data_dict, 'long_term_incentive', 'deferred_income')
```





In [15]: `print(PlotOutlier(data_dict, 'total_payments', 'total_stock_value'))`



None

In [16]: 

```
### 1.1.0 Explore csv file
def make_csv(data_dict):
    """ generates a csv file from a data set"""
    fieldnames = ['name'] + data_dict.itervalues().next().keys()
    with open('data.csv', 'w') as csvfile:
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
```



```

        writer.writerow()
    for record in data_dict:
        person = data_dict[record]
        person['name'] = record
        assert set(person.keys()) == set(fieldnames)
        writer.writerow(person)

### 1.1.1 Dataset Exploration
print('# Exploratory Data Analysis #')
data_dict.keys()
print('Total number of data points: %d' % len(data_dict.keys()))
num_poi = 0
for name in data_dict.keys():
    if data_dict[name]['poi'] == True:
        num_poi += 1
print('Number of Persons of Interest: %d' % num_poi)
print('Number of people without Person of Interest label: %d' % (len(data_dict.keys()) - num_poi))

### 1.1.2 Feature Exploration
all_features = data_dict['ALLEN PHILLIP K'].keys()
print('Each person has %d features available' % len(all_features))
### Evaluate dataset for completeness
missing_values = {}
for feature in all_features:
    missing_values[feature] = 0
for person in data_dict.keys():
    records = 0
    for feature in all_features:
        if data_dict[person][feature] == 'NaN':
            missing_values[feature] += 1
        else:
            records += 1

### Print results of completeness analysis
print('Number of Missing Values for Each Feature:')
for feature in all_features:
    print("%s: %d" % (feature, missing_values[feature]))

```

```

# Exploratory Data Analysis #
Total number of data points: 140
Number of Persons of Interest: 16
Number of people without Person of Interest label: 124
Each person has 21 features available
Number of Missing Values for Each Feature:
salary: 49
to_messages: 57
deferral_payments: 104
total_payments: 20
loan_advances: 139
bonus: 62
email_address: 32
restricted_stock_deferred: 123
deferred_income: 94
total_stock_value: 18
expenses: 49
from_poi_to_this_person: 57
exercised_stock_options: 42
from_messages: 57
other: 52
from_this_person_to_poi: 57
poi: 0
long_term_incentive: 78
shared_receipt_with_poi: 57

```

```
restricted_stock: 34
director_fees: 124
```

## Data Frame

```
In [17]: df=pd.DataFrame.from_dict(data_dict, orient = 'index')
# data=df.T
df.replace('nan',np.nan)
df.replace(to_replace='NaN', value=np.nan, inplace=True)
df.head()
```

```
Out[17]:
```

	salary	to_messages	deferral_payments	total_payments	loan_advances	bonus	email_add
<b>METTS MARK</b>	365788.0	807.0	NaN	1061827.0	NaN	600000.0	mark.metts@enron.
<b>BAXTER JOHN C</b>	267102.0	NaN	1295738.0	5634343.0	NaN	1200000.0	I
<b>ELLIOTT STEVEN</b>	170941.0	NaN	NaN	211725.0	NaN	350000.0	steven.elliott@enron.
<b>CORDES WILLIAM R</b>	NaN	764.0	NaN	NaN	NaN	NaN	bill.cordes@enron.
<b>HANNON KEVIN P</b>	243293.0	1045.0	NaN	288682.0	NaN	1500000.0	kevin.hannon@enron.

5 rows × 21 columns

```
In [18]: df.dtypes
```

```
Out[18]: salary                float64
to_messages                float64
deferral_payments          float64
total_payments             float64
loan_advances              float64
bonus                     float64
email_address              object
restricted_stock_deferred   float64
deferred_income            float64
total_stock_value          float64
expenses                   float64
from_poi_to_this_person    float64
exercised_stock_options    float64
from_messages              float64
other                      float64
from_this_person_to_poi    float64
poi                        bool
long_term_incentive        float64
shared_receipt_with_poi    float64
restricted_stock           float64
director_fees              float64
dtype: object
```

there are many null data in our dataset. In order to select the most appropriate features to explore, we will look for those that are present at least in 70% of the dataset. Considering there are 21 features (from which 70% is approximate to 15 features), we will first observe which instances have more than 15 not null values and choose the most complete features from this selection.

```
In [83]: notNullDataset = df.dropna(thresh=15)
```

```
notNullDataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 54 entries, HANNON KEVIN P to GLISAN JR BEN F
Data columns (total 21 columns):
 #   Column                                Non-Null Count  Dtype
---  -
 0   salary                               54 non-null     float64
 1   to_messages                          54 non-null     float64
 2   deferral_payments                   13 non-null     float64
 3   total_payments                      54 non-null     float64
 4   loan_advances                       0 non-null      float64
 5   bonus                               50 non-null     float64
 6   email_address                       54 non-null     object
 7   restricted_stock_deferred            6 non-null      float64
 8   deferred_income                     23 non-null     float64
 9   total_stock_value                   53 non-null     float64
10   expenses                            48 non-null     float64
11   from_poi_to_this_person              54 non-null     float64
12   exercised_stock_options              47 non-null     float64
13   from_messages                        54 non-null     float64
14   other                               54 non-null     float64
15   from_this_person_to_poi              54 non-null     float64
16   poi                                  54 non-null     bool
17   long_term_incentive                  42 non-null     float64
18   shared_receipt_with_poi              54 non-null     float64
19   restricted_stock                     53 non-null     float64
20   director_fees                       0 non-null      float64
dtypes: bool(1), float64(19), object(1)
memory usage: 8.9+ KB
```

In [ ]:

## Correlation Matrix of missing Value

We want to see the correlation between the missing values in order to see if the missing values are related to one another. The result shows the missing value as the white part. The missing data mechanism that we found is MAR and MCAR.

In [19]:

```
import seaborn as sns
dfp=df.drop(['poi'], axis=1)
## Calculating the correlation among features by Pearson method
correlationDataframe = df.corr()

# Drawing a heatmap with the numeric values in each cell
fig1, ax = plt.subplots(figsize=(14,10))
fig1.subplots_adjust(top=.945)
plt.suptitle('Features correlation from the Enron POI dataset', fontsize=14, fontweight='k

cbar_kws = {'orientation':"vertical", 'pad':0.025, 'aspect':70}
sns.heatmap(correlationDataframe, annot=True, fmt='.2f', linewidths=.3, ax=ax, cbar_kws=ck
```

Features correlation from the Enron POI dataset



```
In [20]: df['poi'].head()
```

```
Out[20]: METTS MARK          False
BAXTER JOHN C            False
ELLIOTT STEVEN           False
CORDES WILLIAM R         False
HANNON KEVIN P            True
Name: poi, dtype: bool
```

```
In [22]: !pip install missingno
```

Collecting missingno

Downloading missingno-0.5.0-py3-none-any.whl (8.8 kB)

Requirement already satisfied: seaborn in /shared-lib/python3.7/py/lib/python3.7/site-packages (from missingno) (0.11.2)

Requirement already satisfied: matplotlib in /shared-lib/python3.7/py/lib/python3.7/site-packages (from missingno) (3.5.1)

Requirement already satisfied: scipy in /shared-lib/python3.7/py/lib/python3.7/site-packages (from missingno) (1.7.3)

Requirement already satisfied: numpy in /shared-lib/python3.7/py/lib/python3.7/site-packages (from missingno) (1.19.5)

Requirement already satisfied: pandas>=0.23 in /shared-lib/python3.7/py/lib/python3.7/site-packages (from seaborn->missingno) (1.2.5)

Requirement already satisfied: packaging>=20.0 in /shared-lib/python3.7/py-core/lib/python

n3.7/site-packages (from matplotlib->missingno) (21.3)  
Requirement already satisfied: fonttools>=4.22.0 in /shared-libs/python3.7/py/lib/python3.7/site-packages (from matplotlib->missingno) (4.28.3)  
Requirement already satisfied: kiwisolver>=1.0.1 in /shared-libs/python3.7/py/lib/python3.7/site-packages (from matplotlib->missingno) (1.3.2)  
Requirement already satisfied: cycler>=0.10 in /shared-libs/python3.7/py/lib/python3.7/site-packages (from matplotlib->missingno) (0.11.0)  
Requirement already satisfied: pyparsing>=2.2.1 in /shared-libs/python3.7/py-core/lib/python3.7/site-packages (from matplotlib->missingno) (3.0.6)  
Requirement already satisfied: python-dateutil>=2.7 in /shared-libs/python3.7/py-core/lib/python3.7/site-packages (from matplotlib->missingno) (2.8.2)  
Requirement already satisfied: pillow>=6.2.0 in /shared-libs/python3.7/py/lib/python3.7/site-packages (from matplotlib->missingno) (8.4.0)  
Requirement already satisfied: pytz>=2017.3 in /shared-libs/python3.7/py/lib/python3.7/site-packages (from pandas>=0.23->seaborn->missingno) (2021.3)  
Requirement already satisfied: six>=1.5 in /shared-libs/python3.7/py-core/lib/python3.7/site-packages (from python-dateutil>=2.7->matplotlib->missingno) (1.16.0)  
Installing collected packages: missingno  
Successfully installed missingno-0.5.0  
WARNING: You are using pip version 20.1.1; however, version 21.3.1 is available.  
You should consider upgrading via the '/root/venv/bin/python -m pip install --upgrade pip' command.

In [25]: `df[df['loan_advances']!='NaN']`

Out[25]:

	salary	to_messages	deferral_payments	total_payments	loan_advances	bonus	email_ac
<b>METTS MARK</b>	365788.0	807.0	NaN	1061827.0	NaN	600000.0	mark.metts@enro
<b>BAXTER JOHN C</b>	267102.0	NaN	1295738.0	5634343.0	NaN	1200000.0	
<b>ELLIOTT STEVEN</b>	170941.0	NaN	NaN	211725.0	NaN	350000.0	steven.elliott@enro
<b>CORDES WILLIAM R</b>	NaN	764.0	NaN	NaN	NaN	NaN	bill.cordes@enro
<b>HANNON KEVIN P</b>	243293.0	1045.0	NaN	288682.0	NaN	1500000.0	kevin.hannon@enro
...	...	...	...	...	...	...	
<b>GRAMM WENDY L</b>	NaN	NaN	NaN	119292.0	NaN	NaN	
<b>CAUSEY RICHARD A</b>	415189.0	1892.0	NaN	1868758.0	NaN	1000000.0	richard.causey@enro
<b>TAYLOR MITCHELL S</b>	265214.0	533.0	227449.0	1092663.0	NaN	600000.0	mittchell.taylor@enro
<b>DONAHUE JR JEFFREY M</b>	278601.0	865.0	NaN	875760.0	NaN	800000.0	jeff.donahue@enro
<b>GLISAN JR BEN F</b>	274975.0	873.0	NaN	1272284.0	NaN	600000.0	ben.glisan@enro

140 rows × 21 columns

In [34]:

```
len(df)
```

Out[34]: 140

```
In [37]: x=len(df[df['director_fees']!='NaN'])
y=len(df)
c=len(df[df['loan_advances']!='NaN'])
w=len(df[df['restricted_stock_deferred']!='NaN'])
```

Observation: The length of the dataset without NaN on 'loan\_advances' is 140, 'restricted\_stock\_deferred' is 140 and 'director fees' is 140, which are similar to the true length 140.

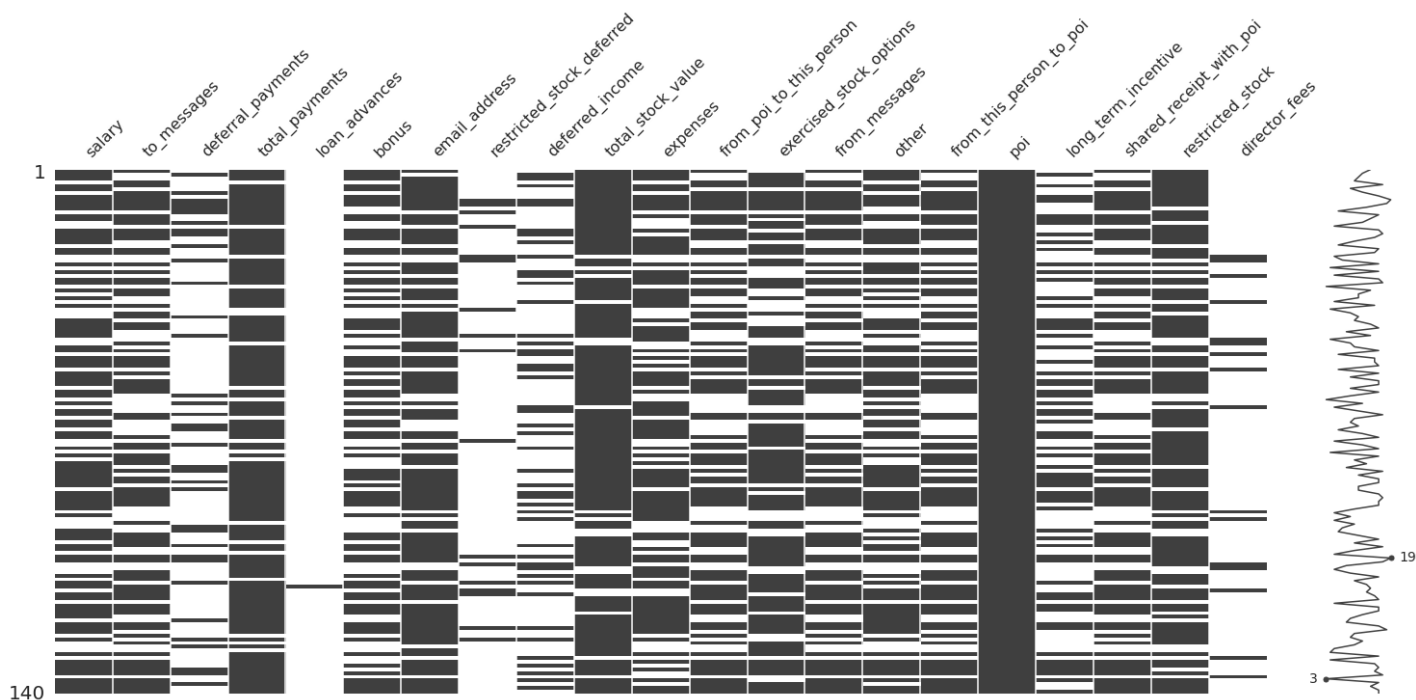
```
In [38]: print(f"we observe that the length of the dataset without NaN on 'loan advance' {c}, 'rest
```

we observe that the length of the dataset without NaN on 'loan advance' 140, 'restricted stock deferred' 140 and 'director fees' 140 is quite similar to the true length 140

```
In [23]: import missingno as msno # # pip install missingno

# Plot correlation heatmap of missingness
msno.matrix(df)
```

Out[23]: <AxesSubplot:>

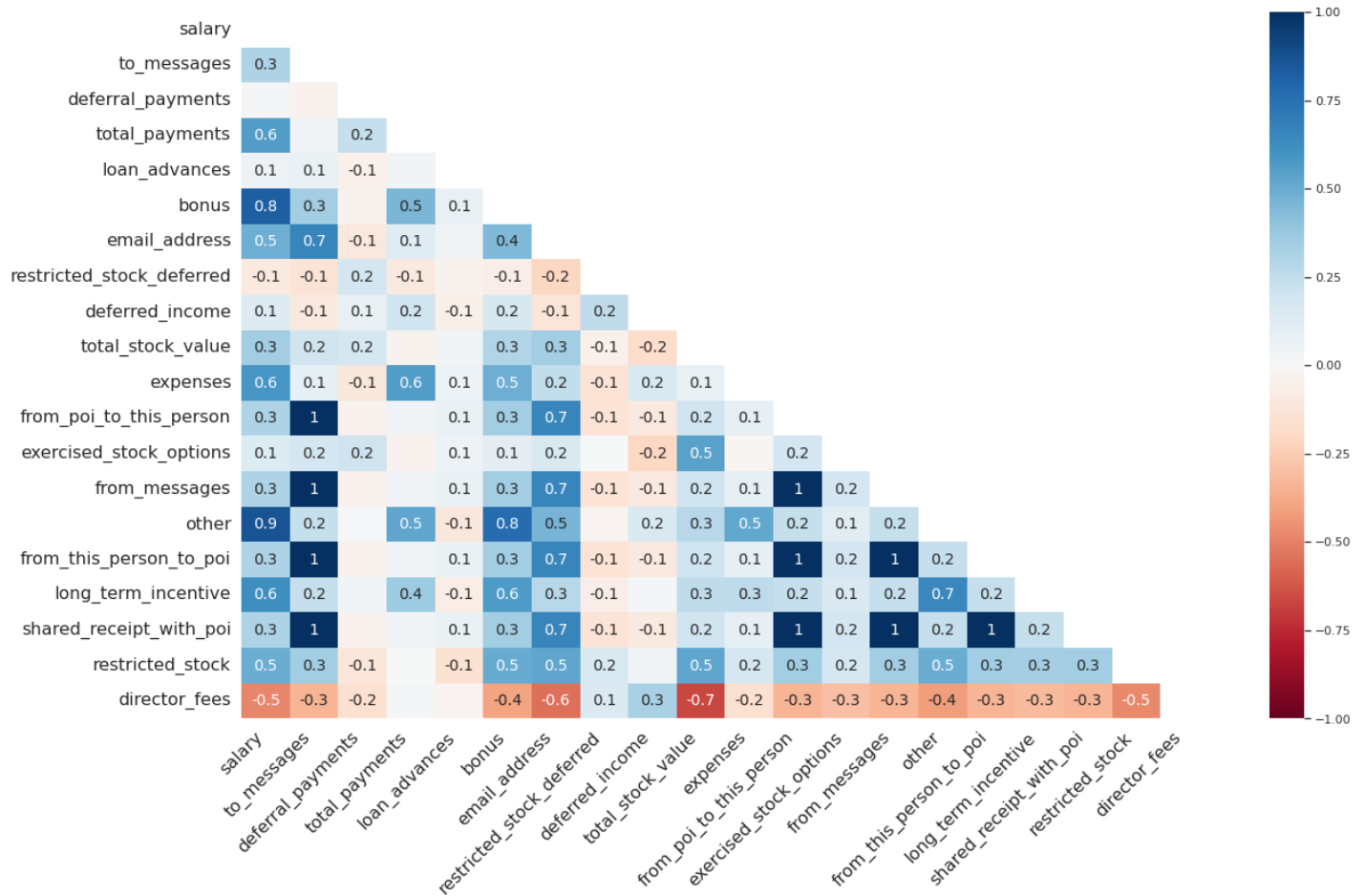


According to the graph above, the data are missing at random (MAR) some columns are almost empty.

Whereas the plot below shows that there is no link between missing values.

```
In [39]: msno.heatmap(df)
```

Out[39]: <AxesSubplot:>



```
In [40]: df.dtypes
```

```
Out[40]: salary                float64
to_messages                float64
deferral_payments          float64
total_payments             float64
loan_advances              float64
bonus                      float64
email_address              object
restricted_stock_deferred   float64
deferred_income            float64
total_stock_value          float64
expenses                   float64
from_poi_to_this_person     float64
exercised_stock_options     float64
from_messages              float64
other                      float64
from_this_person_to_poi     float64
poi                        bool
long_term_incentive         float64
shared_receipt_with_poi     float64
restricted_stock            float64
director_fees              float64
dtype: object
```

## Removing the columns with missing values

Based on the above plots, we are removing the columns 'loan\_advances', 'restricted\_stock\_deferred' and 'director\_fees' as they have a lot of missing values.

```
In [41]: df_min = df[['from_messages',
                    'from_poi_to_this_person',
```





Out[44]:

	from_messages	from_poi_to_this_person	from_this_person_to_poi	shared_receipt_with_poi	to_messages	
METTS MARK	29.000000	38.000000	1.000000	702.000000	807.000000	6.
BAXTER JOHN C	506.993792	32.373317	42.302274	1539.311095	3139.947013	1.
ELLIOTT STEVEN	891.916031	81.025856	61.278353	1015.182259	1836.001065	3.
CORDES WILLIAM R	12.000000	10.000000	0.000000	58.000000	764.000000	1.
HANNON KEVIN P	32.000000	32.000000	21.000000	1035.000000	1045.000000	1.

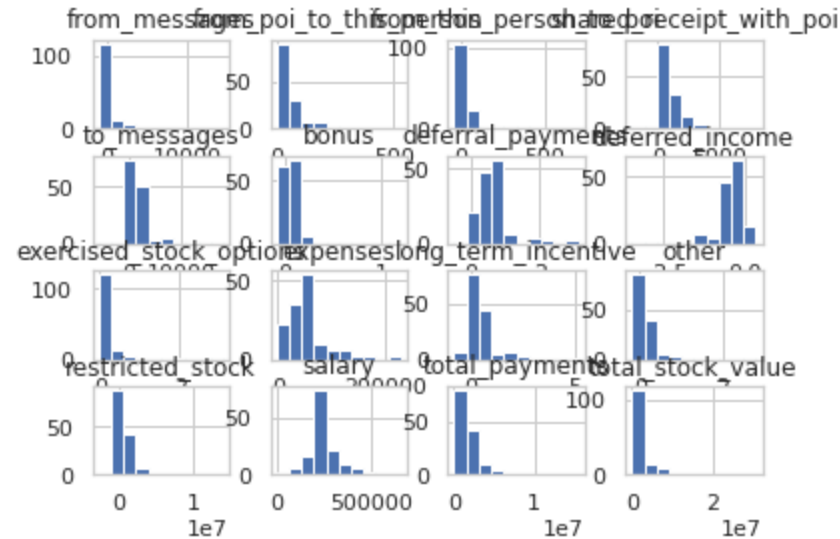
As there are some negative values in the new dataset, we plot the distribution of the new variables using histogram to see the distribution of each variable.

In [45]:

```
df_new.hist()
```

Out[45]:

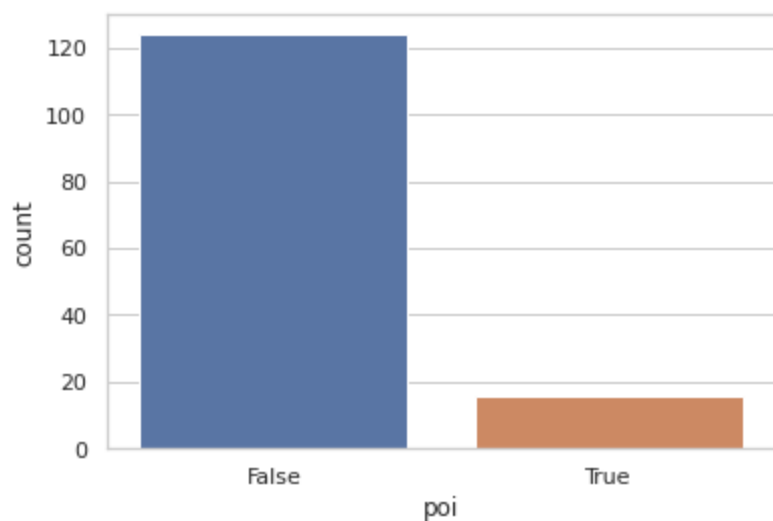
```
array([[<AxesSubplot:title={'center':'from_messages'}>,\n      <AxesSubplot:title={'center':'from_poi_to_this_person'}>,\n      <AxesSubplot:title={'center':'from_this_person_to_poi'}>,\n      <AxesSubplot:title={'center':'shared_receipt_with_poi'}>],\n      [<AxesSubplot:title={'center':'to_messages'}>,\n      <AxesSubplot:title={'center':'bonus'}>,\n      <AxesSubplot:title={'center':'deferral_payments'}>,\n      <AxesSubplot:title={'center':'deferred_income'}>],\n      [<AxesSubplot:title={'center':'exercised_stock_options'}>,\n      <AxesSubplot:title={'center':'expenses'}>,\n      <AxesSubplot:title={'center':'long_term_incentive'}>,\n      <AxesSubplot:title={'center':'other'}>],\n      [<AxesSubplot:title={'center':'restricted_stock'}>,\n      <AxesSubplot:title={'center':'salary'}>,\n      <AxesSubplot:title={'center':'total_payments'}>,\n      <AxesSubplot:title={'center':'total_stock_value'}>]], dtype=object)
```



As we can see, the data are unbalanced and it can be a really big issue.

In [49]:

```
sns.countplot(x='poi', data=df)
plt.show()
```



We proceed with concatenation to add categorical variable poi to the filled dataset.

```
In [50]: print(len(pd.concat([df_new, df.poi],axis=1)))
df_good = pd.concat([df_new, df.poi],axis=1)
df_good.head(10)
```

140

```
Out[50]:
```

	from_messages	from_poi_to_this_person	from_this_person_to_poi	shared_receipt_with_poi	to_messages
<b>METTS MARK</b>	29.000000	38.000000	1.000000	702.000000	807.000000
<b>BAXTER JOHN C</b>	506.993792	32.373317	42.302274	1539.311095	3139.947013
<b>ELLIOTT STEVEN</b>	891.916031	81.025856	61.278353	1015.182259	1836.001065
<b>CORDES WILLIAM R</b>	12.000000	10.000000	0.000000	58.000000	764.000000
<b>HANNON KEVIN P</b>	32.000000	32.000000	21.000000	1035.000000	1045.000000
<b>MORDAUNT KRISTINA M</b>	470.520116	36.648284	4.254155	694.429405	1312.416655
<b>MEYER ROCKFORD G</b>	28.000000	0.000000	0.000000	22.000000	232.000000
<b>MCMAHON JEFFREY</b>	48.000000	58.000000	26.000000	2228.000000	2355.000000
<b>HAEDICKE MARK E</b>	1941.000000	180.000000	61.000000	1847.000000	4009.000000
<b>PIPER GREGORY F</b>	222.000000	61.000000	48.000000	742.000000	1238.000000

```
In [52]: with open("final_project_dataset.pkl", "rb") as data_file:
data_dict = pickle.load(data_file)

my_dataset = data_dict

## 3.2 add new features to dataset
def compute_fraction(poi_messages, all_messages):
```

```

        """ return fraction of messages from/to that person to/from POI"""
        if poi_messages == 'NaN' or all_messages == 'NaN':
            return 0.
        fraction = poi_messages / all_messages
        return fraction

for name in my_dataset:
    data_point = my_dataset[name]
    from_poi_to_this_person = data_point["from_poi_to_this_person"]
    to_messages = data_point["to_messages"]
    fraction_from_poi = compute_fraction(from_poi_to_this_person, to_messages)
    data_point["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 = compute_fraction(from_this_person_to_poi, from_messages)
    data_point["fraction_to_poi"] = fraction_to_poi

# 3.3 create new copies of feature list for grading
my_feature_list = features_list + ['to_messages', 'from_poi_to_this_person', 'from_messages']

# 3.4 get K-best features
num_features = 10

# 3.5 function using SelectKBest
def get_k_best(data_dict, features_list, k):
    """ runs scikit-learn's SelectKBest feature selection
        returns dict where keys=features, values=scores
    """
    data = featureFormat(data_dict, features_list)
    labels, features = targetFeatureSplit(data)

    k_best = SelectKBest(k=k)
    k_best.fit(features, labels)
    scores = k_best.scores_
    print(scores)
    unsorted_pairs = zip(features_list[1:], scores)
    sorted_pairs = list(reversed(sorted(unsorted_pairs, key=lambda x: x[1])))
    k_best_features = dict(sorted_pairs[:k])
    print ("{0} best features: {1}\n".format(k, k_best_features.keys()), scores))
    return k_best_features

best_features = get_k_best(my_dataset, my_feature_list, num_features)

my_feature_list = [target_label] + list(set(best_features.keys()))

# 3.6 print features
print ("{0} selected features: {1}\n".format(len(my_feature_list) - 1, my_feature_list[1:]

# 3.7 extract the features specified in features_list
data = featureFormat(my_dataset, my_feature_list, sort_keys = True)
# split into labels and features
labels, features = targetFeatureSplit(data)

# 3.8 scale features via min-max
from sklearn import preprocessing
scaler = preprocessing.MinMaxScaler()
features = scaler.fit_transform(features)

from sklearn.naive_bayes import GaussianNB
g_clf = GaussianNB()

###4.2 Logistic Regression Classifier

```



[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```

/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1
308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predi
cted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1
308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predi
cted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1
308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predi
cted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1
308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predi
cted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1
308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predi
cted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1
308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predi
cted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
...../shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.
py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no p
redicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1
308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predi
cted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1
308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predi
cted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1
308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predi
cted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1
308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predi
cted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))

```

[illegible]





[illegible]



[illegible]

[illegible]

```

no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
...../shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
...../shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
...../shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
.....done.

precision: 0.22624977668611235
recall:    0.352929834054834
RandomForestClassifier(max_depth=5, max_features='sqrt', n_estimators=10,
                        random_state=42)
/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
Processing../shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classif

```

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]



[illegible]

```

308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
../shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
....../shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
...../shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
...../shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
...../shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
../shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
../shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
../shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
../shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
../shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
done.

```

```

precision: 0.32828769841269845
recall:    0.2859575396825397
Out[52]: (0.32828769841269845, 0.2859575396825397)

```

The results of precision and recall show less than 0.48, which means that dealing with the imbalanced data is not relevant.

precision: 0.32828769841269845 recall: 0.2859575396825397

## Treating imbalanced data

The data are imbalanced, which means that the people with POI is smaller than the people without POI. Using SMOTE in imblearn, we use knn method to create new value of POI person. The result shows the same number of non POI and POI, which means it becomes balanced.

```
In [56]: !pip install imblearn
```

```

Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Collecting imbalanced-learn
  Downloading imbalanced_learn-0.8.1-py3-none-any.whl (189 kB)

```

Requirement already satisfied: numpy>=1.13.3 in /shared-lib/python3.7/py/lib/python3.7/site-packages (from imbalanced-learn->imblearn) (1.19.5)  
Requirement already satisfied: joblib>=0.11 in /shared-lib/python3.7/py/lib/python3.7/site-packages (from imbalanced-learn->imblearn) (1.1.0)  
Requirement already satisfied: scipy>=0.19.1 in /shared-lib/python3.7/py/lib/python3.7/site-packages (from imbalanced-learn->imblearn) (1.7.3)  
Requirement already satisfied: scikit-learn>=0.24 in /shared-lib/python3.7/py/lib/python3.7/site-packages (from imbalanced-learn->imblearn) (1.0.1)  
Requirement already satisfied: threadpoolctl>=2.0.0 in /shared-lib/python3.7/py/lib/python3.7/site-packages (from scikit-learn>=0.24->imbalanced-learn->imblearn) (3.0.0)  
Installing collected packages: imbalanced-learn, imblearn  
Successfully installed imbalanced-learn-0.8.1 imblearn-0.0  
WARNING: You are using pip version 20.1.1; however, version 21.3.1 is available.  
You should consider upgrading via the '/root/venv/bin/python -m pip install --upgrade pip' command.

```
In [57]: X = df_good.loc[:, df_good.columns != 'poi']
y = df_good.loc[:, df_good.columns == 'poi']
from imblearn.over_sampling import SMOTE
os = SMOTE(random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
columns = X_train.columns
os_data_X, os_data_y = os.fit_resample(X_train, y_train)
os_data_X = pd.DataFrame(data=os_data_X, columns=columns)
os_data_y = pd.DataFrame(data=os_data_y, columns=['poi'])
# we can Check the numbers of our data
print("length of the oversampled ", len(os_data_X))
print("Number non poi of the oversampled ", len(os_data_y[os_data_y['poi']==0]))
print("Number poi of the oversampled ", len(os_data_y[os_data_y['poi']==1]))
print("Proportion of non poi of the oversampled ", len(os_data_y[os_data_y['poi']==0])/len(os_data_X))
print("Proportion of poi of the oversampled ", len(os_data_y[os_data_y['poi']==1])/len(os_data_X))
```

length of the oversampled 170  
Number non poi of the oversampled 85  
Number poi of the oversampled 85  
Proportion of non poi of the oversampled 0.5  
Proportion of poi of the oversampled 0.5

Using os data, we can see the statistical description of each variable. The plot shows that the distribution is now balanced.

```
In [58]: os_data_X.describe()
```

```
Out[58]:
```

	from_messages	from_poi_to_this_person	from_this_person_to_poi	shared_receipt_with_poi	to_messages	
count	170.000000	170.000000	170.000000	170.000000	170.000000	170.000000
mean	514.112374	66.544017	38.310794	1235.556688	1953.361735	8.900000
std	1376.696442	66.799277	79.291275	1115.140054	2099.387532	1.310000
min	-904.916759	0.000000	-44.693523	-1478.340681	-3855.469520	-8.220000
25%	34.314445	32.275785	8.019249	567.627194	908.875115	4.000000
50%	203.702448	49.930550	15.959096	1048.303930	1603.816164	7.500000
75%	518.626694	77.457110	37.480373	1599.465239	2487.063574	1.120000
max	14368.000000	528.000000	705.811251	8086.412536	16311.246898	1.190000

```
In [59]: sns.countplot(x='poi', data=os_data_y)
plt.show()
```



We now have the new length of the variable.

```
In [60]: # len(os_data_y)
```

## Turning data into dictionary

We turn the data into dictionary to match picklefind dictionary from the function created.

```
In [61]: data_dict_new = pd.concat([os_data_X, os_data_y],axis=1)
data_dict_new.head(10)
```

```
Out[61]:
```

	from_messages	from_poi_to_this_person	from_this_person_to_poi	shared_receipt_with_poi	to_messages	bonus
0	206.131221	46.098488	15.190182	449.710520	919.381935	2.000000e-
1	63.000000	305.000000	14.000000	1902.000000	2572.000000	6.000000e-
2	125.000000	0.000000	0.000000	23.000000	1088.000000	5.000000e-
3	1073.000000	44.000000	15.000000	1074.000000	2350.000000	9.498732e-
4	510.466039	47.026025	20.534457	454.308875	1024.783145	8.498489e-
5	1144.980111	71.626810	108.617769	2500.849683	4583.291875	1.300000e-
6	1053.000000	156.000000	71.000000	2333.000000	3523.000000	9.000000e-
7	38.000000	37.000000	13.000000	2565.000000	2647.000000	7.000000e-
8	74.000000	115.000000	4.000000	552.000000	714.000000	6.000000e-
9	25.000000	39.000000	18.000000	583.000000	613.000000	1.128370e-

```
In [62]: features_list = ['poi', 'from_messages', 'from_poi_to_this_person', 'from_this_person_to_poi',
'shared_receipt_with_poi', 'to_messages', 'bonus', 'deferral_payments',
'deferred_income', 'exercised_stock_options',
'expenses', 'long_term_incentive', 'other', 'restricted_stock',
'salary', 'total_payments', 'total_stock_value']
```

## Features engineering

Feature engineering refers to the process of using domain knowledge to select and transform the most relevant variables from raw data when creating a predictive model using machine learning or statistical modeling.

```
In [63]: os_data_X['all_messages'] = os_data_X['from_messages']+os_data_X['to_messages']

os_data_X['incentive'] = os_data_X["restricted_stock"]/os_data_X["long_term_incentive"]
os_data_X['bonus_salary'] = os_data_X["bonus"]/os_data_X["salary"]
os_data_X['expenses_salary'] = os_data_X["expenses"]/os_data_X["salary"]
os_data_X['shared_ratio'] = os_data_X["shared_receipt_with_poi"]/os_data_X["all_messages"]
os_data_X['all_poi_ratio'] = ((os_data_X["shared_receipt_with_poi"]+
                             os_data_X["from_poi_to_this_person"]+
                             os_data_X["from_this_person_to_poi"])/
                             os_data_X["all_messages"])
os_data_X['fraction_from_poi'] = os_data_X["from_poi_to_this_person"]/os_data_X["to_messages"]
os_data_X['fraction_to_poi'] = os_data_X["fraction_from_poi"]/os_data_X["from_messages"]
```

The new dictionary we obtain:

```
In [64]: data_dict_new = pd.concat([os_data_X, os_data_y],axis=1)
data_dict_new.head(10)
```

```
Out[64]:
```

	from_messages	from_poi_to_this_person	from_this_person_to_poi	shared_receipt_with_poi	to_messages	boi
0	206.131221	46.098488	15.190182	449.710520	919.381935	2.000000e-
1	63.000000	305.000000	14.000000	1902.000000	2572.000000	6.000000e-
2	125.000000	0.000000	0.000000	23.000000	1088.000000	5.000000e-
3	1073.000000	44.000000	15.000000	1074.000000	2350.000000	9.498732e-
4	510.466039	47.026025	20.534457	454.308875	1024.783145	8.498489e-
5	1144.980111	71.626810	108.617769	2500.849683	4583.291875	1.300000e-
6	1053.000000	156.000000	71.000000	2333.000000	3523.000000	9.000000e-
7	38.000000	37.000000	13.000000	2565.000000	2647.000000	7.000000e-
8	74.000000	115.000000	4.000000	552.000000	714.000000	6.000000e-
9	25.000000	39.000000	18.000000	583.000000	613.000000	1.128370e-

10 rows × 25 columns

Selecting features using selectKbest:

```
In [65]: my_dataset = data_dict_new.to_dict('index')
# 3.3 create new copies of feature list for grading
my_feature_list = features_list + ['shared_ratio','all_poi_ratio',
                                   "incentive","bonus_salary","expenses_salary",
                                   'to_messages', 'from_poi_to_this_person',
                                   'from_messages', 'from_this_person_to_poi','shared_recei
                                   'fraction_to_poi']
# my_feature_list = features_list + ["shared_ratio",'to_messages', 'from_poi_to_this_person

# my_feature_list =

# 3.4 get K-best features
num_features = 7

# 3.5 function using SelectKBest
def get_k_best(data_dict, features_list, k):
    """ runs scikit-learn's SelectKBest feature selection
    returns dict where keys=features, values=scores
```

```

"""
data = featureFormat(data_dict, features_list)
labels, features = targetFeatureSplit(data)

k_best = SelectKBest(k=k)
k_best.fit(features, labels)
scores = k_best.scores_
print(scores)
unsorted_pairs = zip(features_list[1:], scores)
sorted_pairs = list(reversed(sorted(unsorted_pairs, key=lambda x: x[1])))
k_best_features = dict(sorted_pairs[:k])
print ("{0} best features: {1}\n".format(k, k_best_features.keys()), scores))
return k_best_features

best_features = get_k_best(my_dataset, my_feature_list, num_features)

# test=["fraction_to_poi", 'shared_receipt_with_poi', "fraction_from_poi"]#, "salary", 'exerci

my_feature_list = [target_label] +list(set(best_features.keys()))#ltest#list(set(best_fea

# 3.6 print features
# print ("{0} selected features: {1}\n".format(len(my_feature_list) - 1, my_feature_list[1:]
print ("{0} selected features: {1}\n".format(len(my_feature_list), best_features))

# 3.7 extract the features specified in features_list
data = featureFormat(my_dataset, my_feature_list, sort_keys = True)
# split into labels and features
labels, features = targetFeatureSplit(data)

# 3.8 scale features via min-max
from sklearn import preprocessing
scaler = preprocessing.MinMaxScaler()
features = scaler.fit_transform(features)

[ 4.87256554  3.7315301  0.22954509 13.85868103  0.78206845  2.09443291
  5.1399877  19.33211085  9.4985736  6.01394411  2.57713169  9.83340186
 10.03157816 16.09375608  0.05208132 13.81660888  1.01971251  0.76880543
 0.40560114  0.8744787  0.2758199  0.78206845  3.7315301  4.87256554
 0.22954509 13.85868103  0.54077939]
7 best features: dict_keys(['deferred_income', 'salary', 'shared_receipt_with_poi', 'total
_stock_value', 'restricted_stock', 'other'])

7 selected features: {'deferred_income': 19.332110846281193, 'salary': 16.09375607836072,
'shared_receipt_with_poi': 13.858681034011376, 'total_stock_value': 13.816608880768344, 'r
estricted_stock': 10.031578161535952, 'other': 9.833401861698125}

```

## Extracting features and labels from dataset for local testing

In [66]:

```

data = featureFormat(my_dataset, my_feature_list, sort_keys = True)
# split into labels and features
labels, features = targetFeatureSplit(data)

```

## Scaling the features by MinMaxScaler from Sklearn preprocessing module

In [67]:

```

# 3.8 scale features via min-max
from sklearn import preprocessing
scaler = preprocessing.MinMaxScaler()
features = scaler.fit_transform(features)

```

## TASK 4: Try a variety of classifiers

We will try different classifiers to see which of the classifiers' prediction is the best. Classifiers are named clf for easy export.

## 4.1 Gaussian Naive Bayes Classifier

```
In [68]: from sklearn.naive_bayes import GaussianNB
g_clf = GaussianNB()
```

## 4.2 Logistic Regression Classifier

```
In [69]: from sklearn.linear_model import LogisticRegression

l_clf = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ('classifier', LogisticRegression(C=1e-08, class_weight=None, dual=False, fit_intercept=True,
max_iter=100, multi_class='ovr', penalty='l2', random_state=42, solver='liblinear', tol=0.0001)),

l_clf = Pipeline(steps=[
    ('scaler', StandardScaler()),
    ('classifier', LogisticRegression(C=100, class_weight="balanced", dual=False, fit_intercept=True,
max_iter=100, multi_class='ovr', penalty='l1', random_state=42, solver='liblinear', tol=0.0001))])
```

## 4.3 K-means Clustering

```
In [70]: from sklearn.cluster import KMeans
k_clf = KMeans(n_clusters=2, tol=0.001)
```

## 4.4 Support Vector Machine Classifier

```
In [71]: from sklearn.svm import SVC
s_clf = SVC(kernel='rbf', C=1000, gamma = 0.0001, random_state = 42, class_weight = 'balanced')
```

## 4.5 Random Forest

```
In [72]: from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier(max_depth = 5, max_features = 'sqrt', n_estimators = 10, random_state = 42)
```

## 4.6 Gradient Boosting Classifier

```
In [73]: from sklearn.ensemble import GradientBoostingClassifier
gb_clf = GradientBoostingClassifier(loss='deviance', learning_rate=0.1, n_estimators=100, random_state=42)
```

## Evaluate Function

```
In [74]: def evaluate_clf(clf, features, labels, num_iters=1000, test_size=0.3):
    print (clf)
    accuracy = []
    precision = []
    recall = []
    first = True
    for trial in range(num_iters):
        features_train, features_test, labels_train, labels_test = \
            train_test_split(features, labels, test_size=test_size)
        clf.fit(features_train, labels_train)
        predictions = clf.predict(features_test)
```

```

accuracy.append(accuracy_score(labels_test, predictions))
precision.append(precision_score(labels_test, predictions))
recall.append(recall_score(labels_test, predictions))
if trial % 10 == 0:
    if first:
        sys.stdout.write('\nProcessing')
        sys.stdout.write('.')
        sys.stdout.flush()
        first = False

print ("done.\n")
print ("precision: {}".format(mean(precision)))
print ("recall: {}".format(mean(recall)))
return mean(precision), mean(recall)

### 4.8 Evaluate all functions
evaluate_clf(g_clf, features, labels)

```

GaussianNB()

Processing

g.....  
done.

```

precision: 0.8083884391687881
recall:    0.5677869208596437
(0.8083884391687881, 0.5677869208596437)

```

Out[74]:

In [75]:

```

evaluate_clf(g_clf, features, labels)
evaluate_clf(l_clf, features, labels)
evaluate_clf(k_clf, features, labels)
evaluate_clf(s_clf, features, labels)
evaluate_clf(rf_clf, features, labels)
evaluate_clf(gb_clf, features, labels)

```

GaussianNB()

Processing

g.....  
done.

```

precision: 0.8112850424758677
recall:    0.5639472168473169
Pipeline(steps=[('scaler', StandardScaler()),
                 ('classifier',
                  LogisticRegression(C=100, class_weight='balanced',
                                     multi_class='ovr', penalty='l1',
                                     random_state=42, solver='liblinear'))])

```

Processing

g.....  
done.

```

precision: 0.7783732590112813
recall:    0.7621997083341716
KMeans(n_clusters=2, tol=0.001)

```

Processing...../shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/metrics/\_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predicted samples. Use `zero\_division` parameter to control this behavior.

```

    _warn_prf(average, modifier, msg_start, len(result))
.....done.

```



```
precision: 0.7460242911180913
recall:    0.38098787621860014
SVC(C=1000, class_weight='balanced', gamma=0.0001, random_state=42)
```

Processing

```
g.....
done.
```

```
precision: 0.7721769948256569
recall:    0.24134374665738842
RandomForestClassifier(max_depth=5, max_features='sqrt', n_estimators=10,
                        random_state=42)
```

Processing

```
g.....
done.
```

```
precision: 0.8484313218670652
recall:    0.8925135637429819
GradientBoostingClassifier(random_state=42)
```

Processing

```
g.....
done.
```

```
precision: 0.8413264733671205
recall:    0.9218447448235794
(0.8413264733671205, 0.9218447448235794)
```

Out[75]:

As we can see we got really better results

we can still improve our model by selecting the best test set, in order to avoid overfit and data leakage



In [76]:

```
features_list = ['poi', 'from_messages', 'from_poi_to_this_person', 'from_this_person_to_poi',
                 'shared_receipt_with_poi', 'to_messages', 'bonus', 'deferral_payments',
                 'deferred_income', 'exercised_stock_options',
                 'expenses', 'long_term_incentive', 'other', 'restricted_stock',
                 'salary', 'total_payments', 'total_stock_value']
```

A model rely on several parameters and in order to select the best, we will use a GridSearch



In [78]:

```
best_features = get_k_best(my_dataset, features_list, num_features)

my_feature_list = [target_label] + list(set(best_features.keys()))

# 3.6 print features
print ("{0} selected features: {1}\n".format(len(my_feature_list) - 1, my_feature_list[1:]))

# 3.7 extract the features specified in features_list
data = featureFormat(my_dataset, my_feature_list, sort_keys = True)
# split into labels and features
labels, features = targetFeatureSplit(data)

# 3.8 scale features via min-max
from sklearn import preprocessing
scaler = preprocessing.MinMaxScaler()
features = scaler.fit_transform(features)

from sklearn.model_selection import GridSearchCV
```

```

from sklearn.pipeline import make_pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.decomposition import PCA
pl = make_pipeline(SelectKBest(), PCA(random_state = 42, svd_solver='randomized'), DecisionTreeClassifier())
params = dict(
    selectkbest__k = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    decisiontreeclassifier__criterion = ['gini', 'entropy'],
    decisiontreeclassifier__splitter = ['best', 'random']
)

pca = PCA(n_components='mle')

from time import time

grid = GridSearchCV(pl, param_grid = params, scoring = 'recall')

from sklearn.ensemble import AdaBoostClassifier
clf_AdaBoost = AdaBoostClassifier()

[ 4.87256554  3.7315301  0.22954509 13.85868103  0.78206845  2.09443291
  5.1399877  19.33211085  9.4985736  6.01394411  2.57713169  9.83340186
 10.03157816 16.09375608  0.05208132 13.81660888]
7 best features: dict_keys(['deferred_income', 'salary', 'shared_receipt_with_poi', 'total_stock_value', 'restricted_stock', 'other', 'exercised_stock_options'])

7 selected features: ['shared_receipt_with_poi', 'other', 'total_stock_value', 'salary', 'deferred_income', 'restricted_stock', 'exercised_stock_options']

```

## TASK 5: Tune the classifier

We tune the classifier to achieve better precision and recall.

Because of the small size of the dataset, the script uses stratified shuffle split cross validation to test the classifiers. Therefore, the data is shuffled each time before being split into a specified number of folds (compared to StratifiedKFold, which shuffles only once at the beginning).

In practice, the PCA improves training rate, simplifies the required neural structure to represent the data, and results in systems that better characterize the "intermediate structure" of the data instead of having to account for multiple scales - it is more accurate.

Our guess is that there are analogous reasons that apply to random forests of gradient boosted trees or other similar creatures.

In [79]:

```

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)

pca.fit(features_train)
features_train_pca = pca.transform(features_train)
features_test_pca = pca.transform(features_test)

grid.fit(features_train, labels_train)
clf_DT = grid.best_estimator_

t0 = time()
clf_DT.fit(features_train, labels_train)
print ("Decision Tree - training time:", round(time()-t0, 3), "s")
t1 = time()
predictions_DT = clf_DT.predict(features_test)

```

```

print ("Decision Tree - prediction time:", round(time()-t1, 3), "s")

t0 = time()
clf_AdaBoost.fit(features_train_pca, labels_train)
print( "AdaBoost - training time:", round(time()-t0, 3), "s")
t1 = time()
predictions_AdaBoost = clf_AdaBoost.predict(features_test_pca)
print ("AdaBoost - prediction time:", round(time()-t1, 3), "s")

### Stochastic Gradient Descent
from sklearn import linear_model
clf_SGD = linear_model.SGDClassifier(class_weight = "balanced")

### Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB
clf_NB = GaussianNB()

### Random Forests
from sklearn.ensemble import RandomForestClassifier
clf_RF = RandomForestClassifier()

clf_SGD.fit(features_train_pca, labels_train)
predictions_SGD = clf_SGD.predict(features_test_pca)

clf_NB.fit(features_train_pca, labels_train)
predictions_NB = clf_NB.predict(features_test_pca)

clf_RF.fit(features_train_pca, labels_train)
predictions_RF = clf_RF.predict(features_test_pca)

from sklearn.metrics import precision_score, recall_score
print ("precision score for the Gaussian Naive Bayes Classifier : ", precision_score(labels_test, predictions_NB))
print ("recall score for the Gaussian Naive Bayes Classifier : ", recall_score(labels_test, predictions_NB))

print ("precision score for the Decision tree Classifier : ", precision_score(labels_test, predictions_RF))
print ("recall score for the Decision tree Classifier : ", recall_score(labels_test, predictions_RF))

print ("precision score for the AdaBoost Classifier : ", precision_score(labels_test, predictions_AdaBoost))
print ("recall score for the AdaBoost Classifier : ", recall_score(labels_test, predictions_AdaBoost))

print ("precision score for the Random Forest Classifier : ", precision_score(labels_test, predictions_RF))
print ("recall score for the Random Forest Classifier : ", recall_score(labels_test, predictions_RF))

print ("precision score for the Stochastic Gradient Descent Classifier : ", precision_score(labels_test, predictions_SGD))
print ("recall score for the Stochastic Gradient Descent Classifier : ", recall_score(labels_test, predictions_SGD))

```

```

/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/model_selection/_validation.py:372: FitFailedWarning:
60 fits failed out of a total of 200.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

```

Below are more details about the failures:

-----

20 fits failed with the following error:

Traceback (most recent call last):

```

File "/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/model_selection/_validation.py", line 681, in _fit_and_score

```

```

    estimator.fit(X_train, y_train, **fit_params)

```

```

File "/shared-libs/python3.7/py/lib/python3.7/site-packages/sklearn/pipeline.py", line 390, in fit

```

```

    Xt = self._fit(X, y, **fit_params_steps)

```

```

File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/pipeline.py", line 3
55, in _fit
    **fit_params_steps[name],
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/joblib/memory.py", line 349,
in __call__
    return self.func(*args, **kwargs)
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/pipeline.py", line 8
93, in _fit_transform_one
    res = transformer.fit_transform(X, y, **fit_params)
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/base.py", line 850,
in fit_transform
    return self.fit(X, y, **fit_params).transform(X)
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/feature_selection/_u
nivariate_selection.py", line 407, in fit
    self._check_params(X, y)
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/feature_selection/_u
nivariate_selection.py", line 606, in _check_params
    "Use k='all' to return all features." % (X.shape[1], self.k)
ValueError: k should be >=0, <= n_features = 7; got 8. Use k='all' to return all features.

```

-----  
20 fits failed with the following error:

Traceback (most recent call last):

```

File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/model_selection/_val
idation.py", line 681, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/pipeline.py", line 3
90, in fit
    Xt = self._fit(X, y, **fit_params_steps)
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/pipeline.py", line 3
55, in _fit
    **fit_params_steps[name],
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/joblib/memory.py", line 349,
in __call__
    return self.func(*args, **kwargs)
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/pipeline.py", line 8
93, in _fit_transform_one
    res = transformer.fit_transform(X, y, **fit_params)
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/base.py", line 850,
in fit_transform
    return self.fit(X, y, **fit_params).transform(X)
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/feature_selection/_u
nivariate_selection.py", line 407, in fit
    self._check_params(X, y)
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/feature_selection/_u
nivariate_selection.py", line 606, in _check_params
    "Use k='all' to return all features." % (X.shape[1], self.k)
ValueError: k should be >=0, <= n_features = 7; got 9. Use k='all' to return all features.

```

-----  
20 fits failed with the following error:

Traceback (most recent call last):

```

File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/model_selection/_val
idation.py", line 681, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/pipeline.py", line 3
90, in fit
    Xt = self._fit(X, y, **fit_params_steps)
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/pipeline.py", line 3
55, in _fit
    **fit_params_steps[name],
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/joblib/memory.py", line 349,
in __call__
    return self.func(*args, **kwargs)
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/pipeline.py", line 8
93, in _fit_transform_one

```

```

res = transformer.fit_transform(X, y, **fit_params)
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/base.py", line 850,
in fit_transform
    return self.fit(X, y, **fit_params).transform(X)
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/feature_selection/_u
nivariate_selection.py", line 407, in fit
    self._check_params(X, y)
File "/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/feature_selection/_u
nivariate_selection.py", line 606, in _check_params
    "Use k='all' to return all features." % (X.shape[1], self.k)
ValueError: k should be >=0, <= n_features = 7; got 10. Use k='all' to return all feature
s.

warnings.warn(some_fits_failed_message, FitFailedWarning)
/shared-lib/python3.7/py/lib/python3.7/site-packages/sklearn/model_selection/_search.py:9
72: UserWarning: One or more of the test scores are non-finite: [0.68030303 0.76363636 0.8
4848485 0.86363636 0.83030303 0.7969697
0.88181818      nan      nan      nan 0.6469697 0.72878788
0.74545455 0.78030303 0.7469697 0.89848485 0.83333333      nan
      nan      nan 0.68030303 0.72878788 0.83181818 0.84545455
0.74545455 0.78181818 0.83181818      nan      nan      nan
0.6469697 0.72878788 0.79848485 0.79545455 0.78030303 0.86363636
0.85      nan      nan      nan]
category=UserWarning,
Decision Tree - training time: 0.004 s
Decision Tree - prediction time: 0.002 s
AdaBoost - training time: 0.132 s
AdaBoost - prediction time: 0.028 s
precision score for the Gaussian Naive Bayes Classifier : 0.7647058823529411
recall score for the Gaussian Naive Bayes Classifier : 0.5
precision score for the Decision tree Classifier : 0.6896551724137931
recall score for the Decision tree Classifier : 0.7692307692307693
precision score for the AdaBoost Classifier : 0.84
recall score for the AdaBoost Classifier : 0.8076923076923077
precision score for the Random Forest Classifier : 0.7857142857142857
recall score for the Random Forest Classifier : 0.8461538461538461
precision score for the Stochastic Gradient Descent Classifier : 0.8260869565217391
recall score for the Stochastic Gradient Descent Classifier : 0.7307692307692307

```

In [81]:

```

clf_SVC = SVC(gamma=3, C=2)
clf_SVC.fit(features_train, labels_train)
pickle.dump(my_dataset, open("my_dataset.pkl", "wb"))
pickle.dump(my_feature_list, open("my_feature_list.pkl", "wb"))

all_classifiers_list = [g_clf, l_clf, k_clf, s_clf, s_clf, rf_clf, gb_clf]
import tester
for clf in all_classifiers_list:
    print("Results classifier:")
    pickle.dump(clf, open("my_classifier.pkl", "wb"))
    tester.dump_classifier_and_data(clf, my_dataset, my_feature_list)
    tester.main();

```

Results classifier:

##### len test 170

GaussianNB()

Accuracy: 0.66065	Precision: 0.80652	Recall: 0.42238	F1: 0.55441	F
2: 0.46686				

Total predictions: 17000	True positives: 3589	False positives: 861	Fa
lse negatives: 4908	True negatives: 7642		

Results classifier:

##### len test 170

```

Pipeline(steps=[('scaler', StandardScaler()),
                 ('classifier',
                  LogisticRegression(C=100, class_weight='balanced',

```

```

                                multi_class='ovr', penalty='l1',
                                random_state=42, solver='liblinear'))))
    Accuracy: 0.78424      Precision: 0.78197      Recall: 0.78804 F1: 0.78499      F
2: 0.78682
    Total predictions: 17000      True positives: 6696      False positives: 1867      Fa
lse negatives: 1801      True negatives: 6636

Results classifier:
##### len test 170
KMeans(n_clusters=2, tol=0.001)
    Accuracy: 0.53165      Precision: 0.61095      Recall: 0.17336 F1: 0.27008      F
2: 0.20234
    Total predictions: 17000      True positives: 1473      False positives: 938      Fa
lse negatives: 7024      True negatives: 7565

Results classifier:
##### len test 170
SVC(C=1000, class_weight='balanced', gamma=0.0001, random_state=42)
    Accuracy: 0.47059      Precision: 0.47059      Recall: 0.47358 F1: 0.47208      F
2: 0.47298
    Total predictions: 17000      True positives: 4024      False positives: 4527      Fa
lse negatives: 4473      True negatives: 3976

Results classifier:
##### len test 170
SVC(C=1000, class_weight='balanced', gamma=0.0001, random_state=42)
    Accuracy: 0.47059      Precision: 0.47059      Recall: 0.47358 F1: 0.47208      F
2: 0.47298
    Total predictions: 17000      True positives: 4024      False positives: 4527      Fa
lse negatives: 4473      True negatives: 3976

Results classifier:
##### len test 170
RandomForestClassifier(max_depth=5, max_features='sqrt', n_estimators=10,
                        random_state=42)
    Accuracy: 0.88453      Precision: 0.86004      Recall: 0.91844 F1: 0.88828      F
2: 0.90614
    Total predictions: 17000      True positives: 7804      False positives: 1270      Fa
lse negatives: 693      True negatives: 7233

Results classifier:
##### len test 170
GradientBoostingClassifier(random_state=42)
    Accuracy: 0.89835      Precision: 0.85811      Recall: 0.95445 F1: 0.90372      F
2: 0.93349
    Total predictions: 17000      True positives: 8110      False positives: 1341      Fa
lse negatives: 387      True negatives: 7162

```

In [84]:

```

evaluate_clf(clf_SGD, features, labels)
evaluate_clf(clf_NB, features, labels)
evaluate_clf(clf_RF, features, labels)
evaluate_clf(clf_SVC, features, labels)

```

```
SGDClassifier(class_weight='balanced')
```

```

Processing...../shared-libs/python3.7/py/lib/python3.7/site-packages/sklea
rn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision is ill-defined and b
eing set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this
behavior.

```

```

    _warn_prf(average, modifier, msg_start, len(result))
.....done.

```

```

precision: 0.7438787416972873
recall:    0.7216136549827863

```

```
GaussianNB()
```

```
Processing
```

```
g.....  
done.
```

```
precision: 0.8127679802467901  
recall:    0.46190241562507495  
RandomForestClassifier()
```

```
Processing
```

```
g.....  
done.
```

```
precision: 0.8759374814192283  
recall:    0.922577123089399  
SVC(C=2, gamma=3)
```

```
Processing
```

```
g.....  
done.
```

```
precision: 0.8273230288212268  
recall:    0.8431874342418607  
(0.8273230288212268, 0.8431874342418607)
```

Out[84]:

## TASK 6: Dump classifier, dataset, and features\_list

In [85]:

```
pickle.dump(my_dataset, open("my_dataset.pkl", "wb"))  
pickle.dump(my_feature_list, open("my_feature_list.pkl", "wb"))
```

## Final evaluation using tester.py script

In [86]:

```
import tester  
  
all_classifiers_list = [g_clf, l_clf, k_clf, s_clf, s_clf, rf_clf, gb_clf]  
  
for clf in all_classifiers_list:  
    print("Results classifier:")  
    pickle.dump(clf, open("my_classifier.pkl", "wb"))  
    tester.dump_classifier_and_data(clf, my_dataset, my_feature_list)  
    tester.main();
```

Results classifier:

##### len test 170

GaussianNB()

Accuracy: 0.66065	Precision: 0.80652	Recall: 0.42238	F1: 0.55441	F
2: 0.46686				

Total predictions: 17000	True positives: 3589	False positives: 861	Fa
lse negatives: 4908	True negatives: 7642		

Results classifier:

##### len test 170

```
Pipeline(steps=[('scaler', StandardScaler()),  
                ('classifier',  
                 LogisticRegression(C=100, class_weight='balanced',  
                                   multi_class='ovr', penalty='l1',  
                                   random_state=42, solver='liblinear'))])
```

Accuracy: 0.78424	Precision: 0.78197	Recall: 0.78804	F1: 0.78499	F
2: 0.78682				

Total predictions: 17000	True positives: 6696	False positives: 1867	Fa
lse negatives: 1801	True negatives: 6636		

```

Results classifier:
##### len test 170
KMeans(n_clusters=2, tol=0.001)
    Accuracy: 0.53100      Precision: 0.61781      Recall: 0.16170 F1: 0.25632      F
2: 0.18972
    Total predictions: 17000      True positives: 1374      False positives: 850      Fa
lse negatives: 7123      True negatives: 7653

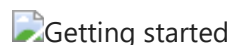
Results classifier:
##### len test 170
SVC(C=1000, class_weight='balanced', gamma=0.0001, random_state=42)
    Accuracy: 0.47059      Precision: 0.47059      Recall: 0.47358 F1: 0.47208      F
2: 0.47298
    Total predictions: 17000      True positives: 4024      False positives: 4527      Fa
lse negatives: 4473      True negatives: 3976

Results classifier:
##### len test 170
SVC(C=1000, class_weight='balanced', gamma=0.0001, random_state=42)
    Accuracy: 0.47059      Precision: 0.47059      Recall: 0.47358 F1: 0.47208      F
2: 0.47298
    Total predictions: 17000      True positives: 4024      False positives: 4527      Fa
lse negatives: 4473      True negatives: 3976

Results classifier:
##### len test 170
RandomForestClassifier(max_depth=5, max_features='sqrt', n_estimators=10,
                        random_state=42)
    Accuracy: 0.88453      Precision: 0.86004      Recall: 0.91844 F1: 0.88828      F
2: 0.90614
    Total predictions: 17000      True positives: 7804      False positives: 1270      Fa
lse negatives: 693      True negatives: 7233

Results classifier:
##### len test 170
GradientBoostingClassifier(random_state=42)
    Accuracy: 0.89835      Precision: 0.85811      Recall: 0.95445 F1: 0.90372      F
2: 0.93349
    Total predictions: 17000      True positives: 8110      False positives: 1341      Fa
lse negatives: 387      True negatives: 7162

```



dump your classifier, dataset and features\_list

so anyone can run/check your results

```

In [87]: clf = gb_clf

pickle.dump(clf, open("my_classifier.pkl", "wb"))
pickle.dump(my_dataset, open("my_dataset.pkl", "wb"))
pickle.dump(my_feature_list, open("my_feature_list.pkl", "wb"))

dump_classifier_and_data(clf, my_dataset, features_list)

```


End conclusion:

Both **Random Forest** and **Gradient Boosting** are superior to the other algorithms used in terms of accuracy, precision and recall.



Because the *recall* represents the number of hits (correctly identified POI's) in relation to the sum of hit and miss (all POI's to be detected), the *recall* is in this context, where the detection of fraud has priority, the most important parameter, especially compared to the in this context secondary *precision*, which describes the ratio of the number of hits relative to all supposedly detected frauds.

Although the Random Forest is superior to Gradient Boosting with about 87% precision, **Gradient Boosting** is therefore suggested as the most suitable algorithm in this context: with about 90% recall it is superior to the Random Forest in this respect.

 Getting started