Binary Classification Model for Credit Card Default

Sundar

Jun, 2019

Dataset Used: Default of Credit Card Clients Data Set

Dataset ML Model: Binary classification with numerical attributes

Dataset Reference: https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients (https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients)

What this data has:

This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005.

Project Aim:

The purpose of this project is to analyze predictions using various machine learning algorithms and to document the steps using a template. Working through machine learning problems from end-to-end requires a structured modeling approach. Working problems through a project template can also encourage us to think about the problem more critically, to challenge our assumptions, and to get proficient at all parts of a modeling project.

For this round of modeling, converting the credit limit and age attributes from ordinal to categorical did not have a noticeable effect on the accuracy of the models.

The project aims to touch on the following areas:

- Document a predictive modeling problem end-to-end.
- Explore data cleaning and transformation options
- Explore non-ensemble and ensemble algorithms for baseline model performance
- Explore algorithm tuning techniques for improving model performance

Any predictive modeling machine learning project genrally can be broken down into about six major tasks:

1. Prepare Problem
2. Summarize Data
3. Prepare Data
4. Model and Evaluate Algorithms
5. Improve Accuracy or Results
6. Finalize Model and Present Results

Section 1 - Prepare Problem

In [1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot
from pandas import read_csv
from pandas import set_option
from pandas import get_dummies
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
# from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.externals.joblib import dump
from sklearn.externals.joblib import load
from datetime import datetime
```

```
C:\Users\sundara.rao.ext\AppData\Local\Continuum\anaconda3\lib\site-packages
\sklearn\ensemble\weight_boosting.py:29: DeprecationWarning: numpy.core.umath
_tests is an internal NumPy module and should not be imported. It will be rem
oved in a future NumPy release.
  from numpy.core.umath_tests import inner1d
```

1.b) Load dataset

In [18]:
```
startTimeScript = datetime.now()

# inputFile = 'default-of-credit-card-clients.csv'
entireDataset = pd.read_csv("C:\\Users\\sundara.rao.ext\\Desktop\\SUNDAR\\R\\S
ocial Prachar Material\\Codes With Examples - Python\\default of credit card c
lients.csv")

# Rename the target variable column to a standard name "targetVar"
entireDataset = entireDataset.rename(columns={'default payment next month': 't
argetVar'})

# Drop the Customer ID field as the label column has no relevance in the predi
ction exercise
entireDataset.drop('ID', axis=1, inplace=True)
```

Section 2 - Summarize Data

To gain a better understanding of the data that we have on-hand, we will leverage a number of descriptive statistics and data visualization techniques. The plan is to use the results to consider new questions, review assumptions, and validate hypotheses that we can investigate later with specialized models.

2.a) Descriptive statistics

In [19]:
```
# Set up a variable for the total number of attribute columns (totAttr)
totCol = len(entireDataset.columns)
totAttr = totCol-1

# Set up the number of row and columns for visualization display. dispRow * di
spCol should be >= totAttr
dispCol = 3
if totAttr % dispCol == 0 :
    dispRow = totAttr // dispCol
else :
    dispRow = (totAttr // dispCol) + 1
```

2.a.i) Peek at the data itself.

In [20]:

```
set_option('display.width', 100)
print(entireDataset.head(20))
```

```
    LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3  PAY_4  PAY
_5    ...      \
0       20000    2          2         1   24      2      2     -1     -1
-2    ...
1      120000    2          2         2   26     -1      2      0      0
0    ...
2       90000    2          2         2   34      0      0      0      0
0    ...
3       50000    2          2         1   37      0      0      0      0
0    ...
4       50000    1          2         1   57     -1      0     -1      0
0    ...
5       50000    1          1         2   37      0      0      0      0
0    ...
6      500000    1          1         2   29      0      0      0      0
0    ...
7      100000    2          2         2   23      0     -1     -1      0
0    ...
8      140000    2          3         1   28      0      0      2      0
0    ...
9       20000    1          3         2   35     -2     -2     -2     -2
-1    ...
10     200000    2          3         2   34      0      0      2      0
0    ...
11     260000    2          1         2   51     -1     -1     -1     -1
-1    ...
12     630000    2          2         2   41     -1      0     -1     -1
-1    ...
13      70000    1          2         2   30      1      2      2      0
0    ...
14     250000    1          1         2   29      0      0      0      0
0    ...
15      50000    2          3         3   23      1      2      0      0
0    ...
16      20000    1          1         2   24      0      0      2      2
2    ...
17     320000    1          1         1   49      0      0      0     -1
-1    ...
18     360000    2          1         1   49      1     -2     -2     -2
-2    ...
19     180000    2          1         2   29      1     -2     -2     -2
-2    ...

    BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2  PAY_AMT3  PAY_AMT4
PAY_AMT5  PAY_AMT6  \
0           0          0          0         0       689         0         0
0         0
1        3272       3455       3261         0      1000      1000      1000
0      2000
2       14331      14948      15549      1518      1500      1000      1000
1000      5000
3       28314      28959      29547      2000      2019      1200      1100
1069      1000
4       20940      19146      19131      2000     36681     10000      9000
689       679
5       19394      19619      20024      2500      1815       657      1000
1000       800
```

```
6      542653     483003     473944     55000     40000     38000     20239
13750     13770
7         221       -159        567       380       601         0       581
1687      1542
8       12211      11793       3719      3329         0       432      1000
1000      1000
9           0      13007      13912         0         0         0     13007
1122         0
10       2513       1828       3731      2306        12        50       300
3738        66
11       8517      22287      13668     21818      9966      8583     22301
0      3640
12       6500       6500       2870      1000      6500      6500      6500
2870         0
13      66782      36137      36894      3200         0      3000      3000
1500         0
14      59696      56875      55512      3000      3000      3000      3000
3000      3000
15      28771      29531      30211         0      1500      1100      1200
1300      1100
16      18338      17905      19104      3200         0      1500         0
1650         0
17      70074       5856     195599     10358     10000     75940     20000
195599     50000
18          0          0          0         0         0         0         0
0         0
19          0          0          0         0         0         0         0
0         0

    targetVar
0           1
1           1
2           0
3           0
4           0
5           0
6           0
7           0
8           0
9           0
10          0
11          0
12          0
13          1
14          0
15          0
16          1
17          0
18          0
19          0

[20 rows x 24 columns]
```

2.a.ii) Dimensions of the dataset.

In [21]:
```
print(entireDataset.shape)
```

```
(30000, 24)
```

2.a.iii) Types of the attributes.

In [22]:
```
print(entireDataset.dtypes)
```

```
LIMIT_BAL    int64
SEX          int64
EDUCATION    int64
MARRIAGE     int64
AGE          int64
PAY_0        int64
PAY_2        int64
PAY_3        int64
PAY_4        int64
PAY_5        int64
PAY_6        int64
BILL_AMT1    int64
BILL_AMT2    int64
BILL_AMT3    int64
BILL_AMT4    int64
BILL_AMT5    int64
BILL_AMT6    int64
PAY_AMT1     int64
PAY_AMT2     int64
PAY_AMT3     int64
PAY_AMT4     int64
PAY_AMT5     int64
PAY_AMT6     int64
targetVar    int64
dtype: object
```

2.a.iv) Statistical summary of all attributes.

In [23]:
```
print(entireDataset.describe().transpose())
```

```
              count           mean            std       min       25%        5
0%        75%  \
LIMIT_BAL  30000.0  167484.322667  129747.661567   10000.0  50000.00  140000.
0  240000.00
SEX        30000.0       1.603733       0.489129       1.0      1.00        2.
0       2.00
EDUCATION  30000.0       1.853133       0.790349       0.0      1.00        2.
0       2.00
MARRIAGE   30000.0       1.551867       0.521970       0.0      1.00        2.
0       2.00
AGE        30000.0      35.485500       9.217904      21.0     28.00       34.
0      41.00
PAY_0      30000.0      -0.016700       1.123802      -2.0     -1.00        0.
0       0.00
PAY_2      30000.0      -0.133767       1.197186      -2.0     -1.00        0.
0       0.00
PAY_3      30000.0      -0.166200       1.196868      -2.0     -1.00        0.
0       0.00
PAY_4      30000.0      -0.220667       1.169139      -2.0     -1.00        0.
0       0.00
PAY_5      30000.0      -0.266200       1.133187      -2.0     -1.00        0.
0       0.00
PAY_6      30000.0      -0.291100       1.149988      -2.0     -1.00        0.
0       0.00
BILL_AMT1  30000.0   51223.330900   73635.860576 -165580.0   3558.75    22381.
5   67091.00
BILL_AMT2  30000.0   49179.075167   71173.768783  -69777.0   2984.75    21200.
0   64006.25
BILL_AMT3  30000.0   47013.154800   69349.387427 -157264.0   2666.25    20088.
5   60164.75
BILL_AMT4  30000.0   43262.948967   64332.856134 -170000.0   2326.75    19052.
0   54506.00
BILL_AMT5  30000.0   40311.400967   60797.155770  -81334.0   1763.00    18104.
5   50190.50
BILL_AMT6  30000.0   38871.760400   59554.107537 -339603.0   1256.00    17071.
0   49198.25
PAY_AMT1   30000.0    5663.580500   16563.280354       0.0   1000.00     2100.
0    5006.00
PAY_AMT2   30000.0    5921.163500   23040.870402       0.0    833.00     2009.
0    5000.00
PAY_AMT3   30000.0    5225.681500   17606.961470       0.0    390.00     1800.
0    4505.00
PAY_AMT4   30000.0    4826.076867   15666.159744       0.0    296.00     1500.
0    4013.25
PAY_AMT5   30000.0    4799.387633   15278.305679       0.0    252.50     1500.
0    4031.50
PAY_AMT6   30000.0    5215.502567   17777.465775       0.0    117.75     1500.
0    4000.00
targetVar  30000.0       0.221200       0.415062       0.0      0.00        0.
0       0.00


                 max
LIMIT_BAL  1000000.0
SEX              2.0
EDUCATION        6.0
MARRIAGE         3.0
AGE             79.0
```

```
PAY_0            8.0
PAY_2            8.0
PAY_3            8.0
PAY_4            8.0
PAY_5            8.0
PAY_6            8.0
BILL_AMT1   964511.0
BILL_AMT2   983931.0
BILL_AMT3  1664089.0
BILL_AMT4   891586.0
BILL_AMT5   927171.0
BILL_AMT6   961664.0
PAY_AMT1    873552.0
PAY_AMT2   1684259.0
PAY_AMT3    896040.0
PAY_AMT4    621000.0
PAY_AMT5    426529.0
PAY_AMT6    528666.0
targetVar        1.0
```

2.a.v) Summarize the levels of the class attribute.

In [24]:
```
print(entireDataset.groupby('targetVar').size())
```

```
targetVar
0    23364
1     6636
dtype: int64
```

2.a.v) Count missing values.

In [25]:
```
print(entireDataset.isnull().sum())
```

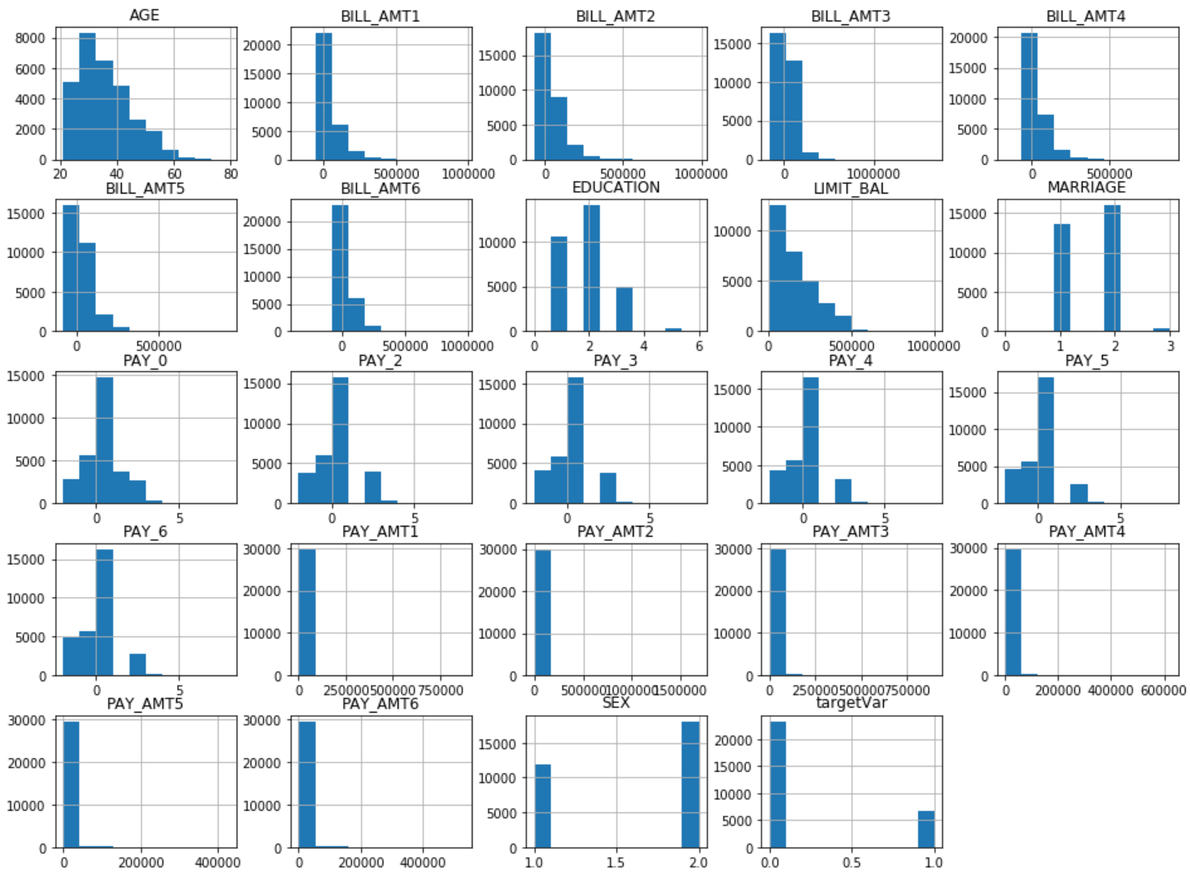
```
LIMIT_BAL    0
SEX          0
EDUCATION    0
MARRIAGE     0
AGE          0
PAY_0        0
PAY_2        0
PAY_3        0
PAY_4        0
PAY_5        0
PAY_6        0
BILL_AMT1    0
BILL_AMT2    0
BILL_AMT3    0
BILL_AMT4    0
BILL_AMT5    0
BILL_AMT6    0
PAY_AMT1     0
PAY_AMT2     0
PAY_AMT3     0
PAY_AMT4     0
PAY_AMT5     0
PAY_AMT6     0
targetVar    0
dtype: int64
```

2.b) Data visualizations

2.b.i) Univariate plots to better understand each attribute

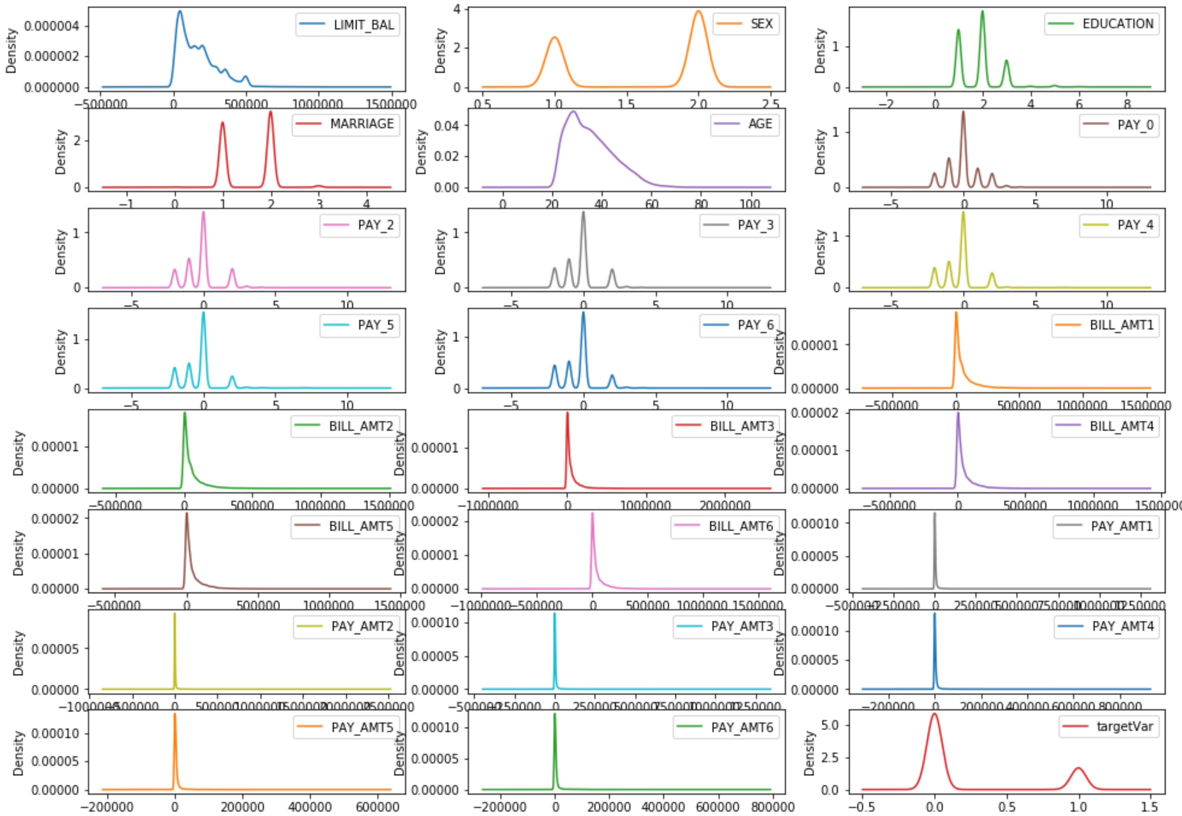
In [26]:
```
# Set figure width to 16 and height to 12 (4:3 aspect ratio)
fig_size = pyplot.rcParams["figure.figsize"]
fig_size[0] = 16
fig_size[1] = 12
pyplot.rcParams["figure.figsize"] = fig_size
```

In [27]:
```
# Histograms for each attribute
entireDataset.hist()
pyplot.show()
```

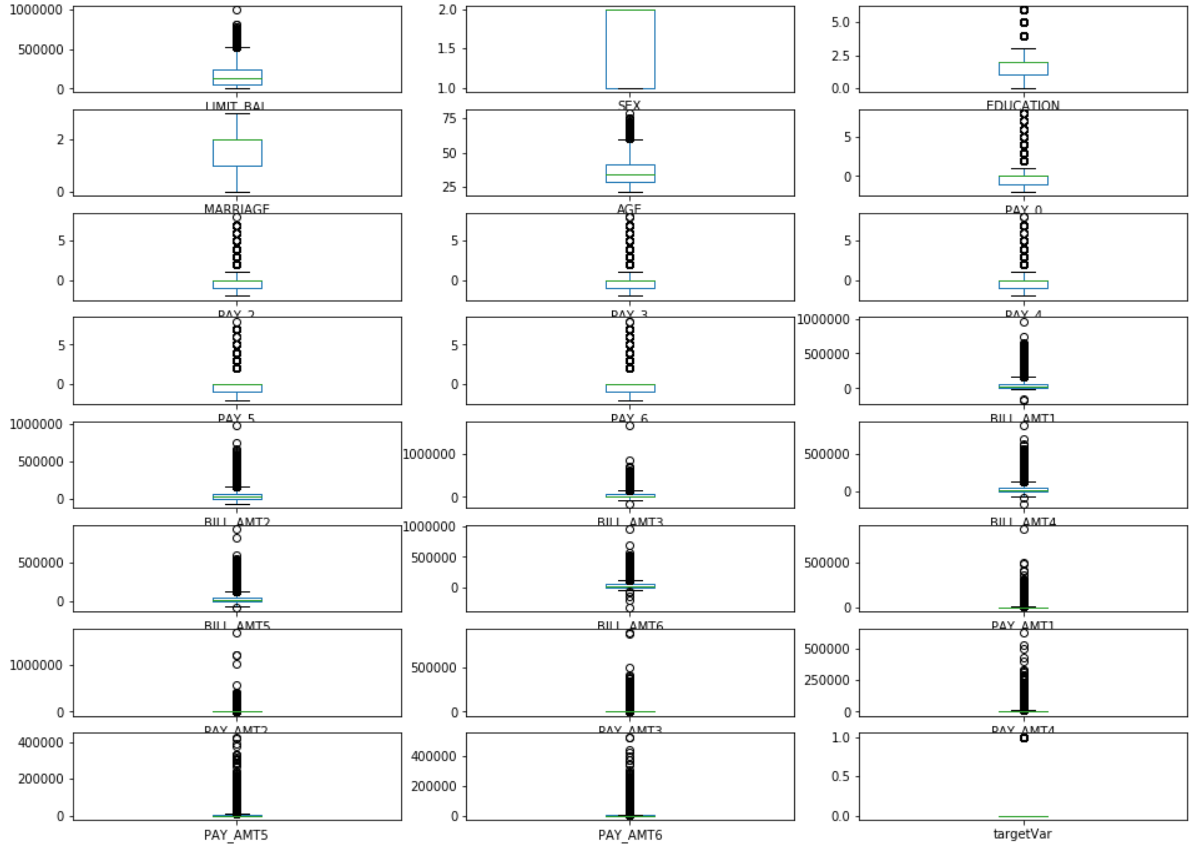


In [30]:

```
# Density plot for each attribute
entireDataset.plot(kind='density', subplots=True, layout=(dispRow,dispCol), sh
arex=False)
pyplot.show()
```

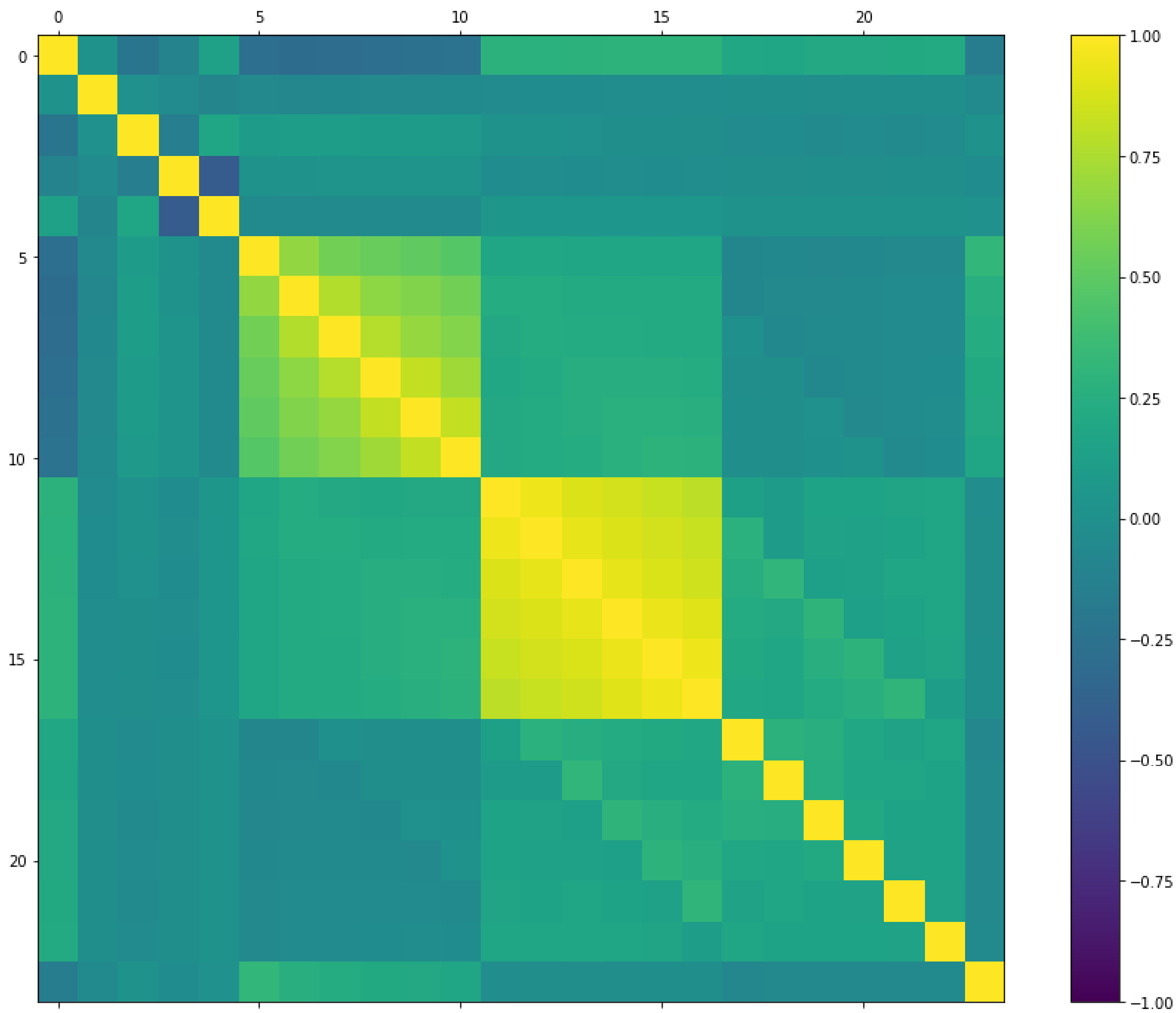


In [29]:
```python
# Box and Whisker plot for each attribute
entireDataset.plot(kind='box', subplots=True, layout=(dispRow,dispCol), sharex=False, sharey=False)
pyplot.show()
```

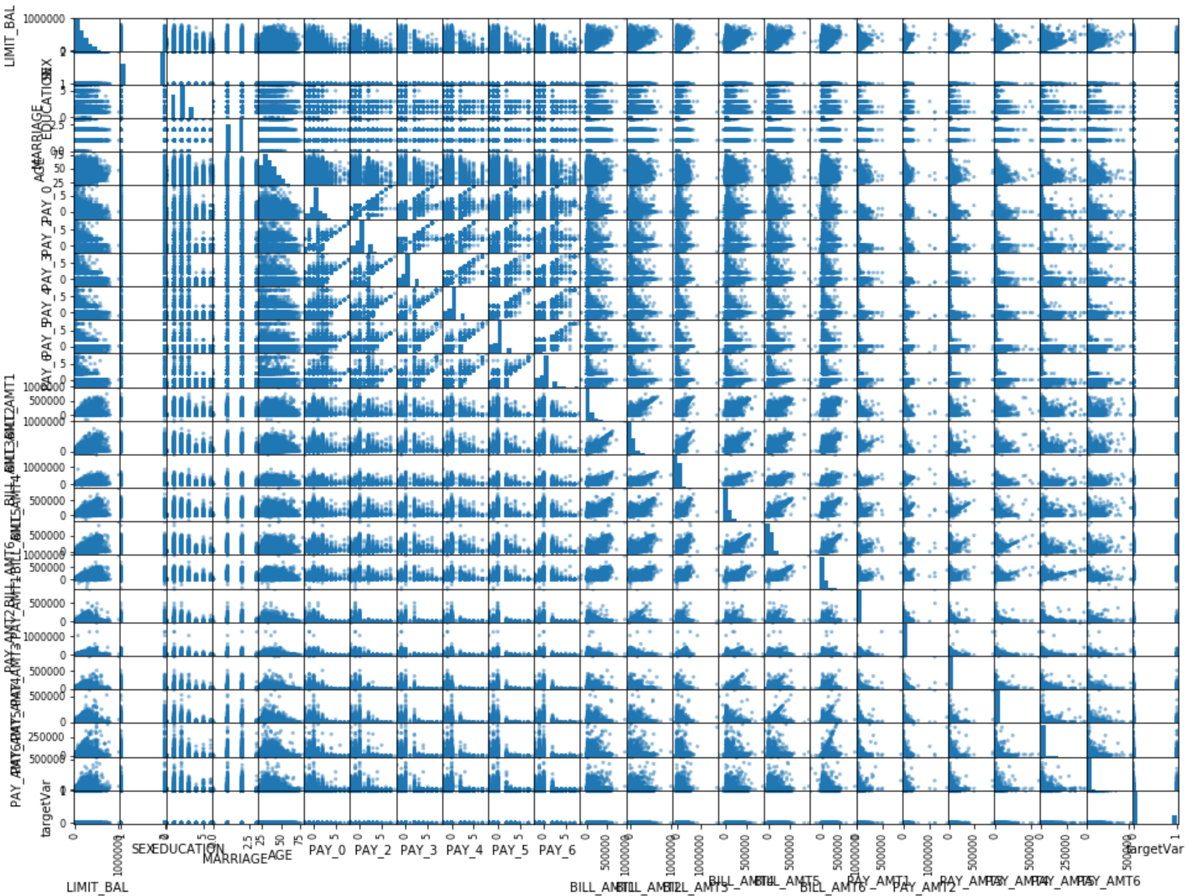


2.b.ii) Multivariate plots to better understand the relationships between attributes

In [31]:
```
# Correlation matrix
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(entireDataset.corr(), vmin=-1, vmax=1)
fig.colorbar(cax)
pyplot.show()
```



In [32]:
```
# Scatterplot matrix
scatter_matrix(entireDataset)
pyplot.show()
```



Section 3 - Prepare Data

Some dataset may require additional preparation activities that will best exposes the structure of the problem and the relationships between the input attributes and the output variable. Some data-prep tasks might include:

- Cleaning data by removing duplicates, marking missing values and even imputing missing values.
- Feature selection where redundant features may be removed.
- Data transforms where attributes are scaled or redistributed in order to best expose the structure of the problem later to learning algorithms.

3.a) Data Cleaning

In [33]:
```
# Correct the invalid values in the Education and Marital Status columns
entireDataset.EDUCATION[entireDataset.EDUCATION == 0] = 4
entireDataset.EDUCATION[entireDataset.EDUCATION == 5] = 4
entireDataset.EDUCATION[entireDataset.EDUCATION == 6] = 4
entireDataset.MARRIAGE[entireDataset.MARRIAGE == 0] = 3
```

3.b) Feature Selection

In [ ]:
```
# Not applicable for this iteration of the project.
```

3.c) Data Transforms

In [34]:
```
# Conver the credit limit attribute into bins
credit_bins = [0, 50000, 100000, 200000, 500000, 800000, 1000000]
credit_labels = [1,2,3,4,5,6]
entireDataset['CREDIT_BIN'] = pd.cut(entireDataset['LIMIT_BAL'], bins=credit_b
ins, labels=credit_labels)

# Drop the Credit Limit field after binning
entireDataset.drop('LIMIT_BAL', axis=1, inplace=True)
```

In [35]:
```
# Conver the age attribute into bins
age_bins = [20, 30, 40, 50, 60, 70, 80]
age_labels = [1,2,3,4,5,6]
entireDataset['AGE_BIN'] = pd.cut(entireDataset['AGE'], bins=age_bins, labels=
age_labels)

# Drop the Age field after binning
entireDataset.drop('AGE', axis=1, inplace=True)
```

In [36]:
```
# Conver the integer variables to categorical variables as appropriate
entireDataset["SEX"] = entireDataset["SEX"].astype('category')
entireDataset["EDUCATION"] = entireDataset["EDUCATION"].astype('category')
entireDataset["MARRIAGE"] = entireDataset["MARRIAGE"].astype('category')
entireDataset["targetVar"] = entireDataset["targetVar"].astype('category')
print(entireDataset.dtypes)
```

```
SEX            category
EDUCATION      category
MARRIAGE       category
PAY_0             int64
PAY_2             int64
PAY_3             int64
PAY_4             int64
PAY_5             int64
PAY_6             int64
BILL_AMT1         int64
BILL_AMT2         int64
BILL_AMT3         int64
BILL_AMT4         int64
BILL_AMT5         int64
BILL_AMT6         int64
PAY_AMT1          int64
PAY_AMT2          int64
PAY_AMT3          int64
PAY_AMT4          int64
PAY_AMT5          int64
PAY_AMT6          int64
targetVar      category
CREDIT_BIN     category
AGE_BIN        category
dtype: object
```

In [37]:
```python
# Apply the One-Hot-Encoding (Dummy Variables) technique
entireDataset_dummies = get_dummies(entireDataset)
entireDataset_dummies['targetVar'] = entireDataset_dummies['targetVar_1']
entireDataset_dummies.pop('targetVar_0')
entireDataset_dummies.pop('targetVar_1')
print(entireDataset_dummies.dtypes)
```

```
PAY_0           int64
PAY_2           int64
PAY_3           int64
PAY_4           int64
PAY_5           int64
PAY_6           int64
BILL_AMT1       int64
BILL_AMT2       int64
BILL_AMT3       int64
BILL_AMT4       int64
BILL_AMT5       int64
BILL_AMT6       int64
PAY_AMT1        int64
PAY_AMT2        int64
PAY_AMT3        int64
PAY_AMT4        int64
PAY_AMT5        int64
PAY_AMT6        int64
SEX_1           uint8
SEX_2           uint8
EDUCATION_1     uint8
EDUCATION_2     uint8
EDUCATION_3     uint8
EDUCATION_4     uint8
MARRIAGE_1      uint8
MARRIAGE_2      uint8
MARRIAGE_3      uint8
CREDIT_BIN_1    uint8
CREDIT_BIN_2    uint8
CREDIT_BIN_3    uint8
CREDIT_BIN_4    uint8
CREDIT_BIN_5    uint8
CREDIT_BIN_6    uint8
AGE_BIN_1       uint8
AGE_BIN_2       uint8
AGE_BIN_3       uint8
AGE_BIN_4       uint8
AGE_BIN_5       uint8
AGE_BIN_6       uint8
targetVar       uint8
dtype: object
```

```
In [38]: print(entireDataset_dummies.head())
```

```
   PAY_0  PAY_2  PAY_3  PAY_4  PAY_5  PAY_6  BILL_AMT1  BILL_AMT2  BILL_AMT3
BILL_AMT4  \
0      2      2     -1     -1     -2     -2       3913       3102        689
0
1     -1      2      0      0      0      2       2682       1725       2682
3272
2      0      0      0      0      0      0      29239      14027      13559
14331
3      0      0      0      0      0      0      46990      48233      49291
28314
4     -1      0     -1      0      0      0       8617       5670      35835
20940

     ...       CREDIT_BIN_4  CREDIT_BIN_5  CREDIT_BIN_6  AGE_BIN_1  AGE_BIN_2
AGE_BIN_3  \
0    ...                  0             0             0          1          0
0
1    ...                  0             0             0          1          0
0
2    ...                  0             0             0          0          1
0
3    ...                  0             0             0          0          1
0
4    ...                  0             0             0          0          0
0

   AGE_BIN_4  AGE_BIN_5  AGE_BIN_6  targetVar
0          0          0          0          1
1          0          0          0          1
2          0          0          0          0
3          0          0          0          0
4          1          0          0          0

[5 rows x 40 columns]
```

3.d) Split-out training and validation datasets

We create a training dataset (variable name "training") and a validation dataset (variable name "validation").

In [39]:
```
seedNum = 777
X_entire = entireDataset_dummies.loc[:, 'PAY_0':'AGE_BIN_6'].values
Y_entire = entireDataset_dummies['targetVar'].values
validation_size = 0.30
X_train, X_validation, Y_train, Y_validation = train_test_split(X_entire, Y_en
tire, test_size=validation_size, random_state=seedNum)
print("X_entire.shape: {} Y_entire.shape: {}".format(X_entire.shape, Y_entire.
shape))
print("X_train.shape: {} Y_train.shape: {}".format(X_train.shape, Y_train.shap
e))
print("X_validation.shape: {} Y_validation.shape: {}".format(X_validation.shap
e, Y_validation.shape))
print ('Total time for data handling and visualization:',(datetime.now() - sta
rtTimeScript))
```

```
X_entire.shape: (30000, 39) Y_entire.shape: (30000,)
X_train.shape: (21000, 39) Y_train.shape: (21000,)
X_validation.shape: (9000, 39) Y_validation.shape: (9000,)
Total time for data handling and visualization: 0:07:02.168140
```

4. Model and Evaluate Algorithms

After the data-prep, we next work on finding a workable model by evaluating a subset of machine learning algorithms that are good at exploiting the structure of the training. The typical evaluation tasks include:

- Defining test options such as cross validation and the evaluation metric to use.
- Spot checking a suite of linear and nonlinear machine learning algorithms.
- Comparing the estimated accuracy of algorithms.

For this project, we will evaluate one linear, four non-linear and five ensemble algorithms:

Linear Algorithm: Logistic Regression

Non-Linear Algorithms: Decision Trees (CART), Naive Bayes, k-Nearest Neighbors, and Support Vector Machine

Ensemble Algorithms: Bagged CART, Random Forest, Extra Trees, AdaBoost, and Stochastic Gradient Boosting

The random number seed is reset before each run to ensure that the evaluation of each algorithm is performed using the same data splits. It ensures the results are directly comparable.

4.a) Set test options and evaluation metric

In [40]:
```
# Run algorithms using 10-fold cross validation
num_folds = 10
scoring = 'accuracy'
```

In [41]:
```
# Set up Algorithms Spot-Checking Array
models = []
models.append(('LR', LogisticRegression(random_state=seedNum)))
models.append(('CART', DecisionTreeClassifier(random_state=seedNum)))
models.append(('NB', GaussianNB()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('SVM', SVC(random_state=seedNum)))
models.append(('BT', BaggingClassifier(random_state=seedNum)))
models.append(('RF', RandomForestClassifier(random_state=seedNum)))
models.append(('ET', ExtraTreesClassifier(random_state=seedNum)))
models.append(('AB', AdaBoostClassifier(random_state=seedNum)))
models.append(('GBM', GradientBoostingClassifier(random_state=seedNum)))
results = []
names = []
metrics = []
```

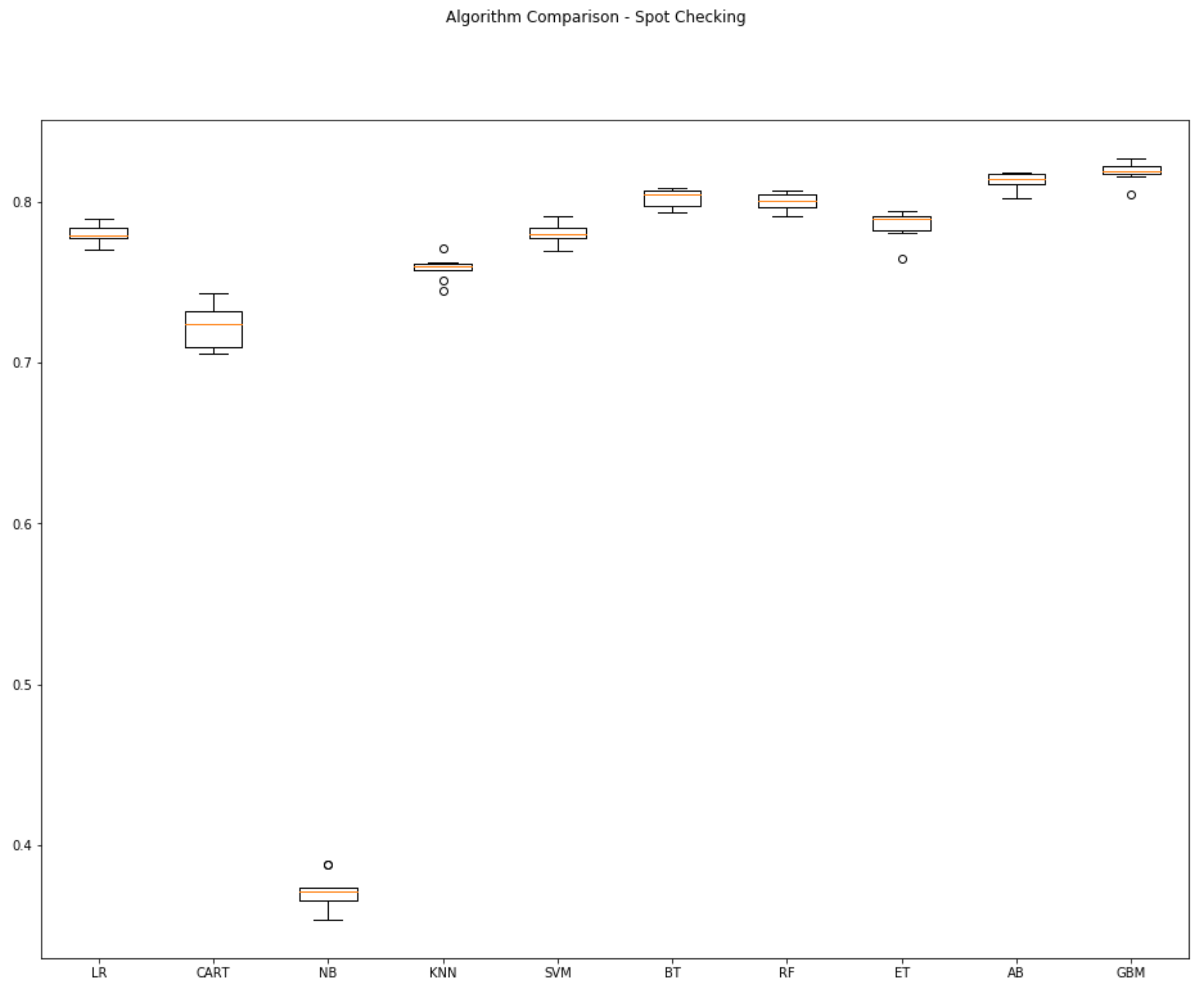
In [42]:
```
# Generate model in turn
for name, model in models:
        startTimeModule = datetime.now()
        kfold = KFold(n_splits=num_folds, random_state=seedNum)
        cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scorin
g=scoring)
        results.append(cv_results)
        names.append(name)
        metrics.append(cv_results.mean())
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)
        print ('Model training time:',(datetime.now() - startTimeModule))
print ('Average metrics (accuracy %) from all models:',np.mean(metrics))
```

```
LR: 0.780190 (0.006231)
Model training time: 0:00:05.545041
CART: 0.722476 (0.012112)
Model training time: 0:00:05.846125
NB: 0.371000 (0.010170)
Model training time: 0:00:00.388880
KNN: 0.759000 (0.006702)
Model training time: 0:00:05.865969
SVM: 0.780238 (0.006279)
Model training time: 0:43:40.136021
BT: 0.802381 (0.005730)
Model training time: 0:01:25.309026
RF: 0.800429 (0.005077)
Model training time: 0:00:13.694374
ET: 0.786381 (0.008452)
Model training time: 0:00:08.147480
AB: 0.813762 (0.004614)
Model training time: 0:00:52.830650
GBM: 0.818857 (0.005743)
Model training time: 0:02:09.900874
Average metrics (accuracy %) from all models: 0.7434714285714286
```

4.b) Spot-checking baseline algorithms

In [43]:

```
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison - Spot Checking')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```



Section 5 - Improve Accuracy

After we achieve a short list of machine learning algorithms with good level of accuracy, we can leverage ways to improve the accuracy of the models.

5.a) Algorithm Tuning

In [44]:

```
# Set up the comparison array
results = []
names = []
```

In [45]:
```
# Tuning algorithm #1 - Random Forest
startTimeModule = datetime.now()
paramGrid1 = dict(n_estimators=np.array([200,300,500,700,900]))
model1 = RandomForestClassifier(random_state=seedNum)
kfold = KFold(n_splits=num_folds, random_state=seedNum)
grid1 = GridSearchCV(estimator=model1, param_grid=paramGrid1, scoring=scoring,
cv=kfold)
grid_result1 = grid1.fit(X_train, Y_train)

print("Best: %f using %s" % (grid_result1.best_score_, grid_result1.best_param
s_))
results.append(grid_result1.cv_results_['mean_test_score'])
names.append('RF')
means = grid_result1.cv_results_['mean_test_score']
stds = grid_result1.cv_results_['std_test_score']
params = grid_result1.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
print ('Model training time:',(datetime.now() - startTimeModule))
```

```
Best: 0.814048 using {'n_estimators': 700}
0.811762 (0.005545) with: {'n_estimators': 200}
0.812524 (0.005217) with: {'n_estimators': 300}
0.813238 (0.005137) with: {'n_estimators': 500}
0.814048 (0.004519) with: {'n_estimators': 700}
0.813714 (0.004601) with: {'n_estimators': 900}
Model training time: 0:35:55.839357
```

In [46]:
```
# Tuning algorithm #2 - AdaBoost
startTimeModule = datetime.now()
paramGrid2 = dict(n_estimators=np.array([100,200,300,400,500]))
model2 = AdaBoostClassifier(random_state=seedNum)
kfold = KFold(n_splits=num_folds, random_state=seedNum)
grid2 = GridSearchCV(estimator=model2, param_grid=paramGrid2, scoring=scoring,
cv=kfold)
grid_result2 = grid2.fit(X_train, Y_train)

print("Best: %f using %s" % (grid_result2.best_score_, grid_result2.best_param
s_))
results.append(grid_result2.cv_results_['mean_test_score'])
names.append('AB')
means = grid_result2.cv_results_['mean_test_score']
stds = grid_result2.cv_results_['std_test_score']
params = grid_result2.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
print ('Model training time:',(datetime.now() - startTimeModule))
```

```
Best: 0.814905 using {'n_estimators': 200}
0.814381 (0.004865) with: {'n_estimators': 100}
0.814905 (0.005675) with: {'n_estimators': 200}
0.814857 (0.005690) with: {'n_estimators': 300}
0.814381 (0.005581) with: {'n_estimators': 400}
0.814286 (0.006102) with: {'n_estimators': 500}
Model training time: 0:15:06.559302
```

In [47]:

```
# Tuning algorithm #3 - Stochastic Gradient Boosting
startTimeModule = datetime.now()
paramGrid3 = dict(n_estimators=np.array([25,50,100,150,200]))
model3 = GradientBoostingClassifier(random_state=seedNum)
kfold = KFold(n_splits=num_folds, random_state=seedNum)
grid3 = GridSearchCV(estimator=model3, param_grid=paramGrid3, scoring=scoring,
cv=kfold)
grid_result3 = grid3.fit(X_train, Y_train)

print("Best: %f using %s" % (grid_result3.best_score_, grid_result3.best_param
s_))
results.append(grid_result3.cv_results_['mean_test_score'])
names.append('GBM')
means = grid_result3.cv_results_['mean_test_score']
stds = grid_result3.cv_results_['std_test_score']
params = grid_result3.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
print ('Model training time:',(datetime.now() - startTimeModule))
```

```
Best: 0.819714 using {'n_estimators': 50}
0.818381 (0.006523) with: {'n_estimators': 25}
0.819714 (0.006062) with: {'n_estimators': 50}
0.818857 (0.005743) with: {'n_estimators': 100}
0.817762 (0.005825) with: {'n_estimators': 150}
0.817190 (0.005976) with: {'n_estimators': 200}
Model training time: 0:06:17.400246
```

In [48]:

```
# Tuning algorithm #4 - Bagged CART
startTimeModule = datetime.now()
paramGrid4 = dict(n_estimators=np.array([50,100,150,200,250]))
model4 = BaggingClassifier(random_state=seedNum)
kfold = KFold(n_splits=num_folds, random_state=seedNum)
grid4 = GridSearchCV(estimator=model4, param_grid=paramGrid4, scoring=scoring,
cv=kfold)
grid_result4 = grid4.fit(X_train, Y_train)

print("Best: %f using %s" % (grid_result4.best_score_, grid_result4.best_param
s_))
results.append(grid_result4.cv_results_['mean_test_score'])
names.append('BT')
means = grid_result4.cv_results_['mean_test_score']
stds = grid_result4.cv_results_['std_test_score']
params = grid_result4.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
print ('Model training time:',(datetime.now() - startTimeModule))
```

```
Best: 0.811762 using {'n_estimators': 250}
0.809429 (0.004348) with: {'n_estimators': 50}
0.810286 (0.004064) with: {'n_estimators': 100}
0.810619 (0.004493) with: {'n_estimators': 150}
0.811714 (0.003945) with: {'n_estimators': 200}
0.811762 (0.004152) with: {'n_estimators': 250}
Model training time: 1:06:13.172768
```

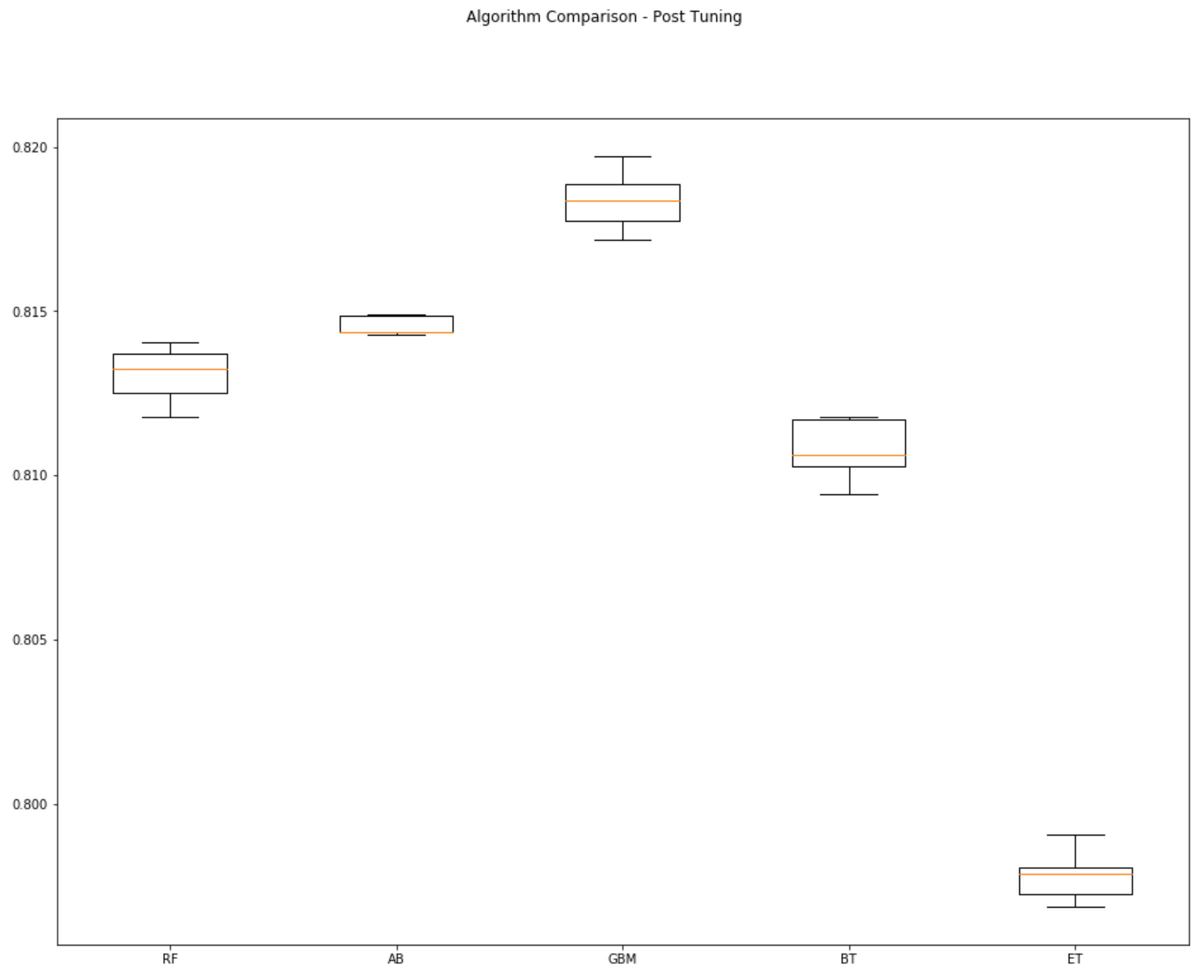
In [49]:
```
# Tuning algorithm #5 - Extra Trees
startTimeModule = datetime.now()
paramGrid5 = dict(n_estimators=np.array([100,150,200,250,300]))
model5 = ExtraTreesClassifier(random_state=seedNum)
kfold = KFold(n_splits=num_folds, random_state=seedNum)
grid5 = GridSearchCV(estimator=model5, param_grid=paramGrid5, scoring=scoring,
cv=kfold)
grid_result5 = grid5.fit(X_train, Y_train)

print("Best: %f using %s" % (grid_result5.best_score_, grid_result5.best_param
s_))
results.append(grid_result5.cv_results_['mean_test_score'])
names.append('ET')
means = grid_result5.cv_results_['mean_test_score']
stds = grid_result5.cv_results_['std_test_score']
params = grid_result5.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
print ('Model training time:',(datetime.now() - startTimeModule))
```

```
Best: 0.799048 using {'n_estimators': 100}
0.799048 (0.006124) with: {'n_estimators': 100}
0.798048 (0.005999) with: {'n_estimators': 150}
0.796857 (0.006711) with: {'n_estimators': 200}
0.797857 (0.006671) with: {'n_estimators': 250}
0.797238 (0.006851) with: {'n_estimators': 300}
Model training time: 0:11:11.527531
```

5.b) Compare Algorithms After Tuning

In [50]:
```
fig = pyplot.figure()
fig.suptitle('Algorithm Comparison - Post Tuning')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```



Section 6 - Finalize Model

Once we have narrow down to a model that we believe can make accurate predictions on unseen data, we are ready to finalize it. Finalizing a model may involve sub-tasks such as:

- Using an optimal model tuned to make predictions on unseen data.
- Creating a standalone model using the tuned parameters
- Saving an optimal model to file for later use.

6.a) Predictions on validation dataset

In [51]:
```
model = GradientBoostingClassifier(n_estimators=50, random_state=seedNum)
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)
print(accuracy_score(Y_validation, predictions))
print(confusion_matrix(Y_validation, predictions))
print(classification_report(Y_validation, predictions))
```

```
0.8275555555555556
[[6714  319]
 [1233  734]]
             precision    recall  f1-score   support

          0       0.84      0.95      0.90      7033
          1       0.70      0.37      0.49      1967

avg / total       0.81      0.83      0.81      9000
```

6.b) Create standalone model on entire training dataset

In [52]:
```
startTimeModule = datetime.now()
finalModel = GradientBoostingClassifier(n_estimators=50, random_state=seedNum)
finalModel.fit(X_entire, Y_entire)
print ('Model training time:',(datetime.now() - startTimeModule))
```

```
Model training time: 0:00:03.367504
```

6.c) Save model for later use

In [54]:
```
modelName = 'finalModel_BinaryClass.sav'
dump(finalModel, modelName)

print ('Total time for the script:',(datetime.now() - startTimeScript))
```

```
Total time for the script: 4:05:08.690040
```

In [ ]:
```
```