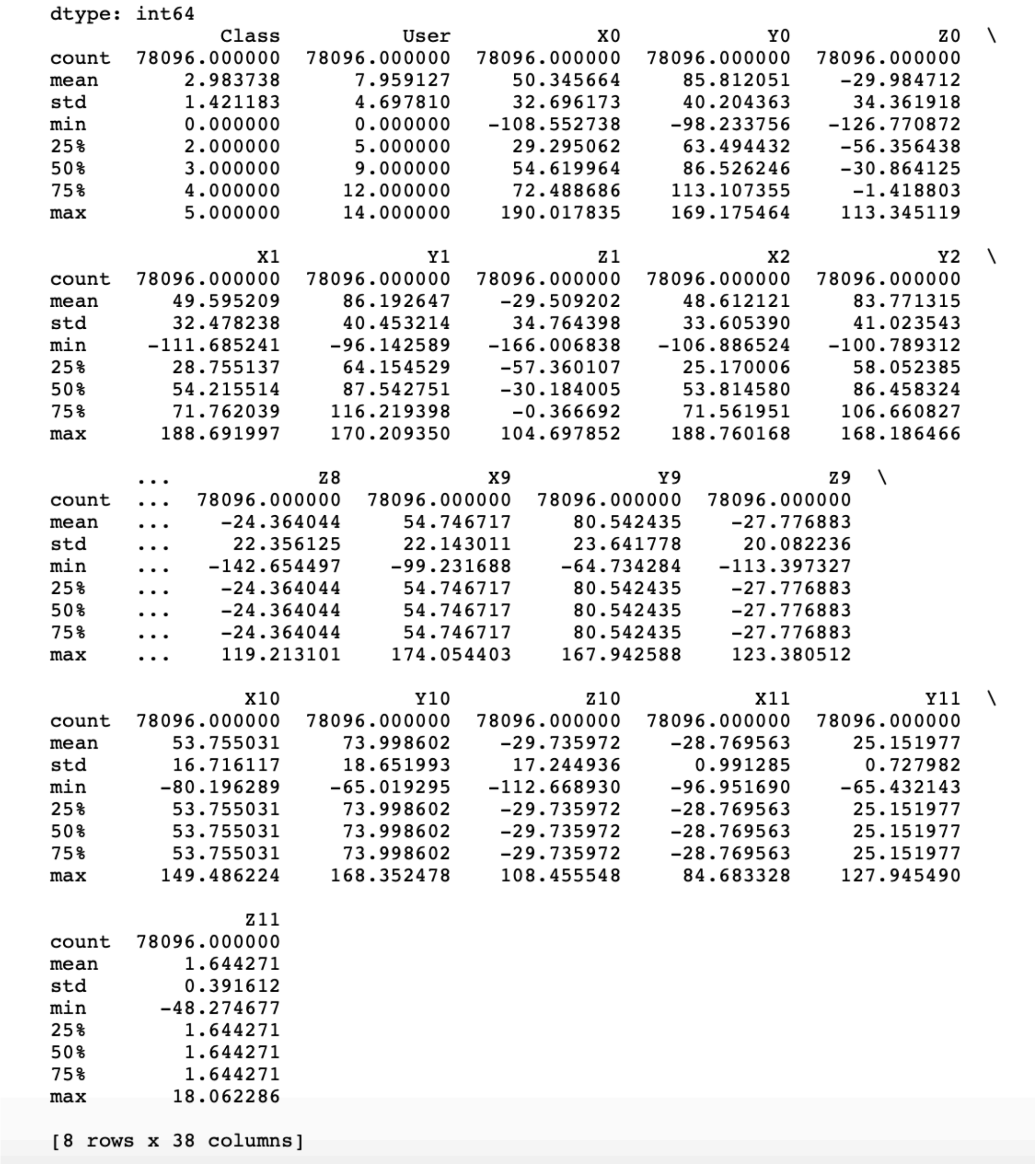
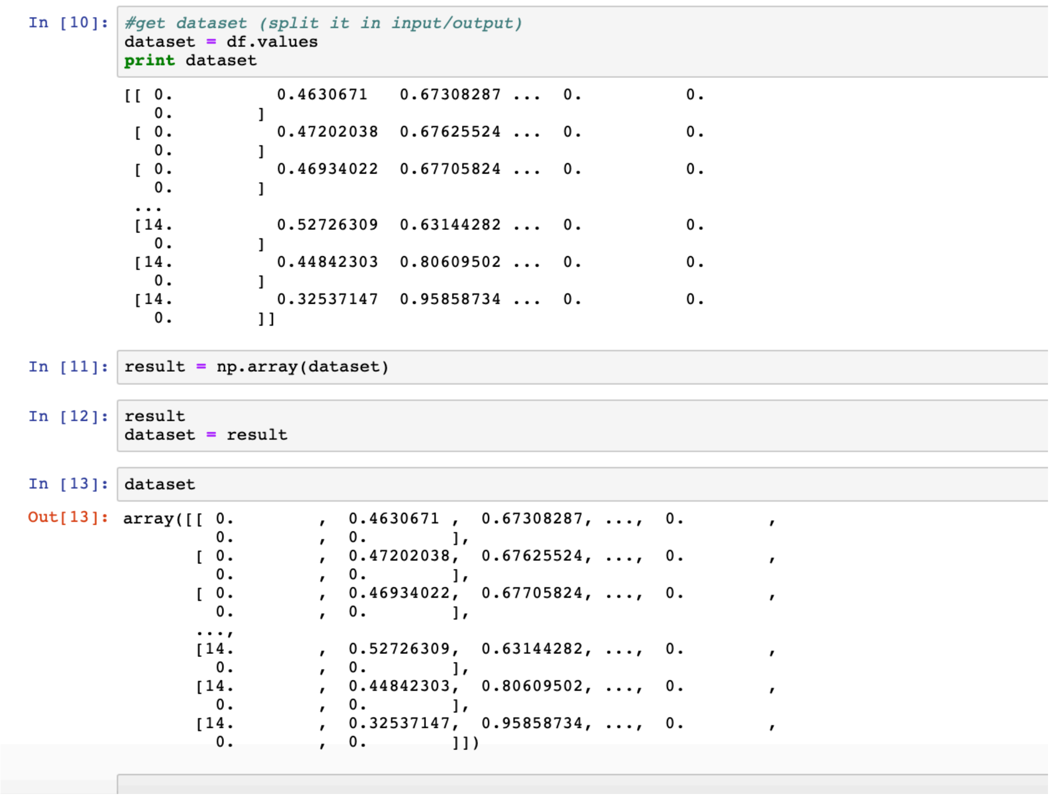
Report for Hand Posture Data Set

First of all, I implemented import essential libraries such as sklearn and all the classifiers.
The data types are float64=9, int64=2 and object=27.
For handling missing values, I fill the missing values with the 'Mean'.



```
dtype: int64
              Class          User            X0            Y0            Z0  \
count  78096.000000  78096.000000  78096.000000  78096.000000  78096.000000
mean       2.983738      7.959127     50.345664     85.812051    -29.984712
std        1.421183      4.697810     32.696173     40.204363     34.361918
min        0.000000      0.000000   -108.552738    -98.233756   -126.770872
25%        2.000000      5.000000     29.295062     63.494432    -56.356438
50%        3.000000      9.000000     54.619964     86.526246    -30.864125
75%        4.000000     12.000000     72.488686    113.107355     -1.418803
max        5.000000     14.000000    190.017835    169.175464    113.345119

                 X1            Y1            Z1            X2            Y2  \
count  78096.000000  78096.000000  78096.000000  78096.000000  78096.000000
mean      49.595209     86.192647    -29.509202     48.612121     83.771315
std       32.478238     40.453214     34.764398     33.605390     41.023543
min     -111.685241    -96.142589   -166.006838   -106.886524   -100.789312
25%       28.755137     64.154529    -57.360107     25.170006     58.052385
50%       54.215514     87.542751    -30.184005     53.814580     86.458324
75%       71.762039    116.219398     -0.366692     71.561951    106.660827
max      188.691997    170.209350    104.697852    188.760168    168.186466

       ...            Z8            X9            Y9            Z9  \
count  ...  78096.000000  78096.000000  78096.000000  78096.000000
mean   ...    -24.364044     54.746717     80.542435    -27.776883
std    ...     22.356125     22.143011     23.641778     20.082236
min    ...   -142.654497    -99.231688    -64.734284   -113.397327
25%    ...    -24.364044     54.746717     80.542435    -27.776883
50%    ...    -24.364044     54.746717     80.542435    -27.776883
75%    ...    -24.364044     54.746717     80.542435    -27.776883
max    ...    119.213101    174.054403    167.942588    123.380512

                X10           Y10           Z10           X11           Y11  \
count  78096.000000  78096.000000  78096.000000  78096.000000  78096.000000
mean      53.755031     73.998602    -29.735972    -28.769563     25.151977
std       16.716117     18.651993     17.244936      0.991285      0.727982
min      -80.196289    -65.019295   -112.668930    -96.951690    -65.432143
25%       53.755031     73.998602    -29.735972    -28.769563     25.151977
50%       53.755031     73.998602    -29.735972    -28.769563     25.151977
75%       53.755031     73.998602    -29.735972    -28.769563     25.151977
max      149.486224    168.352478    108.455548     84.683328    127.945490

                Z11
count  78096.000000
mean       1.644271
std        0.391612
min      -48.274677
25%        1.644271
50%        1.644271
75%        1.644271
max       18.062286

[8 rows x 38 columns]
```

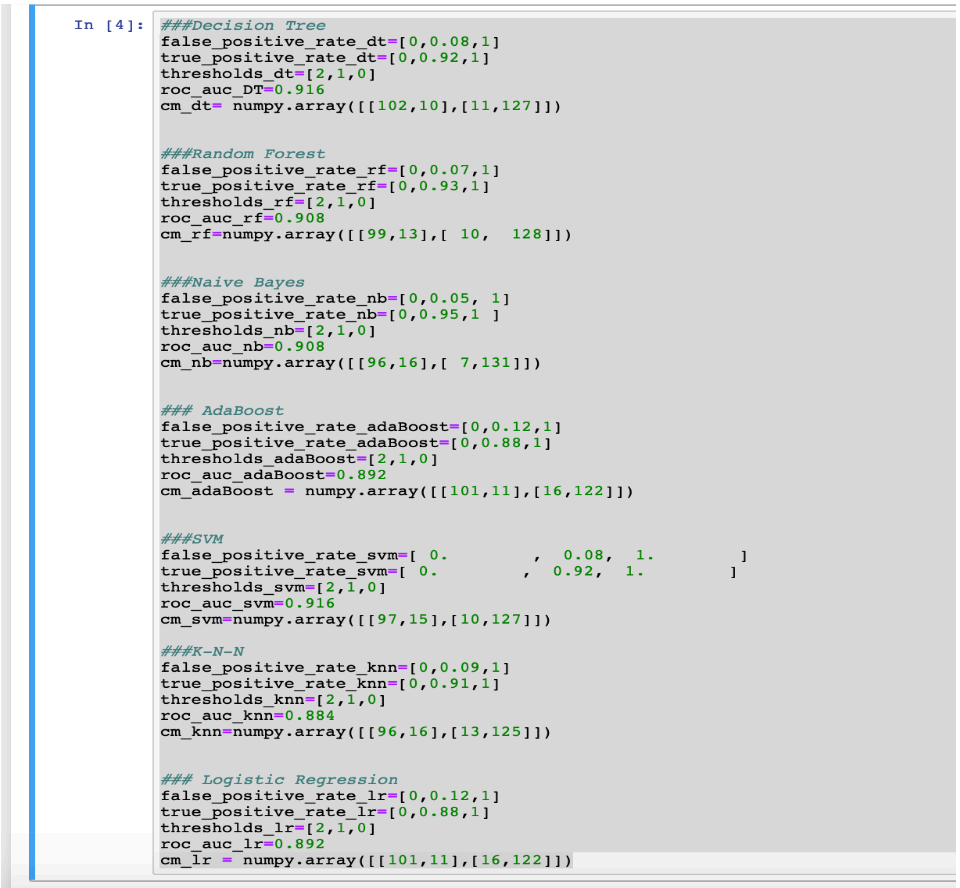
I normalized data and check again with the np.array( )



```
In [10]: #get dataset (split it in input/output)
         dataset = df.values
         print dataset

         [[ 0.          0.4630671   0.67308287 ...  0.          0.
            0.        ]
          [ 0.          0.47202038  0.67625524 ...  0.          0.
            0.        ]
          [ 0.          0.46934022  0.67705824 ...  0.          0.
            0.        ]
          ...
          [14.          0.52726309  0.63144282 ...  0.          0.
            0.        ]
          [14.          0.44842303  0.80609502 ...  0.          0.
            0.        ]
          [14.          0.32537147  0.95858734 ...  0.          0.
            0.        ]]

In [11]: result = np.array(dataset)

In [12]: result
         dataset = result

In [13]: dataset

Out[13]: array([[ 0.        ,  0.4630671 ,  0.67308287, ...,  0.        ,
                  0.        ,  0.        ],
                [ 0.        ,  0.47202038,  0.67625524, ...,  0.        ,
                  0.        ,  0.        ],
                [ 0.        ,  0.46934022,  0.67705824, ...,  0.        ,
                  0.        ,  0.        ],
                ...,
                [14.        ,  0.52726309,  0.63144282, ...,  0.        ,
                  0.        ,  0.        ],
                [14.        ,  0.44842303,  0.80609502, ...,  0.        ,
                  0.        ,  0.        ],
                [14.        ,  0.32537147,  0.95858734, ...,  0.        ,
                  0.        ,  0.        ]])
```

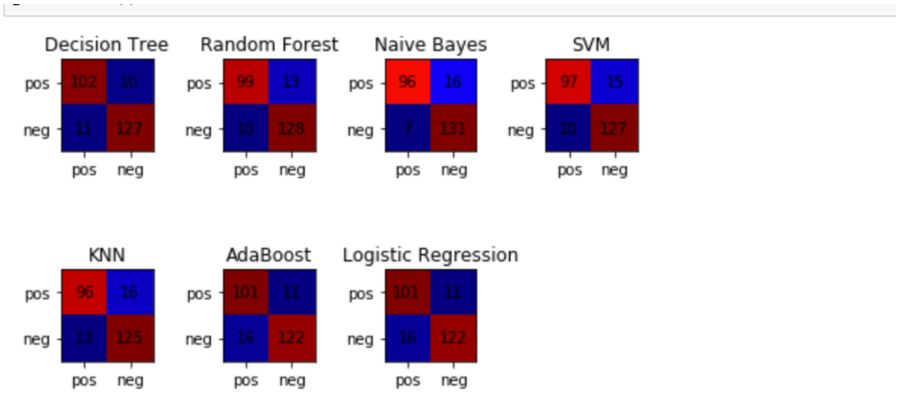
The features I implemented for this Hand Postures data set is converting integers to one hot encoded. Since this dataset doesn’t come with the test dataset so that I split into train set, validation set and test set into 0.8,0.2 ratio.
After that I can be able to apply in each of the algorithms as features create/train/fit/predict.

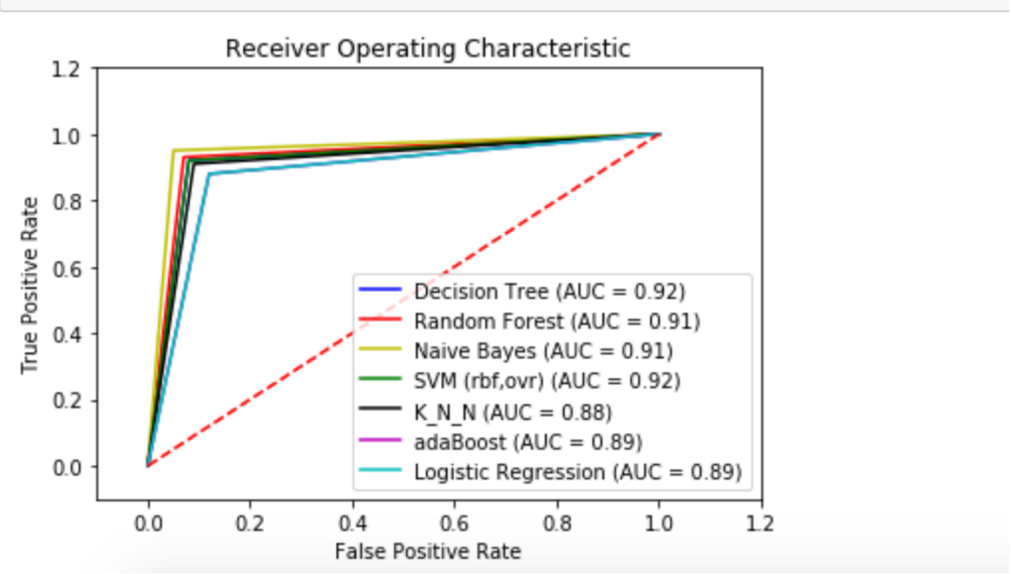
The result of each models are shown below.



```
In [4]: ###Decision Tree
        false_positive_rate_dt=[0,0.08,1]
        true_positive_rate_dt=[0,0.92,1]
        thresholds_dt=[2,1,0]
        roc_auc_DT=0.916
        cm_dt= numpy.array([[102,10],[11,127]])


        ###Random Forest
        false_positive_rate_rf=[0,0.07,1]
        true_positive_rate_rf=[0,0.93,1]
        thresholds_rf=[2,1,0]
        roc_auc_rf=0.908
        cm_rf=numpy.array([[99,13],[ 10,  128]])


        ###Naive Bayes
        false_positive_rate_nb=[0,0.05, 1]
        true_positive_rate_nb=[0,0.95,1 ]
        thresholds_nb=[2,1,0]
        roc_auc_nb=0.908
        cm_nb=numpy.array([[96,16],[ 7,131]])


        ### AdaBoost
        false_positive_rate_adaBoost=[0,0.12,1]
        true_positive_rate_adaBoost=[0,0.88,1]
        thresholds_adaBoost=[2,1,0]
        roc_auc_adaBoost=0.892
        cm_adaBoost = numpy.array([[101,11],[16,122]])


        ###SVM
        false_positive_rate_svm=[ 0.        ,  0.08,  1.        ]
        true_positive_rate_svm=[ 0.        ,  0.92,  1.        ]
        thresholds_svm=[2,1,0]
        roc_auc_svm=0.916
        cm_svm=numpy.array([[97,15],[10,127]])

        ###K-N-N
        false_positive_rate_knn=[0,0.09,1]
        true_positive_rate_knn=[0,0.91,1]
        thresholds_knn=[2,1,0]
        roc_auc_knn=0.884
        cm_knn=numpy.array([[96,16],[13,125]])


        ### Logistic Regression
        false_positive_rate_lr=[0,0.12,1]
        true_positive_rate_lr=[0,0.88,1]
        thresholds_lr=[2,1,0]
        roc_auc_lr=0.892
        cm_lr = numpy.array([[101,11],[16,122]])
```

As a result, I compare of each algorithms confusion matrix and ROC/AUC.





Conclusion:

According to the graphs, Naïve Bayes algorithms and Random Forest algorithms are two top performances upon this dataset. Since Naïve Bayes algorithms works well with multi class prediction, it need less training to apply compare to Logistic Regression.

Random Forest algorithms builds multiple decision trees and merges them together to get a more and accurate and stable prediction. Random Forest adds additional randomness to the model, while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Even though we can get good testing performance, KNN testing is very slow (it does not generalize over data in advance, and it scans historical database each time a prediction is needed and it is susceptible to noise. Also, in a very high dimensional space, the distance to all neighbors become more or less the same.

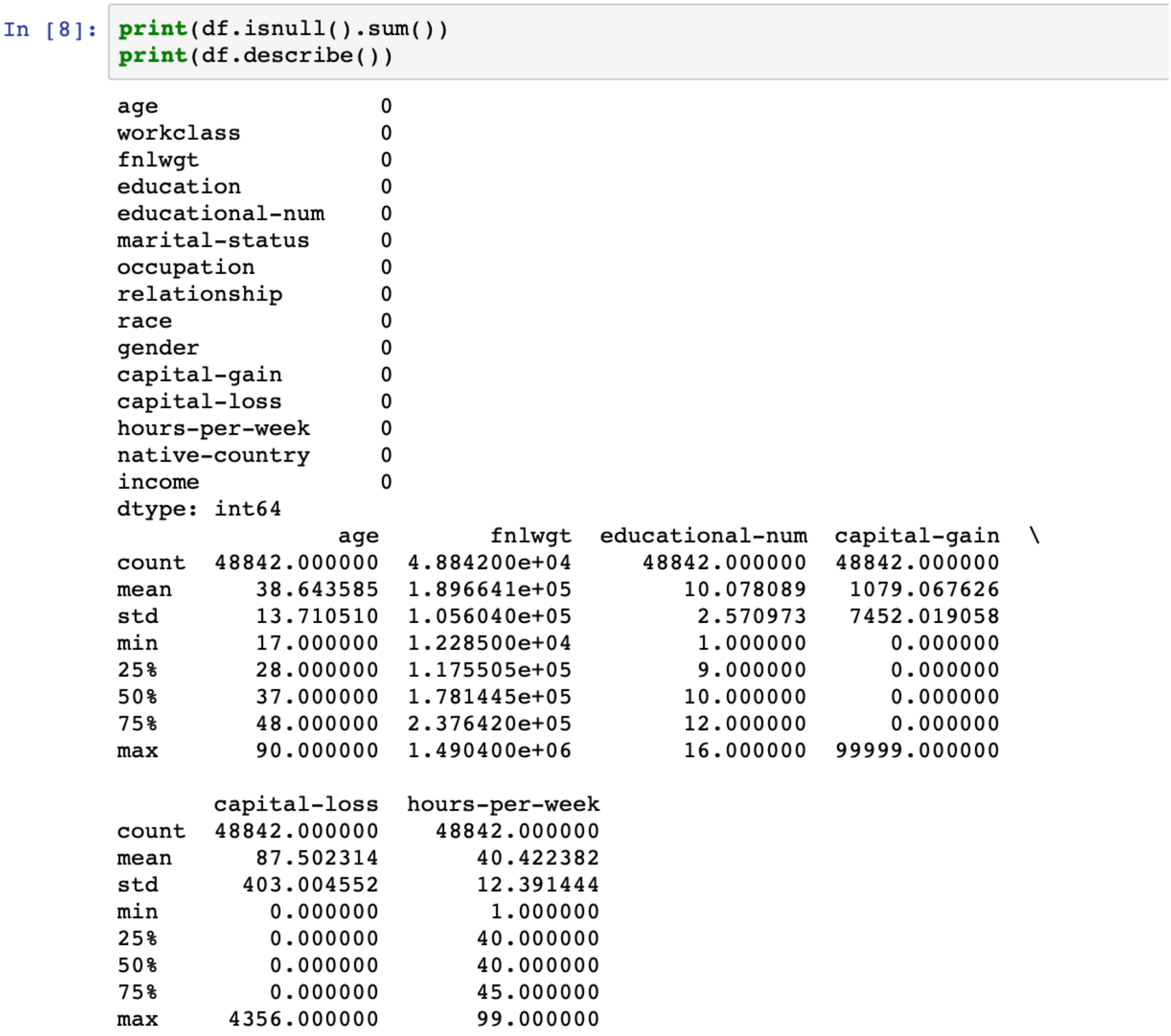
Report for adult Data Set

For adult data set, this data description is mixed with the text data, category data and numeric data. For the preprocessing data, first of all, essential thing for finding missing values. The result will be

Description about the attributes containing number of missing values

- Workclass 2799 5.73%
- Occupation 2809 5.75%
- native-country 857 1.75%

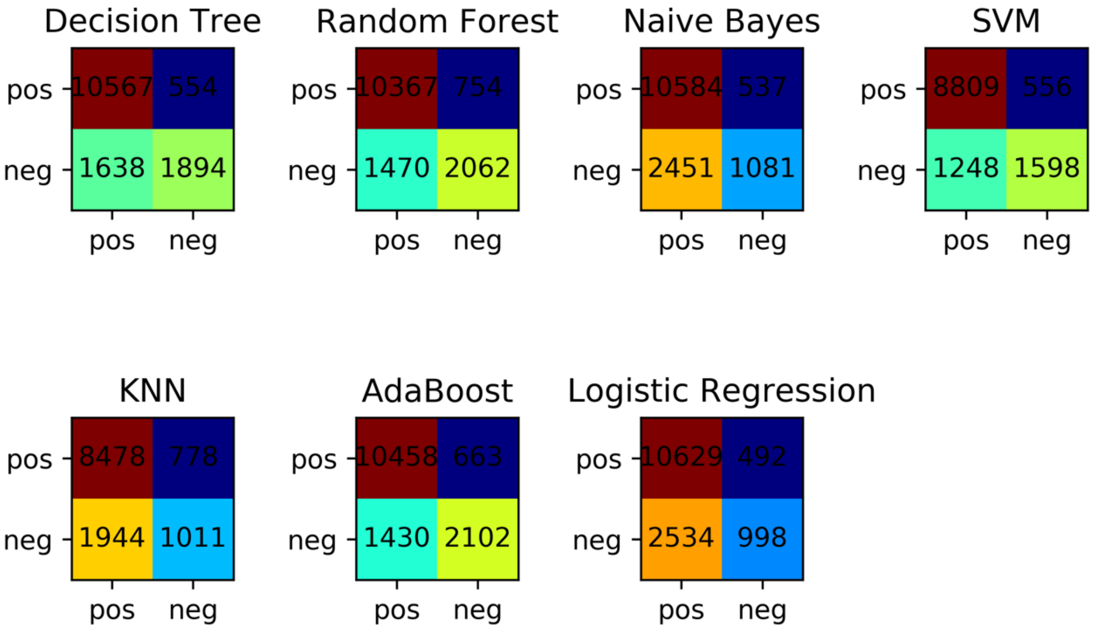
For handling missing values, I fill with missing value with NaN with top occurring value in the column.

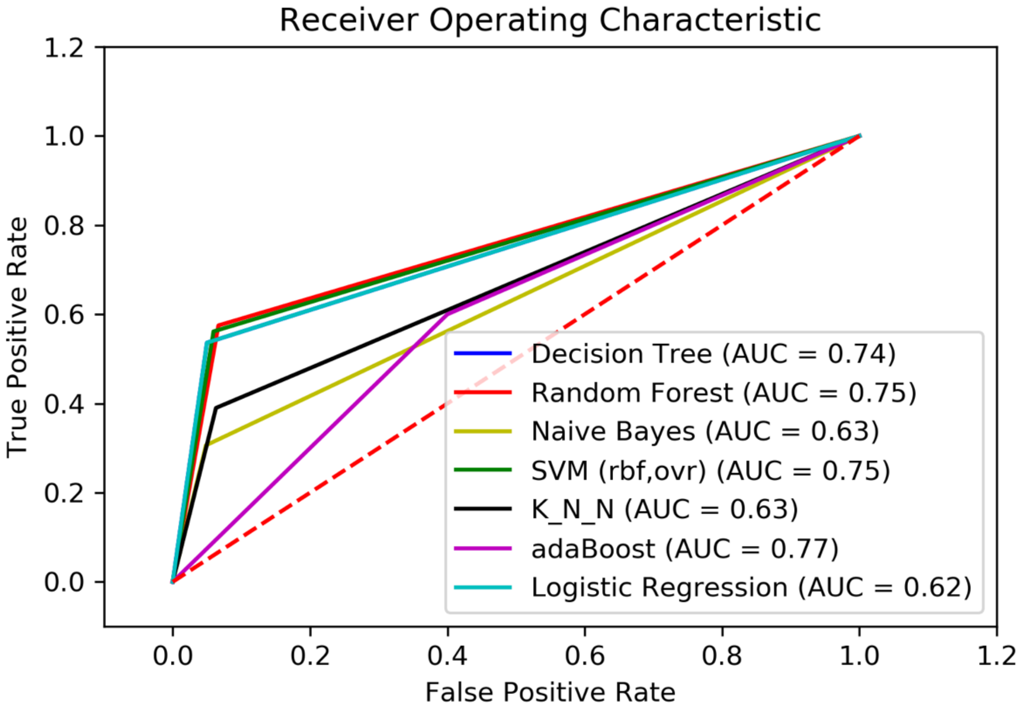


```
In [8]: print(df.isnull().sum())
        print(df.describe())
```

```
age                0
workclass          0
fnlwgt             0
education          0
educational-num    0
marital-status     0
occupation         0
relationship       0
race               0
gender             0
capital-gain       0
capital-loss       0
hours-per-week     0
native-country     0
income             0
dtype: int64
                age        fnlwgt  educational-num  capital-gain  \
count  48842.000000  4.884200e+04     48842.000000  48842.000000   
mean      38.643585  1.896641e+05        10.078089   1079.067626   
std       13.710510  1.056040e+05         2.570973   7452.019058   
min       17.000000  1.228500e+04         1.000000      0.000000   
25%       28.000000  1.175505e+05         9.000000      0.000000   
50%       37.000000  1.781445e+05        10.000000      0.000000   
75%       48.000000  2.376420e+05        12.000000      0.000000   
max       90.000000  1.490400e+06        16.000000  99999.000000   

       capital-loss  hours-per-week  
count  48842.000000    48842.000000  
mean      87.502314       40.422382  
std      403.004552       12.391444  
min        0.000000        1.000000  
25%        0.000000       40.000000  
50%        0.000000       40.000000  
75%        0.000000       45.000000  
max     4356.000000       99.000000  
```

And the I converted the categorical data into numeric data.
And I applied into each of the algorithms.


Decision Tree
pos
neg
10567
554
1638
1894
pos
neg
Random Forest
pos
neg
10367
754
1470
2062
pos
neg
Naive Bayes
pos
neg
10584
537
2451
1081
pos
neg
SVM
pos
neg
8809
556
1248
1598
pos
neg
KNN
pos
neg
8478
778
1944
1011
pos
neg
AdaBoost
pos
neg
10458
663
1430
2102
pos
neg
Logistic Regression
pos
neg
10629
492
2534
998
pos
neg


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Decision Tree (AUC = 0.74)
Random Forest (AUC = 0.75)
Naive Bayes (AUC = 0.63)
SVM (rbf,ovr) (AUC = 0.75)
K_N_N (AUC = 0.63)
adaBoost (AUC = 0.77)
Logistic Regression (AUC = 0.62)

Conclusion:

AdaBoost focuses on classification problems and aims to convert a set of weak classifiers into a strong one. AdaBoost are decision trees with one level and these trees are so short and only contain one decision for classification, I think that this is the reason why it does not perform well in that dataset.

Expectedly, Random Forest performs well on both dataset because the algorithm use multiple decision trees to make predictions.
So far, Random Forest is my favorite algorithm.