Problem Statement

The description of the dataset is as follows:

Data Set Information: Extraction was done by Barry Becker from the 1994 Census database. A set of reasonably clean records was extracted using the following conditions:

####((AAGE>16) && (AGI>100) && (AFNLWGT>1)&& (HRSWK>0))

Attribute Information: Listing of attributes:

- > 50K, <=50K.

- age: continuous.
- workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov,Without-pay, Never-worked.
- fnlwgt: continuous.
- education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool. education-num: continuous.
- marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
- occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof- specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing,

Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.

- relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
- race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
- sex: Female, Male.
- capital-gain: continuous.
- capital-loss: continuous.
- hours-per-week: continuous.
- native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import math
from pandas.plotting import scatter_matrix
%matplotlib inline
import sklearn.ensemble as ske
from sklearn import datasets, model_selection, tree, preprocessing, metrics, linear_model
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LinearRegression, LogisticRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import precision_recall_fscore_support, roc_curve, auc
```

Load training and test data

In [2]:

```
train_set = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adu

test_set = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adul

col_labels = ['age', 'workclass', 'fnlwgt', 'education', 'education_num', 'marital_status',
'occupation','relationship', 'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week
'native_country', 'wage_class']
train_set.columns = col_labels
test_set.columns = col_labels
```

In [3]:

```
train_set.shape,test_set.shape
```

Out[3]:

```
((32561, 15), (16281, 15))
```

View training and test data sample

In [4]:

```
train_set.sample(4, random_state = 42)
```

Out[4]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relations
14160	27	Private	160178	Some-college	10	Divorced	Adm-clerical	Not-in-fa
27048	45	State-gov	50567	HS-grad	9	Married-civ-spouse	Exec-managerial	\
28868	29	Private	185908	Bachelors	13	Married-civ-spouse	Exec-managerial	Husb
5667	30	Private	190040	Bachelors	13	Never-married	Machine-op-inspct	Not-in-fa

In [5]:

```
test_set.sample(4, random_state = 42)
```

Out[5]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relations
13633	29	Private	189346	HS-grad	9	Never-married	Transport-moving	Unmar
1921	31	Private	137076	Bachelors	13	Married-civ-spouse	Protective-serv	Husb
12140	52	Federal-gov	35546	HS-grad	9	Married-civ-spouse	Tech-support	Husb
9933	54	Local-gov	116428	10th	6	Married-civ-spouse	Exec-managerial	Husb

Check for null values if any in training and test data sets

In [6]:

```
train_set.isnull().sum()
```

Out[6]:

```
age               0
workclass         0
fnlwgt            0
education         0
education_num     0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
capital_gain      0
capital_loss      0
hours_per_week    0
native_country    0
wage_class        0
dtype: int64
```

In [7]:

```
test_set.isnull().sum()
```

Out[7]:

```
age               0
workclass         0
fnlwgt            0
education         0
education_num     0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
capital_gain      0
capital_loss      0
hours_per_week    0
native_country    0
wage_class        0
dtype: int64
```

In [8]:

```
pd.DataFrame([train_set.dtypes, test_set.dtypes], index = ['train_set','test_set']).T
```

Out[8]:

	train_set	test_set
age	int64	int64
workclass	object	object
fnlwgt	int64	int64
education	object	object
education_num	int64	int64
marital_status	object	object
occupation	object	object
relationship	object	object
race	object	object
sex	object	object
capital_gain	int64	int64
capital_loss	int64	int64
hours_per_week	int64	int64
native_country	object	object
wage_class	object	object

Find the columns having data types as object

In [9]:

```
for i in train_set.columns:
    if train_set[i].dtypes == 'object':
        print(i)
```

```
workclass
education
marital_status
occupation
relationship
race
sex
native_country
wage_class
```

In [10]:

```
train_set.workclass.value_counts()
```

Out[10]:

```
 Private             22696
 Self-emp-not-inc     2541
 Local-gov            2093
 ?                    1836
 State-gov            1298
 Self-emp-inc         1116
 Federal-gov           960
 Without-pay            14
 Never-worked            7
Name: workclass, dtype: int64
```

In [11]:

```
train_set.native_country.value_counts()
```

Out[11]:

```
 United-States                 29170
 Mexico                          643
 ?                               583
 Philippines                     198
 Germany                         137
 Canada                          121
 Puerto-Rico                     114
 El-Salvador                     106
 India                           100
 Cuba                             95
 England                          90
 Jamaica                          81
 South                            80
 China                            75
 Italy                            73
 Dominican-Republic               70
 Vietnam                          67
 Guatemala                        64
 Japan                            62
 Poland                           60
 Columbia                         59
 Taiwan                           51
 Haiti                            44
 Iran                             43
 Portugal                         37
 Nicaragua                        34
 Peru                             31
 Greece                           29
 France                           29
 Ecuador                          28
 Ireland                          24
 Hong                             20
 Cambodia                         19
 Trinadad&Tobago                  19
 Thailand                         18
 Laos                             18
 Yugoslavia                       16
 Outlying-US(Guam-USVI-etc)       14
 Honduras                         13
 Hungary                          13
 Scotland                         12
 Holand-Netherlands                1
Name: native_country, dtype: int64
```

In [12]:

```
train_set
```

Out[12]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black
5	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White

In [13]:

```
train_set.relationship.value_counts()
```

Out[13]:

```
 Husband           13193
 Not-in-family      8305
 Own-child          5068
 Unmarried          3446
 Wife               1568
 Other-relative      981
Name: relationship, dtype: int64
```

Unique counts for the features

In [14]:

```
train_set.workclass.nunique(),train_set.education.nunique(),train_set.marital_status.nuniqu
```

Out[14]:

```
(9, 16, 7, 42)
```

Concatenate training datasets and test datasets into a common dataframe Sample

In [15]:

```
X_train = train_set.copy()
X_test = test_set.copy()
```

In [16]:

```
X_train.columns
```

Out[16]:

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education_num',
       'marital_status', 'occupation', 'relationship', 'race', 'sex',
       'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
       'wage_class'],
      dtype='object')
```

In [17]:

```
Sample = X_train.append(X_test)
```

Summary Statistics of Continuous Values

In [18]:

```
Sample.describe()
```

Out[18]:

	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_wee
count	48842.000000	4.884200e+04	48842.000000	48842.000000	48842.000000	48842.00000
mean	38.643585	1.896641e+05	10.078089	1079.067626	87.502314	40.42238
std	13.710510	1.056040e+05	2.570973	7452.019058	403.004552	12.39144
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.00000
25%	28.000000	1.175505e+05	9.000000	0.000000	0.000000	40.00000
50%	37.000000	1.781445e+05	10.000000	0.000000	0.000000	40.00000
75%	48.000000	2.376420e+05	12.000000	0.000000	0.000000	45.00000
max	90.000000	1.490400e+06	16.000000	99999.000000	4356.000000	99.00000

Summary Statistics of Categorical Values

In [19]:

```
Sample.describe(include=['O'])
```

Out[19]:

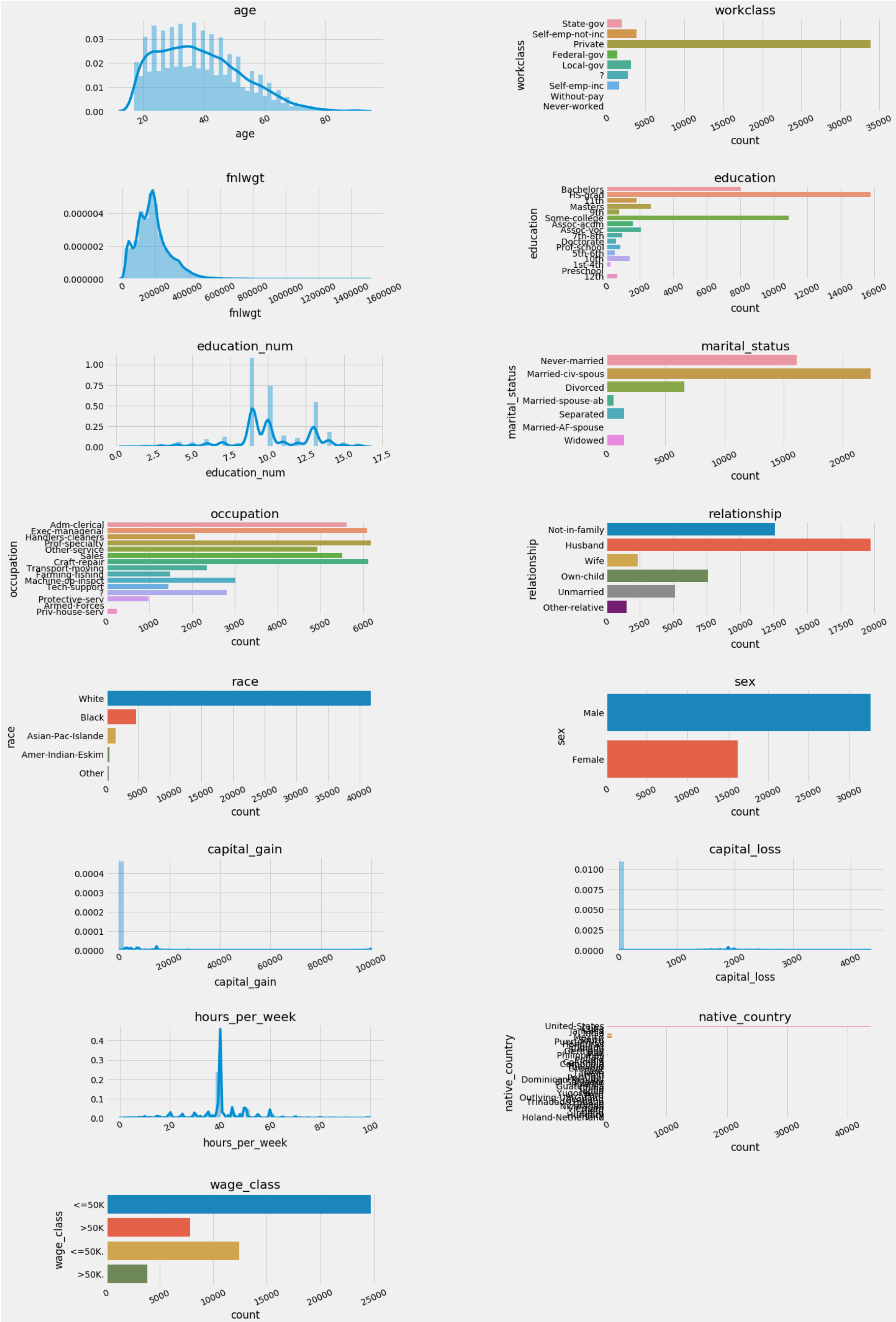
	workclass	education	marital_status	occupation	relationship	race	sex	native_co
count	48842	48842	48842	48842	48842	48842	48842	4
unique	9	16	7	15	6	5	2	
top	Private	HS-grad	Married-civ-spouse	Prof-specialty	Husband	White	Male	United-S
freq	33906	15784	22379	6172	19716	41762	32650	4

data visualization

In [20]:

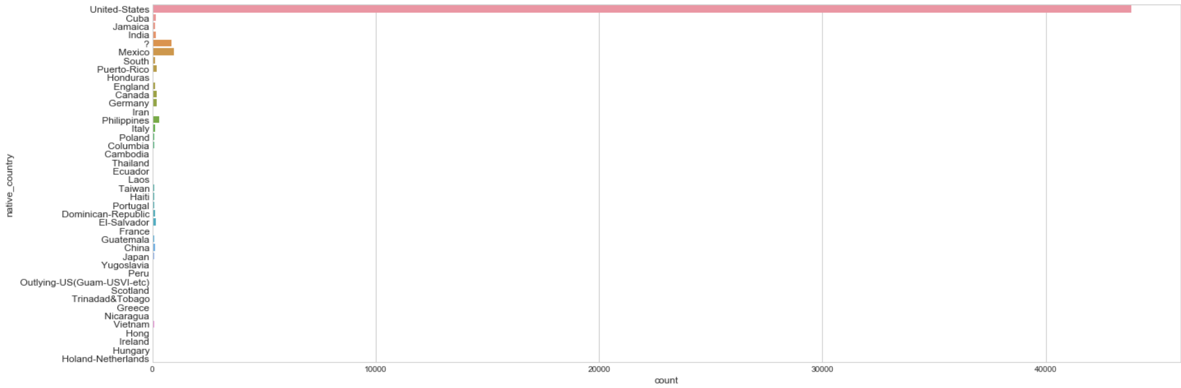
```
def plot_distribution(dataset, cols=5, width=20, height=15, hspace=0.2, wspace=0.5):
    plt.style.use('fivethirtyeight')
    fig = plt.figure(figsize=(width,height))
    fig.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=wspace, hspace
    rows = math.ceil(float(dataset.shape[1]) / cols)
    for i, column in enumerate(dataset.columns):
        ax = fig.add_subplot(rows, cols, i + 1)
        ax.set_title(column)
        if dataset.dtypes[column] == np.object:
            g = sns.countplot(y=column, data=dataset)
            substrings = [s.get_text()[:18] for s in g.get_yticklabels()]
            g.set(yticklabels=substrings)
            plt.xticks(rotation=25)
            #plt.show()
        else:
            g = sns.distplot(dataset[column])
            plt.xticks(rotation=25)
            #plt.show()
plot_distribution(Sample, cols=2, width=20, height=35, hspace=0.8, wspace=0.8)
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: U
serWarning: The 'normed' kwarg is deprecated, and has been replaced by the
'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\santhu\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: U
serWarning: The 'normed' kwarg is deprecated, and has been replaced by the
'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\santhu\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: U
serWarning: The 'normed' kwarg is deprecated, and has been replaced by the
'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\santhu\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: U
serWarning: The 'normed' kwarg is deprecated, and has been replaced by the
'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\santhu\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: U
serWarning: The 'normed' kwarg is deprecated, and has been replaced by the
'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
C:\Users\santhu\Anaconda3\lib\site-packages\matplotlib\axes\_axes.py:6462: U
serWarning: The 'normed' kwarg is deprecated, and has been replaced by the
'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```


age
0.03
0.02
0.01
0.00
20
40
60
80
age
workclass
State-gov
Self-emp-not-inc
Private
Federal-gov
Local-gov
?
Self-emp-inc
Without-pay
Never-worked
0
5000
10000
15000
20000
25000
30000
35000
count
fnlwgt
0.000004
0.000002
0.000000
0
200000
400000
600000
800000
1000000
1200000
1400000
1600000
fnlwgt
education
Bachelors
HS-grad
11th
Masters
9th
Some-college
Assoc-acdm
Assoc-voc
7th-8th
Doctorate
Prof-school
5th-6th
10th
1st-4th
Preschool
12th
0
2000
4000
6000
8000
10000
12000
14000
16000
count
education_num
1.00
0.75
0.50
0.25
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
education_num
marital_status
Never-married
Married-civ-spous
Divorced
Married-spouse-ab
Separated
Married-AF-spouse
Widowed
0
5000
10000
15000
20000
count
occupation
Adm-clerical
Exec-managerial
Handlers-cleaners
Prof-specialty
Other-service
Sales
Craft-repair
Transport-moving
Farming-fishing
Machine-op-inspct
Tech-support
?
Protective-serv
Armed-Forces
Priv-house-serv
0
1000
2000
3000
4000
5000
6000
count
relationship
Not-in-family
Husband
Wife
Own-child
Unmarried
Other-relative
0
2500
5000
7500
10000
12500
15000
17500
20000
count
race
White
Black
Asian-Pac-Islande
Amer-Indian-Eskim
Other
0
5000
10000
15000
20000
25000
30000
35000
40000
count
sex
Male
Female
0
5000
10000
15000
20000
25000
30000
count
capital_gain
0.0004
0.0003
0.0002
0.0001
0.0000
0
20000
40000
60000
80000
100000
capital_gain
capital_loss
0.0100
0.0075
0.0050
0.0025
0.0000
0
1000
2000
3000
4000
capital_loss
hours_per_week
0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100
hours_per_week
native_country
United-States
Holand-Netherland
0
10000
20000
30000
40000
count
wage_class
<=50K
>50K
<=50K.
>50K.
0
5000
10000
15000
20000
25000
count

In [21]:

```
sns.set_style('whitegrid')
%matplotlib inline
plt.figure(figsize=(20,8))
g = sns.countplot(y='native_country',data=Sample)

g.set_yticklabels(g.get_yticklabels(), rotation = 0, fontsize = 12)
plt.show()
```



Eliminating irrelevant data vale ? from the triaining and test data set

In [22]:

```
train_set = train_set.apply(lambda x : x.replace(' ?',np.nan))
test_set = test_set.apply(lambda x : x.replace(' ?',np.nan))
```

In [23]:

```
train_set.isnull().sum()
```

Out[23]:

```
age                  0
workclass         1836
fnlwgt               0
education            0
education_num        0
marital_status       0
occupation        1843
relationship         0
race                 0
sex                  0
capital_gain         0
capital_loss         0
hours_per_week       0
native_country     583
wage_class           0
dtype: int64
```

In [24]:

```
test_set.isnull().sum()
```

Out[24]:

```
age                 0
workclass         963
fnlwgt              0
education           0
education_num       0
marital_status      0
occupation        966
relationship        0
race                0
sex                 0
capital_gain        0
capital_loss        0
hours_per_week      0
native_country    274
wage_class          0
dtype: int64
```

In [25]:

```
test_set.dropna(inplace=True)
train_set.dropna(inplace=True)
```

In [26]:

```
test_set.isnull().sum(),train_set.isnull().sum()
```

Out[26]:

```
(age              0
 workclass        0
 fnlwgt           0
 education        0
 education_num    0
 marital_status   0
 occupation       0
 relationship     0
 race             0
 sex              0
 capital_gain     0
 capital_loss     0
 hours_per_week   0
 native_country   0
 wage_class       0
 dtype: int64, age              0
 workclass        0
 fnlwgt           0
 education        0
 education_num    0
 marital_status   0
 occupation       0
 relationship     0
 race             0
 sex              0
 capital_gain     0
 capital_loss     0
 hours_per_week   0
 native_country   0
 wage_class       0
 dtype: int64)
```

Converting Categorical Values to Numeric Values

In [27]:

```
dict_sex = {}
count = 0
for i in X_train.sex.unique():
    dict_sex[i] = count
    count +=1
```

In [28]:

```
dict_workclass ={}
count = 0
for i in X_train.workclass.unique():
    dict_workclass[i] = count
    count +=1
```

In [29]:

```
dict_education = {}
count = 0
for i in X_train.education.unique():
    dict_education[i] = count
    count +=1

dict_marital_status = {}
count = 0
for i in X_train.marital_status.unique():
    dict_marital_status[i] = count
    count +=1

dict_occupation = {}
count = 0
for i in X_train.occupation.unique():
    dict_occupation[i] = count
    count +=1
dict_relationship = {}
count = 0
for i in X_train.relationship.unique():
    dict_relationship[i] = count
    count +=1
dict_race = {}
count = 0
for i in X_train.race.unique():
    dict_race[i] = count
    count +=1

dict_native_country ={}
count = 0
for i in X_train.native_country.unique():
    dict_native_country[i] = count
    count +=1

dict_wage_class = {}
count = 0
for i in X_train.wage_class.unique():
    dict_wage_class[i] = count
    count +=1
```

In [30]:

```
dict_sex,dict_education,dict_wage_class,dict_native_country,dict_race,dict_occupation ,dict
```

Out[30]:

```
({' Male': 0, ' Female': 1},
 {' Bachelors': 0,
  ' HS-grad': 1,
  ' 11th': 2,
  ' Masters': 3,
  ' 9th': 4,
  ' Some-college': 5,
  ' Assoc-acdm': 6,
  ' Assoc-voc': 7,
  ' 7th-8th': 8,
  ' Doctorate': 9,
  ' Prof-school': 10,
  ' 5th-6th': 11,
  ' 10th': 12,
  ' 1st-4th': 13,
  ' Preschool': 14,
  ' 12th': 15},
 {' <=50K': 0, ' >50K': 1},
 {' United-States': 0,
  ' Cuba': 1,
  ' Jamaica': 2,
  ' India': 3,
  ' ?': 4,
  ' Mexico': 5,
  ' South': 6,
  ' Puerto-Rico': 7,
  ' Honduras': 8,
  ' England': 9,
  ' Canada': 10,
  ' Germany': 11,
  ' Iran': 12,
  ' Philippines': 13,
  ' Italy': 14,
  ' Poland': 15,
  ' Columbia': 16,
  ' Cambodia': 17,
  ' Thailand': 18,
  ' Ecuador': 19,
  ' Laos': 20,
  ' Taiwan': 21,
  ' Haiti': 22,
  ' Portugal': 23,
  ' Dominican-Republic': 24,
  ' El-Salvador': 25,
  ' France': 26,
  ' Guatemala': 27,
  ' China': 28,
  ' Japan': 29,
  ' Yugoslavia': 30,
  ' Peru': 31,
  ' Outlying-US(Guam-USVI-etc)': 32,
  ' Scotland': 33,
  ' Trinadad&Tobago': 34,
  ' Greece': 35,
  ' Nicaragua': 36,
```

```
 ' Vietnam': 37,
 ' Hong': 38,
 ' Ireland': 39,
 ' Hungary': 40,
 ' Holand-Netherlands': 41},
{' White': 0,
 ' Black': 1,
 ' Asian-Pac-Islander': 2,
 ' Amer-Indian-Eskimo': 3,
 ' Other': 4},
{' Adm-clerical': 0,
 ' Exec-managerial': 1,
 ' Handlers-cleaners': 2,
 ' Prof-specialty': 3,
 ' Other-service': 4,
 ' Sales': 5,
 ' Craft-repair': 6,
 ' Transport-moving': 7,
 ' Farming-fishing': 8,
 ' Machine-op-inspct': 9,
 ' Tech-support': 10,
 ' ?': 11,
 ' Protective-serv': 12,
 ' Armed-Forces': 13,
 ' Priv-house-serv': 14},
{' Never-married': 0,
 ' Married-civ-spouse': 1,
 ' Divorced': 2,
 ' Married-spouse-absent': 3,
 ' Separated': 4,
 ' Married-AF-spouse': 5,
 ' Widowed': 6})
```

In [31]:

```
X_train['sex'] = X_train['sex'].map(dict_sex)
X_train['education'] = X_train['education'].map(dict_education)
X_train['wage_class'] = X_train['wage_class'].map(dict_wage_class)
X_train['native_country'] = X_train['native_country'].map(dict_native_country)
X_train['race'] = X_train['race'].map(dict_race)
X_train['occupation']=X_train['occupation'].map(dict_occupation)
X_train['marital_status'] = X_train['marital_status'].map(dict_marital_status)
X_train['workclass'] = X_train['workclass'].map(dict_workclass)
X_train['relationship'] = X_train['relationship'].map(dict_relationship)
```

In [32]:

```
X_train.isnull().sum()
```

Out[32]:

```
age               0
workclass         0
fnlwgt            0
education         0
education_num     0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
capital_gain      0
capital_loss      0
hours_per_week    0
native_country    0
wage_class        0
dtype: int64
```

In [33]:

```
Xtrain = X_train.astype(int)
```

In [34]:

```
X_train.head()
```

Out[34]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship
0	39	0	77516	0	13	0	0	0
1	50	1	83311	0	13	1	1	1
2	38	2	215646	1	9	2	2	0
3	53	2	234721	2	7	1	2	1
4	28	2	338409	0	13	1	3	2

In [35]:

```
X_train.describe()
```

Out[35]:

	age	workclass	fnlwgt	education	education_num	marital_status
count	32561.000000	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000
mean	38.581647	2.309972	1.897784e+05	3.424465	10.080679	1.083781
std	13.640433	1.225728	1.055500e+05	3.453582	2.572720	1.251381
min	17.000000	0.000000	1.228500e+04	0.000000	1.000000	0.000000
25%	28.000000	2.000000	1.178270e+05	1.000000	9.000000	0.000000
50%	37.000000	2.000000	1.783560e+05	2.000000	10.000000	1.000000
75%	48.000000	2.000000	2.370510e+05	5.000000	12.000000	1.000000
max	90.000000	8.000000	1.484705e+06	15.000000	16.000000	6.000000

In [36]:

```
print(X_train.wage_class.value_counts())
print(X_test.wage_class.value_counts())
```

```
0    24720
1     7841
Name: wage_class, dtype: int64
 <=50K.    12435
 >50K.      3846
Name: wage_class, dtype: int64
```

In [37]:

```
dict_wage_class = {}
count = 0
for i in X_test.wage_class.unique():
    dict_wage_class[i] = count
    count +=1

dict_native_country ={}
count = 0
for i in X_test.native_country.unique():
    dict_native_country[i] = count
    count +=1
```

In [38]:

```
X_test['sex'] = X_test['sex'].map(dict_sex)
X_test['education'] = X_test['education'].map(dict_education)
X_test['wage_class'] = X_test['wage_class'].map(dict_wage_class)
X_test['native_country'] = X_test['native_country'].map(dict_native_country)
X_test['race'] = X_test['race'].map(dict_race)
X_test['occupation']=X_test['occupation'].map(dict_occupation)
X_test['marital_status'] = X_test['marital_status'].map(dict_marital_status)
X_test['workclass'] = X_test['workclass'].map(dict_workclass)
X_test['relationship'] = X_test['relationship'].map(dict_relationship)
```

In [39]:

```
dict_wage_class
```

Out[39]:

```
{' <=50K.': 0, ' >50K.': 1}
```

In [40]:

```
X_test.head()
```

Out[40]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship
0	25	2	226802	2	7	0	9	3
1	38	2	89814	1	9	1	8	1
2	28	4	336951	6	12	1	12	1
3	44	2	160323	5	10	1	9	1
4	18	5	103497	5	10	0	11	3

In [41]:

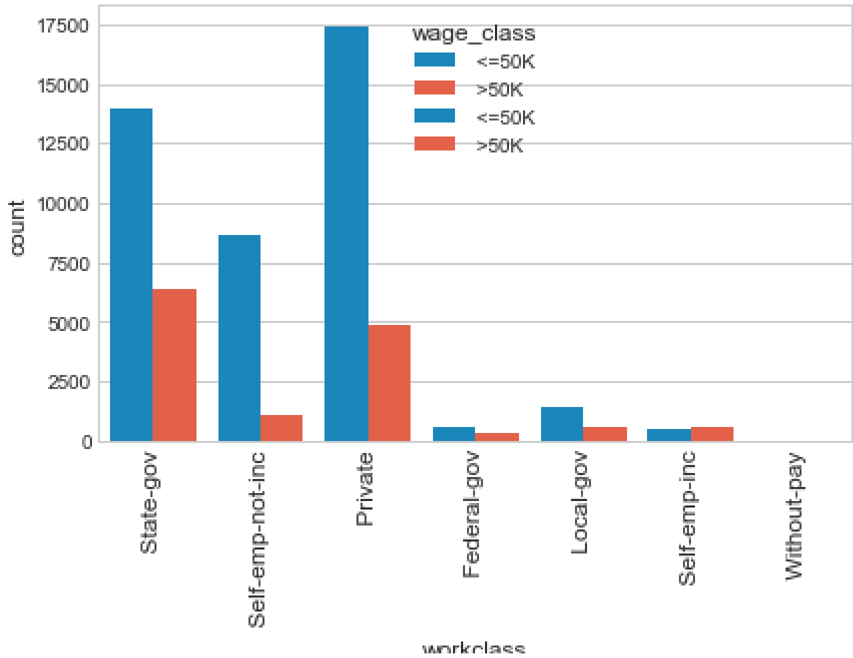
```
X_test.describe()
```

Out[41]:

	age	workclass	fnlwgt	education	education_num	marital_status
count	16281.000000	16281.000000	1.628100e+04	16281.000000	16281.000000	16281.000000
mean	38.767459	2.315030	1.894357e+05	3.386954	10.072907	1.084270
std	13.849187	1.246499	1.057149e+05	3.440725	2.567545	1.269622
min	17.000000	0.000000	1.349200e+04	0.000000	1.000000	0.000000
25%	28.000000	2.000000	1.167360e+05	1.000000	9.000000	0.000000
50%	37.000000	2.000000	1.778310e+05	2.000000	10.000000	1.000000
75%	48.000000	2.000000	2.383840e+05	5.000000	12.000000	1.000000
max	90.000000	8.000000	1.490400e+06	15.000000	16.000000	6.000000

In [42]:

```
# Annual Income Data Analysis using Visualization

import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(40,20))
sns.set_style('whitegrid')
%matplotlib inline
sns.countplot('sex',data=train_set,hue='wage_class')

g = sns.countplot('workclass',data=train_set,hue='wage_class')
g.set_xticklabels(g.get_xticklabels(), rotation = 90, fontsize = 12)
plt.show()


g = sns.countplot('education',data=train_set,hue='wage_class')
g.set_xticklabels(g.get_xticklabels(), rotation = 90, fontsize = 12)
plt.show()

g = sns.countplot('wage_class',data=train_set)
g.set_xticklabels(g.get_xticklabels(), rotation = 45, fontsize = 12)
plt.show()

pd.DataFrame.hist(train_set,figsize = [15,15])
plt.show()
```



In [43]:

```
l = X_train.append(X_test)
l.wage_class.value_counts()

Features = l.drop('wage_class',axis=1)
Labels = l['wage_class']

Features.native_country.unique()
```

Out[43]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41], dtype=int64)
```

Separating the Training Label and Test Label from the the training and test Features variables

In [44]:

```
x_train = X_train.drop('wage_class',axis=1)
y_train = X_train['wage_class']

x_test = X_test.drop('wage_class',axis=1)
y_test = X_test['wage_class']
```

In [45]:

```
X = x_train.values
Y = y_train.values
```

Validation Features and Labels

In [46]:

```
Xtest = x_test.values
Ytest = y_test.values
```

In [47]:

```
x_train.shape,y_train.shape,X.shape,Y.shape,Xtest.shape,Ytest.shape
```

Out[47]:

```
((32561, 14), (32561,), (32561, 14), (32561,), (16281, 14), (16281,))
```

In [48]:

```
Xtest
```

Out[48]:

```
array([[    25,      2, 226802, ...,      0,     40,      0],
       [    38,      2,  89814, ...,      0,     50,      0],
       [    28,      4, 336951, ...,      0,     40,      0],
       ...,
       [    38,      2, 374983, ...,      0,     50,      0],
       [    44,      2,  83891, ...,      0,     40,      0],
       [    35,      6, 182148, ...,      0,     60,      0]], dtype=int64)
```

Logistic Regression

In [49]:

```
model_accuracy = {}
#Build the model
LR = LogisticRegression()
#traing the model
LR.fit(X,Y)
#Model parameters study
Ypred = LR.predict(Xtest)
Ypred_proba = LR.predict_proba(Xtest)
# generate evaluation metrics
print(metrics.accuracy_score(Ytest, Ypred))
model_accuracy['Logistic Regression'] = metrics.accuracy_score(Ytest, Ypred)
```

```
0.8002579694121983
```

In [50]:

```
test_data =[]
for i in x_test.columns:
    test_data.append(x_test[i].max())
print(test_data)
test = np.array(test_data).reshape(-1,14)
print(test.shape)
print("Predicted Label \n ")
print(LR.predict(test))
print('Prediction Probabilities \n ')
print(LR.predict_proba(test))
print('coefficients = ',LR.coef_)
```

```
[90, 8, 1490400, 15, 16, 6, 14, 5, 4, 1, 99999, 3770, 99, 40]
(1, 14)
Predicted Label

[1]
Prediction Probabilities

[[1.26076927e-12 1.00000000e+00]]
coefficients =  [[-1.18083782e-03 -3.23061154e-03 -4.49424085e-06 -9.7512202
0e-03
  -1.53235632e-03 -1.74086823e-03 -1.49893813e-02 -8.22045814e-03
  -1.34537842e-03 -2.97005889e-03  3.26383422e-04  7.41484784e-04
  -5.96609395e-03 -4.26655749e-03]]
```

Evaluation of Logistic Regression Model

Confusion Matrix

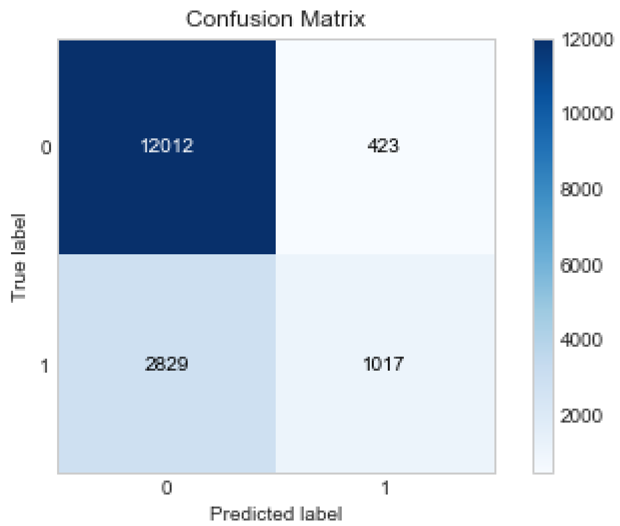
In [57]:

```
!pip install scikit-plot
```

```
Collecting scikit-plot
  Downloading https://files.pythonhosted.org/packages/7c/47/32520e259340c140
a4ad27c1b97050dd3254fdc517b1d59974d47037510e/scikit_plot-0.3.7-py3-none-any.
whl (https://files.pythonhosted.org/packages/7c/47/32520e259340c140a4ad27c1b
97050dd3254fdc517b1d59974d47037510e/scikit_plot-0.3.7-py3-none-any.whl)
Requirement already satisfied: scikit-learn>=0.18 in c:\users\santhu\anacond
a3\lib\site-packages (from scikit-plot) (0.19.1)
Collecting joblib>=0.10 (from scikit-plot)
  Downloading https://files.pythonhosted.org/packages/0d/1b/995167f6c66848d4
eb7eabc386aebe07a1571b397629b2eac3b7bebdc343/joblib-0.13.0-py2.py3-none-any.
whl (https://files.pythonhosted.org/packages/0d/1b/995167f6c66848d4eb7eabc38
6aebe07a1571b397629b2eac3b7bebdc343/joblib-0.13.0-py2.py3-none-any.whl) (276
kB)
Requirement already satisfied: matplotlib>=1.4.0 in c:\users\santhu\anaconda
3\lib\site-packages (from scikit-plot) (2.2.2)
Requirement already satisfied: scipy>=0.9 in c:\users\santhu\anaconda3\lib\s
ite-packages (from scikit-plot) (1.1.0)
Requirement already satisfied: numpy>=1.7.1 in c:\users\santhu\anaconda3\lib
\site-packages (from matplotlib>=1.4.0->scikit-plot) (1.14.3)
Requirement already satisfied: cycler>=0.10 in c:\users\santhu\anaconda3\lib
\site-packages (from matplotlib>=1.4.0->scikit-plot) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
 c:\users\santhu\anaconda3\lib\site-packages (from matplotlib>=1.4.0->scikit
-plot) (2.2.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\santhu\anaco
nda3\lib\site-packages (from matplotlib>=1.4.0->scikit-plot) (2.7.3)
Requirement already satisfied: pytz in c:\users\santhu\anaconda3\lib\site-pa
ckages (from matplotlib>=1.4.0->scikit-plot) (2018.4)
Requirement already satisfied: six>=1.10 in c:\users\santhu\anaconda3\lib\si
te-packages (from matplotlib>=1.4.0->scikit-plot) (1.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\santhu\anaconda
3\lib\site-packages (from matplotlib>=1.4.0->scikit-plot) (1.0.1)
Requirement already satisfied: setuptools in c:\users\santhu\anaconda3\lib\s
ite-packages (from kiwisolver>=1.0.1->matplotlib>=1.4.0->scikit-plot) (39.1.
0)
Installing collected packages: joblib, scikit-plot
Successfully installed joblib-0.13.0 scikit-plot-0.3.7
```

In [58]:

```
import scikitplot
scikitplot.metrics.plot_confusion_matrix(Ytest,Ypred)
print()
```

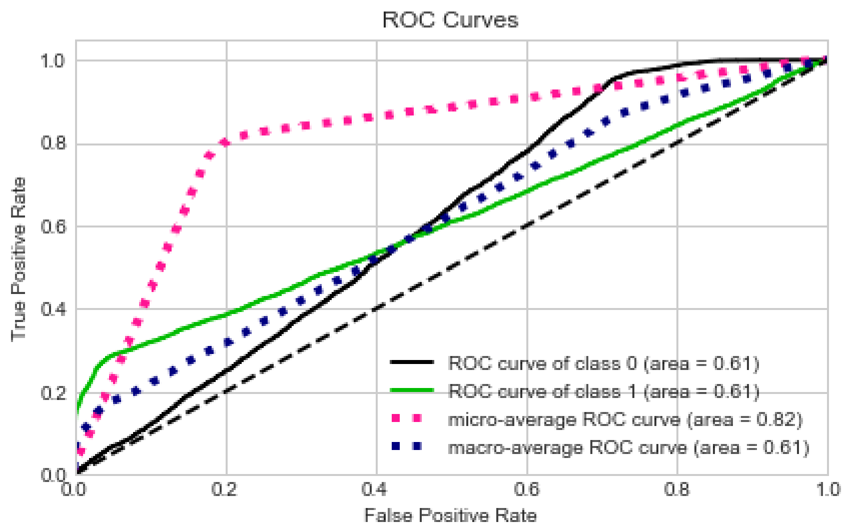


In [59]:

```
scikitplot.metrics.plot_roc(Ytest,Ypred_proba)
```

Out[59]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xb6b9240>
```

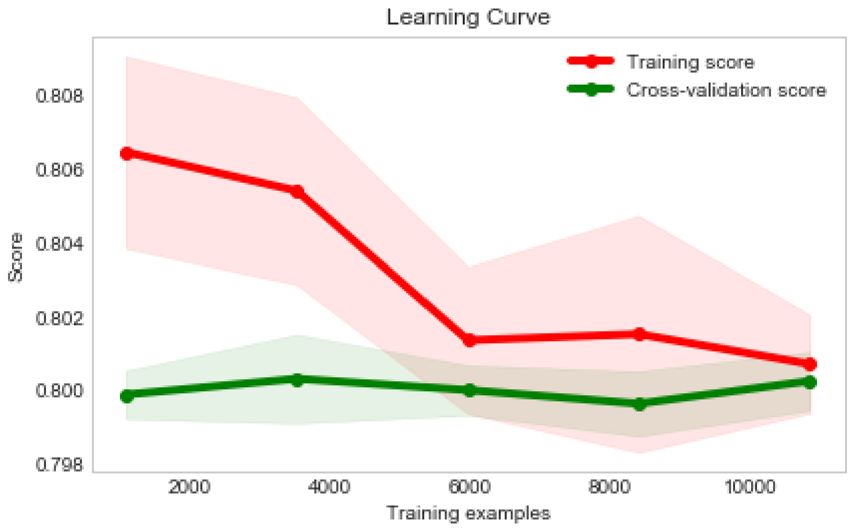


In [60]:

```
scikitplot.estimators.plot_learning_curve(LR,Xtest,Ytest)
```

Out[60]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xe0ed240>
```



In [61]:

```
print("ROC : ",(metrics.roc_curve(Ytest,Ypred_proba[:,1])))
print("AUC : ",(metrics.roc_auc_score(Ytest,Ypred_proba[:,1])))
```

```
ROC :  (array([0.        , 0.        , 0.        , ..., 0.99935665, 0.999356
65,
       1.        ]), array([2.60010400e-04, 7.80031201e-04, 1.30005200e-03,
...,
       9.99739990e-01, 1.00000000e+00, 1.00000000e+00]), array([1.00000000e+
00, 1.00000000e+00, 1.00000000e+00, ...,
       1.19127297e-02, 1.08109010e-02, 8.52835137e-04]))
AUC :  0.6113746865918063
```

In [64]:

```
model_accuracy['AUC_Logistic_Regression'] = metrics.roc_auc_score(Ytest,Ypred_proba[:,1])
```

In [63]:

```
from sklearn.metrics import classification_report
print(classification_report(Ytest,Ypred))
```

```
             precision    recall  f1-score   support

          0       0.81      0.97      0.88     12435
          1       0.71      0.26      0.38      3846

avg / total       0.79      0.80      0.76     16281
```

Applying 10 Fold Cross Validation to Logistic Regression Model

In [65]:

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(estimator= LogisticRegression(),     # Model to test
                X= Features,
                y = Labels,      # Target variable
                scoring = "accuracy",              # Scoring metric
                cv=10)                             # Cross validation folds

print("Accuracy per fold: ")
print("Cross Validation score: ", scores)
print("Average accuracy: ", scores.mean())
model_accuracy['10 CV Score-Logistic Regression'] = scores.mean()
```

```
Accuracy per fold:
Cross Validation score:  [0.7975435  0.79672467 0.79897646 0.80204708 0.7948
8229 0.80118755
 0.80405405 0.79909891 0.79582224 0.79848454]
Average accuracy:  0.7988821300510068
```

Feature Selection for Logistic Regression Model

In [66]:

```
from sklearn.feature_selection import RFE, RFECV
selector =  RFE(estimator=LogisticRegression(), step=1)
selector.fit(Features,Labels)
```

Out[66]:

```
RFE(estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_i
ntercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
  n_features_to_select=None, step=1, verbose=0)
```

In [67]:

```
ranks = selector.ranking_.tolist()
ranks
```

Out[67]:

```
[2, 1, 8, 4, 1, 1, 1, 1, 1, 1, 7, 6, 3, 5]
```

In [68]:

```
df_rank = pd.DataFrame({'Feature':Features.columns,'Rank':ranks})
df_rank
```

Out[68]:

	Feature	Rank
0	age	2
1	workclass	1
2	fnlwgt	8
3	education	4
4	education_num	1
5	marital_status	1
6	occupation	1
7	relationship	1
8	race	1
9	sex	1
10	capital_gain	7
11	capital_loss	6
12	hours_per_week	3
13	native_country	5

In [69]:

```
#most imporatnt features

imp= df_rank.Feature[df_rank.Rank == 1]
print("The important Features in the sample data are as follows :-\n",imp.values)
```

```
The important Features in the sample data are as follows :-
 ['workclass' 'education_num' 'marital_status' 'occupation' 'relationship'
 'race' 'sex']
```

In [70]:

```
selector =  RFECV(estimator=LogisticRegression(), step=1,cv=10)
selector.fit(Features,Labels)
ranks = selector.ranking_.tolist()
df_rank_cv = pd.DataFrame({'Feature':Features.columns,'Rank':ranks})
df_rank_cv
```

Out[70]:

	Feature	Rank
0	age	1
1	workclass	1
2	fnlwgt	2
3	education	1
4	education_num	1
5	marital_status	1
6	occupation	1
7	relationship	1
8	race	1
9	sex	1
10	capital_gain	1
11	capital_loss	1
12	hours_per_week	1
13	native_country	1

In [71]:

```
impcv= df_rank_cv.Feature[df_rank_cv.Rank == 1]
print("The important Features in the sample data after REFCV are as follows :-\n",impcv.val
```

```
The important Features in the sample data after REFCV are as follows :-
 ['age' 'workclass' 'education' 'education_num' 'marital_status'
 'occupation' 'relationship' 'race' 'sex' 'capital_gain' 'capital_loss'
 'hours_per_week' 'native_country']
```

Applying Decison Tree Classifier Model

In [72]:

```
#build the model
DT = DecisionTreeClassifier(random_state=0)
#train the model
DT.fit(X,Y)
#Model parameters study
Ypred = DT.predict(Xtest)
Ypred_proba = DT.predict_proba(Xtest)
# generate evaluation metrics
print("accuracy of Decision Tree Classifier :",metrics.accuracy_score(Ytest, Ypred))
#model_accuracy['Decision Tree Classifier'] = metrics.accuracy_score(Ytest, Ypred)
```

```
accuracy of Decision Tree Classifier : 0.8160432405871875
```

In [73]:

```
for depth in range(20):
    #build the model
    depth = depth + 1
    DT = DecisionTreeClassifier(max_depth=depth,random_state=0)
    #train the model
    DT.fit(X,Y)
    #Model parameters study
    Ypred = DT.predict(Xtest)
    Ypred_proba = DT.predict_proba(Xtest)
    # generate evaluation metrics
    print("accuracy of Decision Tree Classifier for max_depth ", depth," : ",metrics.accura

    #model_accuracy[auc] = metrics.roc_auc_score(Ytest,Ypred_proba[:,1])
```

```
accuracy of Decision Tree Classifier for max_depth  1  :  0.8049259873472145
accuracy of Decision Tree Classifier for max_depth  2  :  0.8049259873472145
accuracy of Decision Tree Classifier for max_depth  3  :  0.8228610036238561
accuracy of Decision Tree Classifier for max_depth  4  :  0.8442356120631411
accuracy of Decision Tree Classifier for max_depth  5  :  0.8447884036607088
accuracy of Decision Tree Classifier for max_depth  6  :  0.8527117498925127
accuracy of Decision Tree Classifier for max_depth  7  :  0.8540015969535041
accuracy of Decision Tree Classifier for max_depth  8  :  0.8554142865917327
accuracy of Decision Tree Classifier for max_depth  9  :  0.8575026104047663
accuracy of Decision Tree Classifier for max_depth  10  :  0.858608193599901
7
accuracy of Decision Tree Classifier for max_depth  11  :  0.856949818807198
6
accuracy of Decision Tree Classifier for max_depth  12  :  0.855782814323444
5
accuracy of Decision Tree Classifier for max_depth  13  :  0.853325962778699
1
accuracy of Decision Tree Classifier for max_depth  14  :  0.848289417111971
accuracy of Decision Tree Classifier for max_depth  15  :  0.844481297217615
7
accuracy of Decision Tree Classifier for max_depth  16  :  0.840488913457404
3
accuracy of Decision Tree Classifier for max_depth  17  :  0.838154904489896
2
accuracy of Decision Tree Classifier for max_depth  18  :  0.836128001965481
2
accuracy of Decision Tree Classifier for max_depth  19  :  0.831337141453227
7
accuracy of Decision Tree Classifier for max_depth  20  :  0.827221915115779
1
```

In [74]:

```
#since the model has best accuracy for max_depth 10 so retraining the model with max_depth
#build the model
DT = DecisionTreeClassifier(max_depth=10,random_state=0)
#train the model
DT.fit(X,Y)
#Model parameters study
Ypred = DT.predict(Xtest)
Ypred_proba = DT.predict_proba(Xtest)
# generate evaluation metrics
print("accuracy of Decision Tree Classifier :",metrics.accuracy_score(Ytest, Ypred))
model_accuracy['Accuracy Score of Decision Tree Classifier Model'] = metrics.accuracy_score
model_accuracy['AUC of Decision Tree Model Classifier - depth 10'] = metrics.roc_auc_score(
```

```
accuracy of Decision Tree Classifier : 0.8586081935999017
```

Evaluation of the Decision Tree Model trained with max_depth as 10

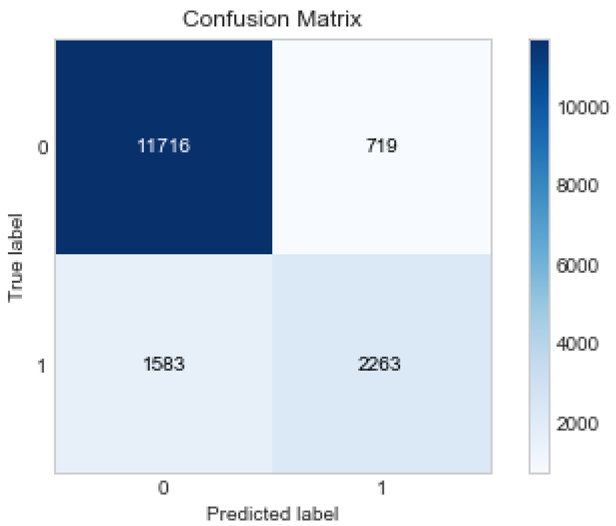
Confusion Matrix

In [75]:

```
scikitplot.metrics.plot_confusion_matrix(Ytest,Ypred)
```

Out[75]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xd2242b0>
```

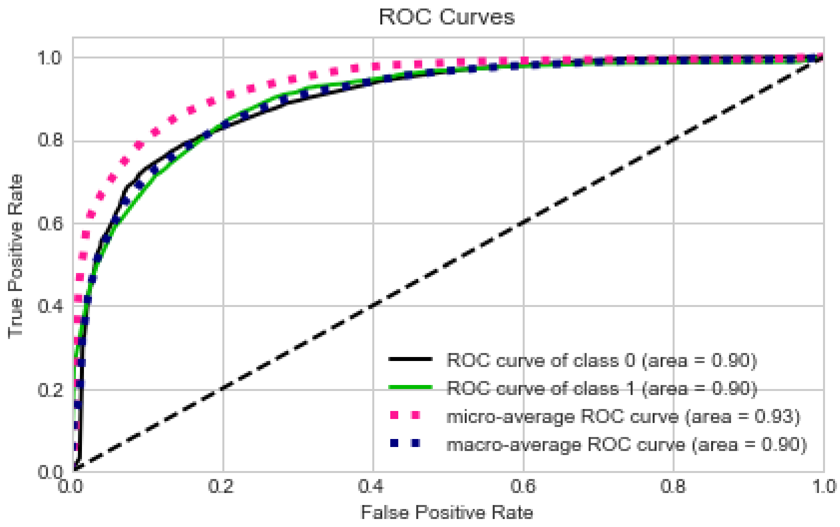


In [76]:

```
scikitplot.metrics.plot_roc(Ytest,Ypred_proba)
```

Out[76]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xc9d9518>
```

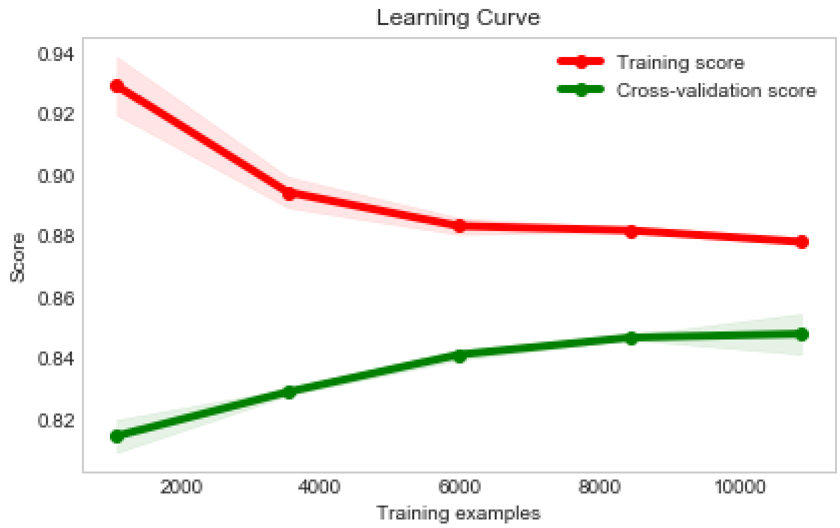


In [77]:

```
scikitplot.estimators.plot_learning_curve(DT,Xtest,Ytest)
```

Out[77]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xe0ed9e8>
```

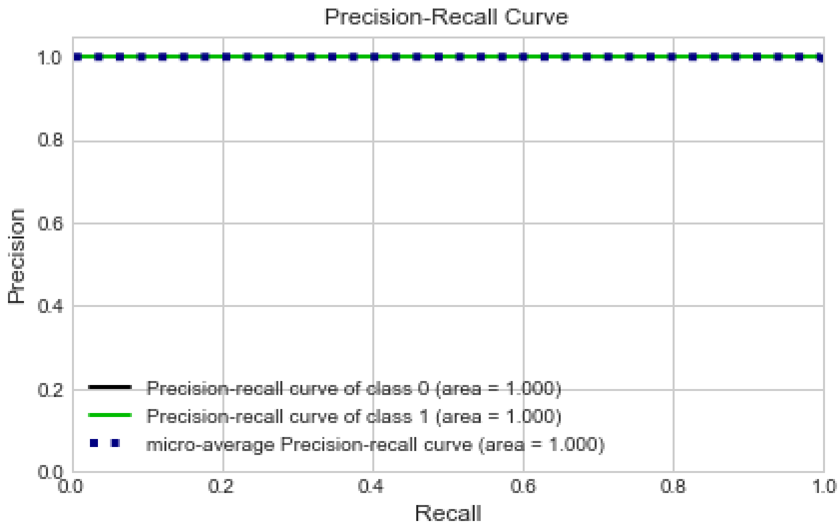


In [78]:

```
scikitplot.metrics.plot_precision_recall(Ypred,Ypred_proba)
```

Out[78]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xe136d68>
```



Applying 10 Fold cross validation to DEcision Tree Classifier Model

In [79]:

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(estimator= DecisionTreeClassifier(random_state=0),     # Model to
                X= Features,
                y = Labels,      # Target variable
                scoring = "accuracy",              # Scoring metric
                cv=10)                             # Cross validation folds

print("Accuracy per fold: ")
print("Cross Validation score: ", scores)
print("Average accuracy: ", scores.mean())
```

```
Accuracy per fold:
Cross Validation score:  [0.82006141 0.80900716 0.81453429 0.81576254 0.8200
6141 0.81900082
 0.81511057 0.82387876 0.80749539 0.80810977]
Average accuracy:  0.8153022124714788
```

In [80]:

```
for depth in range(20):
    depth = depth + 1
    scores = cross_val_score(estimator= DecisionTreeClassifier(max_depth=depth,random_state
                X= Features,
                y = Labels,      # Target variable
                scoring = "accuracy",             # Scoring metric
                cv=10)                            # Cross validation folds


    print("Average accuracy for max_depth",depth," : ", scores.mean())
```

```
Average accuracy for max_depth 1  :  0.8033863622153798
Average accuracy for max_depth 2  :  0.8033863622153798
Average accuracy for max_depth 3  :  0.8117204714601914
Average accuracy for max_depth 4  :  0.8436592070752982
Average accuracy for max_depth 5  :  0.8434339063910231
Average accuracy for max_depth 6  :  0.8506614044011134
Average accuracy for max_depth 7  :  0.8537938604476187
Average accuracy for max_depth 8  :  0.8550835897483091
Average accuracy for max_depth 9  :  0.8547149303740069
Average accuracy for max_depth 10  :  0.8565987329269996
Average accuracy for max_depth 11  :  0.8572333034676698
Average accuracy for max_depth 12  :  0.8572948165520635
Average accuracy for max_depth 13  :  0.8540190479381689
Average accuracy for max_depth 14  :  0.8534046631628607
Average accuracy for max_depth 15  :  0.8527495714842275
Average accuracy for max_depth 16  :  0.8486750003376944
Average accuracy for max_depth 17  :  0.8447236650264083
Average accuracy for max_depth 18  :  0.842430374615817
Average accuracy for max_depth 19  :  0.8374549780171325
Average accuracy for max_depth 20  :  0.8353666725643867
```

In [81]:

```
scores = cross_val_score(estimator= DecisionTreeClassifier(max_depth=12,random_state=0),
                X= Features,
                y = Labels,      # Target variable
                scoring = "accuracy",             # Scoring metric
                cv=10)                            # Cross validation folds

print("Accuracy per fold: ")
print("Cross Validation score: ", scores)
print("Average accuracy: ", scores.mean())
model_accuracy['10 CV Score-Decision Tree Classifier, max depth  12'] = scores.mean()
```

```
Accuracy per fold:
Cross Validation score:  [0.8493347  0.8616172  0.85301945 0.85875128 0.8661
2078 0.85892711
 0.85749386 0.86053656 0.85091133 0.85623592]
Average accuracy:  0.8572948165520635
```

Feature Selection using REFCV for Decision Tree Classifier

In [82]:

```
selector =  RFECV(estimator=DecisionTreeClassifier(max_depth=12,random_state=0), step=1,cv=
selector.fit(Features,Labels)
ranks = selector.ranking_.tolist()
df_rank_cv = pd.DataFrame({'Feature':Features.columns,'Rank':ranks})
df_rank_cv
```

Out[82]:

	Feature	Rank
0	age	2
1	workclass	6
2	fnlwgt	5
3	education	7
4	education_num	1
5	marital_status	1
6	occupation	4
7	relationship	10
8	race	11
9	sex	9
10	capital_gain	1
11	capital_loss	1
12	hours_per_week	3
13	native_country	8

In [83]:

```
impcvDT= df_rank_cv.Feature[df_rank_cv.Rank == 1]
print("The important Features in the sample data after REFCV are as follows :-\n",impcvDT.v
```

```
The important Features in the sample data after REFCV are as follows :-
 ['education_num' 'marital_status' 'capital_gain' 'capital_loss']
```

Applying K- Nearest Neighbor Model to sample Data

In [84]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [85]:

```
k = []
scores = []
errors = []
for K in range(30):
    K_value = K+1
    neigh = KNeighborsClassifier(n_neighbors = K_value, weights='uniform', algorithm='auto'
    neigh.fit(X,Y)
    y_pred = neigh.predict(Xtest)
    print("Accuracy is ", metrics.accuracy_score(Ytest,y_pred)*100,"% for K-Value:",K_value
    print("Error  is ", 100 - metrics.accuracy_score(Ytest,y_pred)*100,"% for K-Value:",K_v
    k.append(K_value)
    scores.append(metrics.accuracy_score(Ytest,y_pred)*100)
    errors.append(1 - metrics.accuracy_score(Ytest,y_pred) )
```

```
Accuracy is  72.7043793378785 % for K-Value: 1
Error  is  27.295620662121493 % for K-Value: 1
Accuracy is  78.57011240095817 % for K-Value: 2
Error  is  21.429887599041834 % for K-Value: 2
Accuracy is  75.98427615011363 % for K-Value: 3
Error  is  24.01572384988637 % for K-Value: 3
Accuracy is  78.92635587494625 % for K-Value: 4
Error  is  21.073644125053747 % for K-Value: 4
Accuracy is  77.5566611387507 % for K-Value: 5
Error  is  22.443338861249302 % for K-Value: 5
Accuracy is  79.25188870462502 % for K-Value: 6
Error  is  20.748111295374983 % for K-Value: 6
Accuracy is  78.44112769485903 % for K-Value: 7
Error  is  21.558872305140966 % for K-Value: 7
Accuracy is  79.51600024568516 % for K-Value: 8
Error  is  20.483999754314837 % for K-Value: 8
Accuracy is  78.99391929242675 % for K-Value: 9
Error  is  21.006080707573247 % for K-Value: 9
Accuracy is  79.80468030219274 % for K-Value: 10
Error  is  20.195319697807264 % for K-Value: 10
Accuracy is  79.48528960137584 % for K-Value: 11
Error  is  20.51471039862416 % for K-Value: 11
Accuracy is  79.98894416804865 % for K-Value: 12
Error  is  20.011055831951353 % for K-Value: 12
Accuracy is  79.71869049812665 % for K-Value: 13
Error  is  20.281309501873352 % for K-Value: 13
Accuracy is  79.99508629691051 % for K-Value: 14
Error  is  20.004913703089485 % for K-Value: 14
Accuracy is  79.81696455991647 % for K-Value: 15
Error  is  20.18303544008353 % for K-Value: 15
Accuracy is  80.16092377618082 % for K-Value: 16
Error  is  19.839076223819177 % for K-Value: 16
Accuracy is  80.08721822983847 % for K-Value: 17
Error  is  19.91278177016153 % for K-Value: 17
Accuracy is  80.16706590504269 % for K-Value: 18
Error  is  19.83293409495731 % for K-Value: 18
Accuracy is  80.17320803390456 % for K-Value: 19
Error  is  19.826791966095442 % for K-Value: 19
Accuracy is  80.30219274000369 % for K-Value: 20
Error  is  19.69780725999631 % for K-Value: 20
Accuracy is  80.22848719366132 % for K-Value: 21
Error  is  19.77151280633867 7 % for K-Value: 21
Accuracy is  80.23462932252319 % for K-Value: 22
Error  is  19.76537067747681 % for K-Value: 22
Accuracy is  80.21006080707573 % for K-Value: 23
```

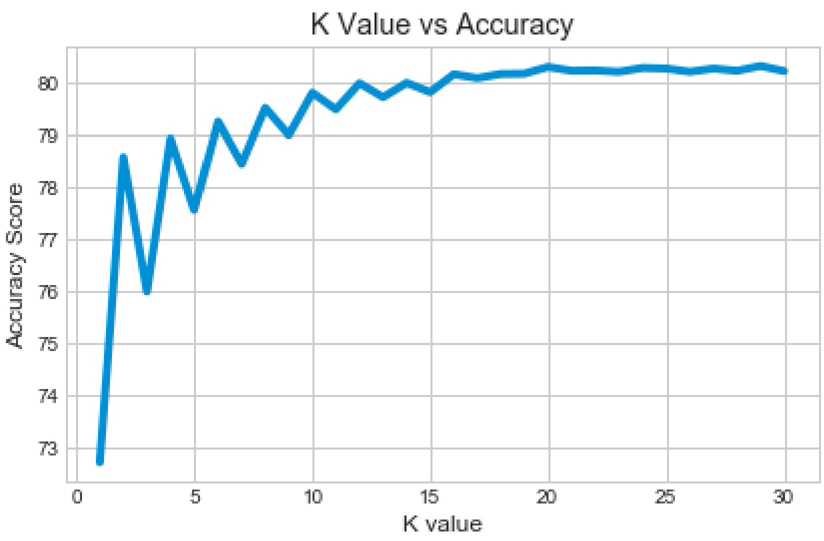
```
Error  is  19.789939192924265 % for K-Value: 23
Accuracy is  80.2837663534181 % for K-Value: 24
Error  is  19.716233646581898 % for K-Value: 24
Accuracy is  80.27148209569437 % for K-Value: 25
Error  is  19.728517904305633 % for K-Value: 25
Accuracy is  80.21006080707573 % for K-Value: 26
Error  is  19.789939192924265 % for K-Value: 26
Accuracy is  80.27148209569437 % for K-Value: 27
Error  is  19.728517904305633 % for K-Value: 27
Accuracy is  80.22848719366132 % for K-Value: 28
Error  is  19.771512806338677 % for K-Value: 28
Accuracy is  80.32061912658928 % for K-Value: 29
Error  is  19.67938087341072 % for K-Value: 29
Accuracy is  80.22234506479946 % for K-Value: 30
Error  is  19.777654935200545 % for K-Value: 30
```

In [86]:

```
plt.plot(k,scores)
plt.xlabel('K value')
plt.ylabel('Accuracy Score')
plt.title('K Value vs Accuracy')
```

Out[86]:

```
Text(0.5,1,'K Value vs Accuracy')
```

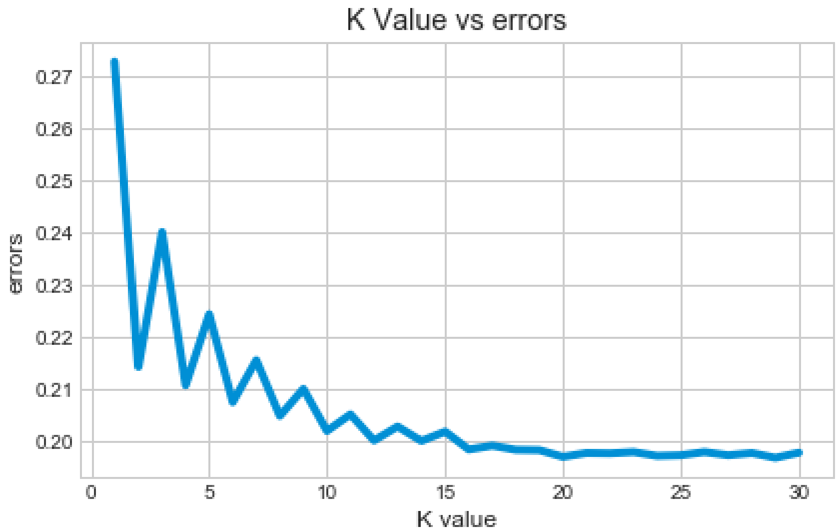


In [87]:

```
plt.plot(k,errors)
plt.xlabel('K value')
plt.ylabel('errors')
plt.title('K Value vs errors')
```

Out[87]:

```
Text(0.5,1,'K Value vs errors')
```



In [88]:

```
knn = KNeighborsClassifier(n_neighbors = 20, weights='uniform', algorithm='auto')
knn.fit(X,Y)
Ypred = knn.predict(Xtest)
Ypred_proba = knn.predict_proba(Xtest)
# generate evaluation metrics
print("accuracy of KNN Classifier :",metrics.accuracy_score(Ytest, Ypred))
model_accuracy['Accuracy Score of KNN Classifier neigbors-20'] = metrics.accuracy_score(Yte

model_accuracy['AUC of KNN Classifier neighbors-20'] = metrics.roc_auc_score(Ytest,Ypred_pr
```

```
accuracy of KNN Classifier : 0.8030219274000369
```

Evaluation of K Nearest Neighbor for K = 20

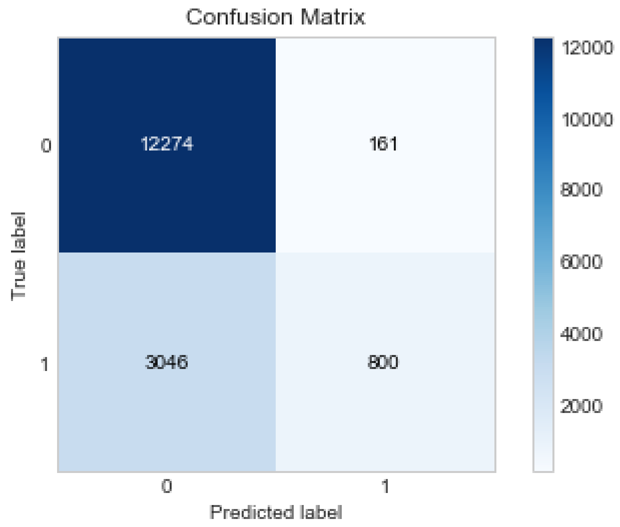
Confusion Matrix

In [89]:

```
scikitplot.metrics.plot_confusion_matrix(Ytest,Ypred)
```

Out[89]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xe0130f0>
```



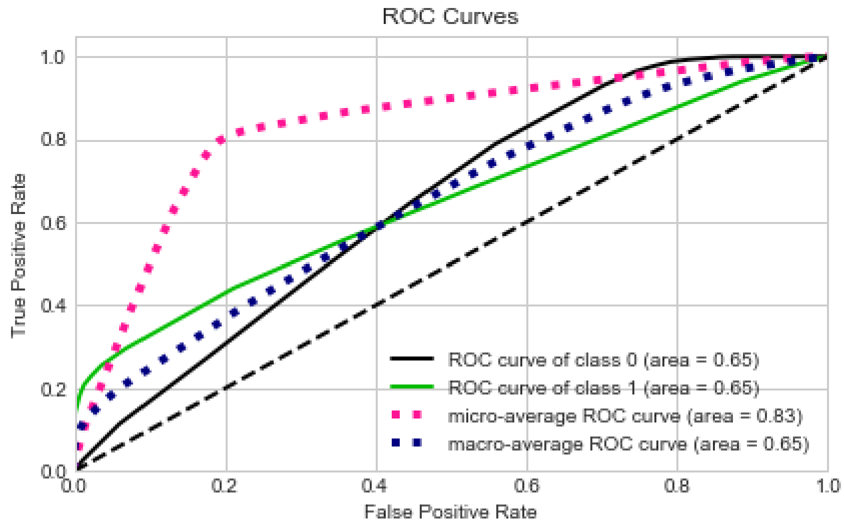
Receiver operating characteristic curve

In [90]:

```
scikitplot.metrics.plot_roc(Ytest,Ypred_proba)
```

Out[90]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xe01f4e0>
```



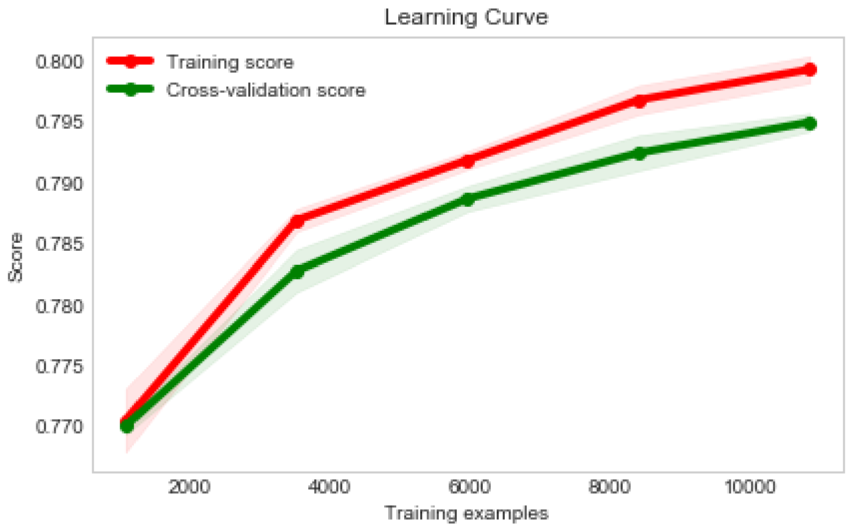
Learning Curve of KNN Classifier Model

In [91]:

```
scikitplot.estimators.plot_learning_curve(knn,Xtest,Ytest)
```

Out[91]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xcb785c0>
```



10 Fold Cross Validation applied to K nearest neighbor model

In [92]:

```
from sklearn.cross_validation import cross_val_score

scores = cross_val_score(estimator= KNeighborsClassifier(n_neighbors = 20, weights='uniform
                X= Features,
                y = Labels,      # Target variable
                scoring = "accuracy",               # Scoring metric
                cv=10)                              # Cross validation folds

print("Accuracy per fold: ")
print("Cross Validation score: ", scores)
print("Average accuracy: ", scores.mean())
model_accuracy['10 CV Score-KNN Classifier neighbors-20'] = scores.mean()
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41:
DeprecationWarning: This module was deprecated in version 0.18 in favor of t
he model_selection module into which all the refactored classes and function
s are moved. Also note that the interface of the new CV iterators are differ
ent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)

Accuracy per fold:
Cross Validation score:  [0.80143296 0.80225179 0.80429887 0.80266121 0.7995
9058 0.7993448
 0.8046683  0.80032767 0.79746058 0.79930371]
Average accuracy:  0.8011340470216448
```

Ensemble model bagging technique

Bagging with Logistic Regression

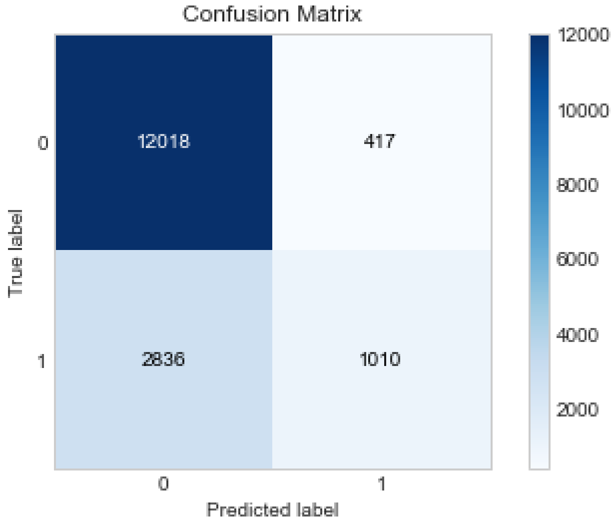
In [93]:

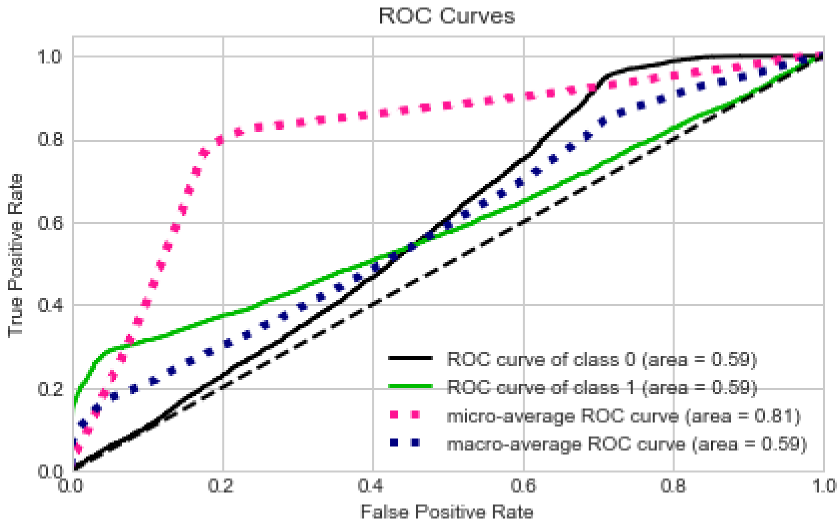
```
from sklearn.ensemble import BaggingClassifier
import scikitplot
bag_LR = BaggingClassifier(LogisticRegression(),
                            n_estimators=10, max_samples=0.5,
                            bootstrap=True, random_state=3)

bag_LR.fit(X,Y)

#### Predictions by the Bagging Ensemble model

bag_preds = bag_LR.predict(Xtest)
print("Predictions : ",bag_preds)

bag_preds_proba = bag_LR.predict_proba(Xtest)
print("Prediction Probabilities : ",bag_preds_proba)

#### Score of the bagging ensemble model

bag_LR.score(Xtest,Ytest)

print("Accuracy Score of Bagging for single Logistic Regression Model :",metrics.accuracy_s

#### Confusion Matrix

scikitplot.metrics.plot_confusion_matrix(Ytest,bag_preds)

#### ROC

scikitplot.metrics.plot_roc(Ytest,bag_preds_proba)
model_accuracy['Accuracy Score-Bagging-Logistic Regression'] = metrics.accuracy_score(Ytest
model_accuracy['AUC-Bagging-Logistic Regression'] = metrics.roc_auc_score(Ytest,bag_preds_p

scikitplot.estimators.plot_learning_curve(bag_LR,Xtest,Ytest)
```

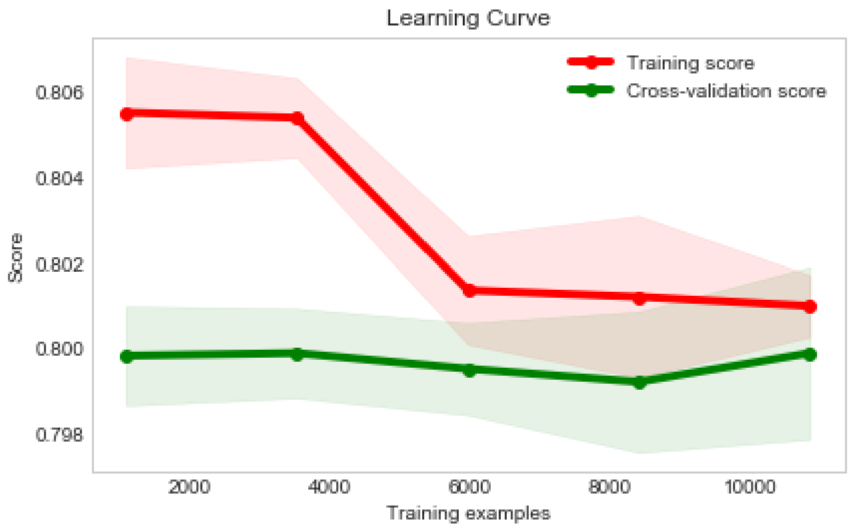
```
Predictions :  [0 0 0 ... 0 1 0]
Prediction Probabilities :  [[0.80520228 0.19479772]
 [0.72535378 0.27464622]
 [0.87297754 0.12702246]
 ...
 [0.88734883 0.11265117]
 [0.27170902 0.72829098]
 [0.79276083 0.20723917]]
Accuracy Score of Bagging for single Logistic Regression Model : 0.800196548
1235797
```

Out[93]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x545cf28>
```







Bagging with KNN Model

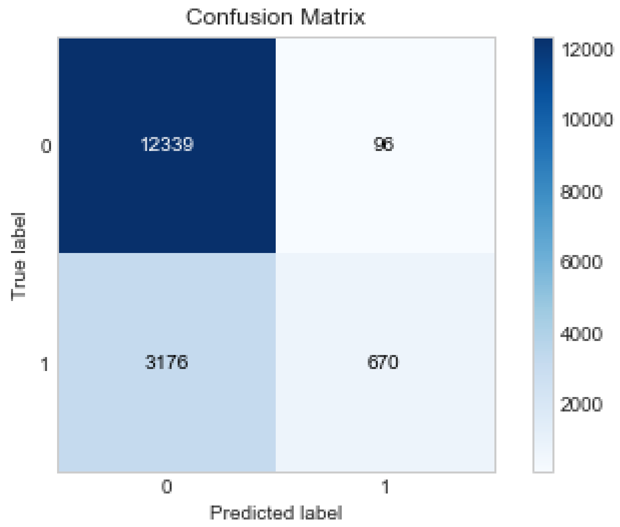
In [94]:

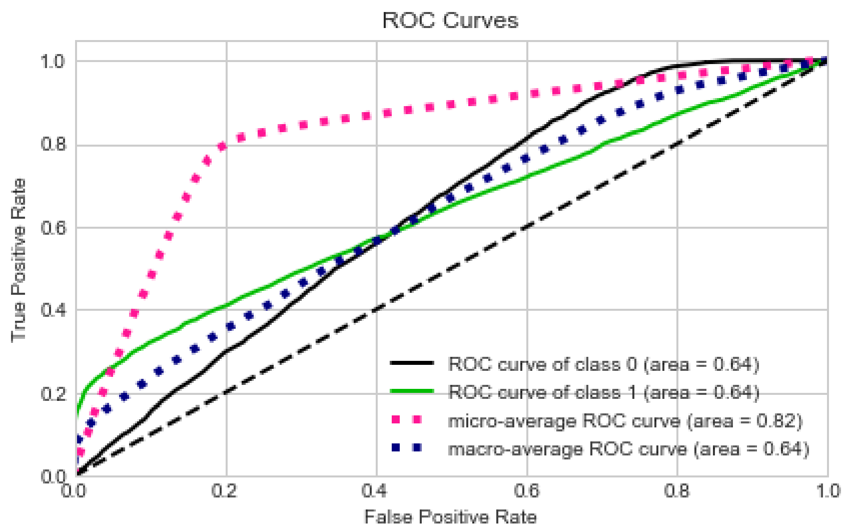
```
from sklearn.ensemble import BaggingClassifier
bag_KNN = BaggingClassifier(KNeighborsClassifier(n_neighbors = 20, weights='uniform', algor
                            n_estimators=10, max_samples=0.5,
                            bootstrap=True, random_state=3)

bag_KNN.fit(X,Y)

#### Predictions by the Bagging Ensemble model

bag_preds = bag_KNN.predict(Xtest)
print("Predictions : ",bag_preds)

bag_preds_proba = bag_KNN.predict_proba(Xtest)
print("Prediction Probabilities : ",bag_preds_proba)

#### Score of the bagging ensemble model

bag_KNN.score(Xtest,Ytest)

print("Accuracy Score of Bagging for single KNN Model :",metrics.accuracy_score(Ytest,bag_p

#### Confusion Matrix

scikitplot.metrics.plot_confusion_matrix(Ytest,bag_preds)

#### ROC

scikitplot.metrics.plot_roc(Ytest,bag_preds_proba)
model_accuracy['Accuracy Score-Bagging-KNN neighbors -20'] = metrics.accuracy_score(Ytest,b
model_accuracy['AUC-Bagging-KNN neighbors -20'] = metrics.roc_auc_score(Ytest,bag_preds_pro
scikitplot.estimators.plot_learning_curve(bag_KNN,Xtest,Ytest)
```

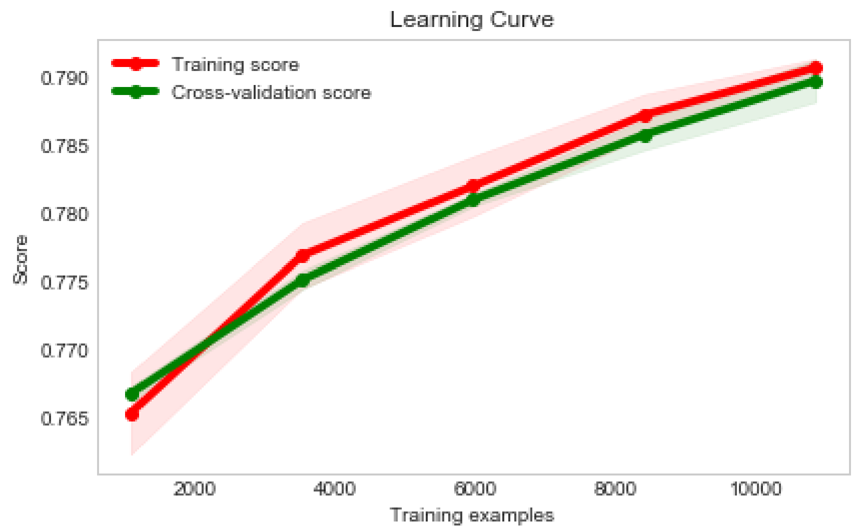
```
Predictions :  [0 0 0 ... 0 0 0]
Prediction Probabilities :  [[0.675 0.325]
 [0.73  0.27 ]
 [0.855 0.145]
 ...
 [0.815 0.185]
 [0.53  0.47 ]
 [0.865 0.135]]
Accuracy Score of Bagging for single KNN Model : 0.7990295436398256
```

Out[94]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xb735198>
```







RANDOM FOREST CLASSIFIER model

In [95]:

```
from sklearn.ensemble import  RandomForestClassifier
RF = RandomForestClassifier()
RF.fit(X,Y)
Ypred = RF.predict(Xtest)
Ypred_proba = RF.predict_proba(Xtest)
print("accuracy of Random Forest Classifier :",metrics.accuracy_score(Ytest, Ypred))
model_accuracy['Accuracy score of Random Forest Classifier'] = metrics.accuracy_score(Ytest
```

```
accuracy of Random Forest Classifier : 0.8477366255144033
```

Evaluate the Random Forest Model

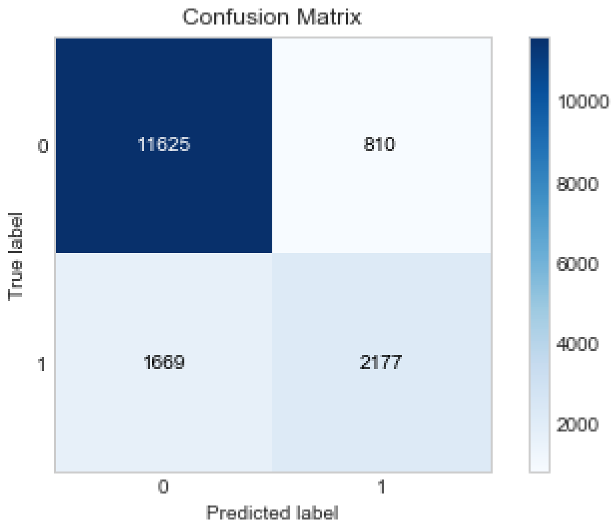
Confusion Matrix

In [96]:

```
scikitplot.metrics.plot_confusion_matrix(Ytest,Ypred)
```

Out[96]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x54e1160>
```



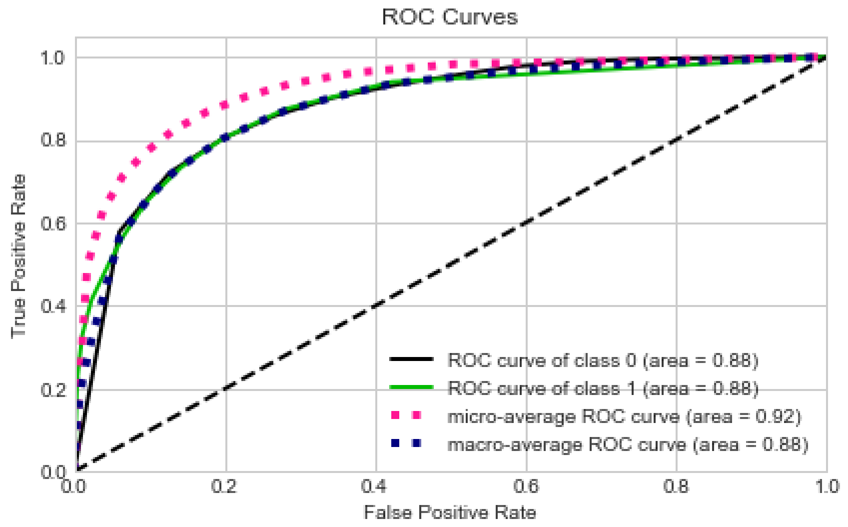
Reciver Operating Characteristic Curve for Random Forest Model

In [97]:

```
scikitplot.metrics.plot_roc(Ytest,Ypred_proba)
```

Out[97]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xbaab978>
```



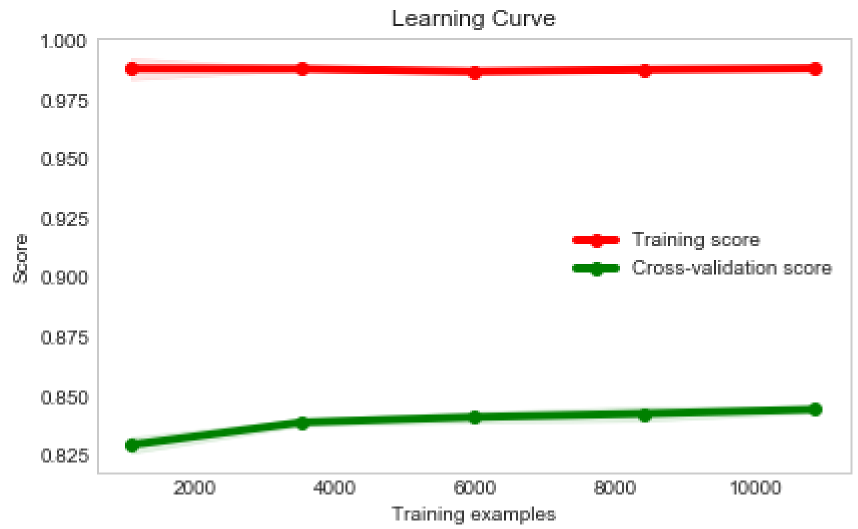
Learning Curve

In [98]:

```
scikitplot.estimators.plot_learning_curve(RF,Xtest,Ytest)
```

Out[98]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xbebcfd0>
```



In [99]:

```
print("AUC for Random Forest Classifier : ",metrics.roc_auc_score(Ytest,Ypred_proba[:,1]))
model_accuracy['AUC for Random Forest Classifier'] = metrics.roc_auc_score(Ytest,Ypred_prob
```

```
AUC for Random Forest Classifier :  0.879345566263342
```

Feature Selection using feature_importances_ parameter of Random Forest Model

In [100]:

```
RF.fit(Xtest, Ytest)
print("Features sorted by their score:")
print(sorted(zip(map(lambda x: round(x, 4), RF.feature_importances_), Features.columns),rev
```

```
Features sorted by their score:
[(0.1603, 'fnlwgt'), (0.1406, 'age'), (0.1364, 'capital_gain'), (0.0933, 'ma
rital_status'), (0.0907, 'hours_per_week'), (0.0852, 'education_num'), (0.07
45, 'occupation'), (0.0643, 'relationship'), (0.0411, 'workclass'), (0.0392,
'capital_loss'), (0.0284, 'education'), (0.0184, 'native_country'), (0.0147,
'race'), (0.0129, 'sex')]
```

Feature Selection using RFECV - Recursive Feature Elimination Using Cross Validation

In [101]:

```
selector =  RFECV(estimator=RandomForestClassifier(), step=1,cv=10)
selector.fit(Features,Labels)
ranks = selector.ranking_.tolist()
df_rank_cv = pd.DataFrame({'Feature':Features.columns,'Rank':ranks})
df_rank_cv
```

Out[101]:

	Feature	Rank
0	age	1
1	workclass	1
2	fnlwgt	1
3	education	2
4	education_num	1
5	marital_status	1
6	occupation	1
7	relationship	1
8	race	5
9	sex	4
10	capital_gain	1
11	capital_loss	1
12	hours_per_week	1
13	native_country	3

In [102]:

```
impcvRF= df_rank_cv.Feature[df_rank_cv.Rank == 1]
print("The important Features in the sample data after REFCV are as follows :-\n",impcvRF.v
```

```
The important Features in the sample data after REFCV are as follows :-
 ['age' 'workclass' 'fnlwgt' 'education_num' 'marital_status' 'occupation'
 'relationship' 'capital_gain' 'capital_loss' 'hours_per_week']
```

10 Fold Cross Validation for Random Forest Classifier

In [103]:

```
from sklearn.cross_validation import cross_val_score

scores = cross_val_score(estimator= RandomForestClassifier(),     # Model to test
                X= Features,
                y = Labels,      # Target variable
                scoring = "accuracy",              # Scoring metric
                cv=10)                             # Cross validation folds

print("Accuracy per fold: ")
print("Cross Validation score: ", scores)
print("Average accuracy: ", scores.mean())
model_accuracy['10 CV Score-Random Forest Classifier'] = scores.mean()
```

```
Accuracy per fold:
Cross Validation score:  [0.84360287 0.85711361 0.84646878 0.85220061 0.8522
0061 0.85503686
 0.85176085 0.85132091 0.84579152 0.84333402]
Average accuracy:  0.8498830642906963
```

Using Boosting Method of Ensemble model to predict the annual income

In [104]:

```
from xgboost.sklearn import XGBClassifier
#set the parameters for the xgbosst model
params = {
    'objective': 'binary:logistic',
    'max_depth': 2,
    'learning_rate': 1.0,
    'silent': 1.0,
    'n_estimators': 5
}
params['eval_metric'] = ['logloss', 'auc']
```

Train the XGBClassifier model

In [116]:

```
bst = XGBClassifier(**params).fit(X,Y)
```

Predict the annual income

In [106]:

```
preds = bst.predict(Xtest)
preds
```

```
C:\Users\santhu\Anaconda3\lib\site-packages\sklearn\preprocessing\label.py:1
51: DeprecationWarning: The truth value of an empty array is ambiguous. Retu
rning False, but in future this will result in an error. Use `array.size > 0
` to check that an array is not empty.
  if diff:
```

Out[106]:

```
array([0, 0, 0, ..., 1, 0, 1], dtype=int64)
```

In [107]:

```
preds_proba = bst.predict_proba(Xtest)
preds_proba
```

Out[107]:

```
array([[0.9862895 , 0.01371049],
       [0.6448917 , 0.35510832],
       [0.8749048 , 0.1250952 ],
       ...,
       [0.28199995, 0.71800005],
       [0.71667016, 0.28332984],
       [0.17598617, 0.8240138 ]], dtype=float32)
```

Measure the accuracy of the model

In [108]:

```
correct = 0
from sklearn.metrics import accuracy_score
for i in range(len(preds)):
    if (y_test[i] == preds[i]):
        correct += 1

acc = accuracy_score(Ytest, preds)

print('Predicted correctly: {0}/{1}'.format(correct, len(preds)))
print('Accuracy Score :{:.4f}'.format(acc))
print('Error: {0:.4f}'.format(1-acc))
model_accuracy['Accuracy Score of XGBOOST Model'] = acc
```

```
Predicted correctly: 13897/16281
Accuracy Score :0.8536
Error: 0.1464
```

In [109]:

```
from sklearn.metrics import classification_report
print(classification_report(Ytest,preds))
```

```
             precision    recall  f1-score   support

          0       0.87      0.95      0.91     12435
          1       0.76      0.55      0.64      3846

avg / total       0.85      0.85      0.85     16281
```

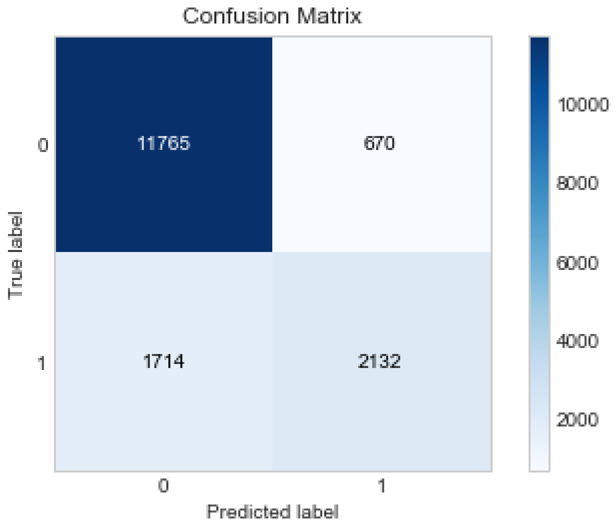
Confusion Matrix

In [110]:

```
import scikitplot
scikitplot.metrics.plot_confusion_matrix(Ytest, preds)
```

Out[110]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xc2fa358>
```



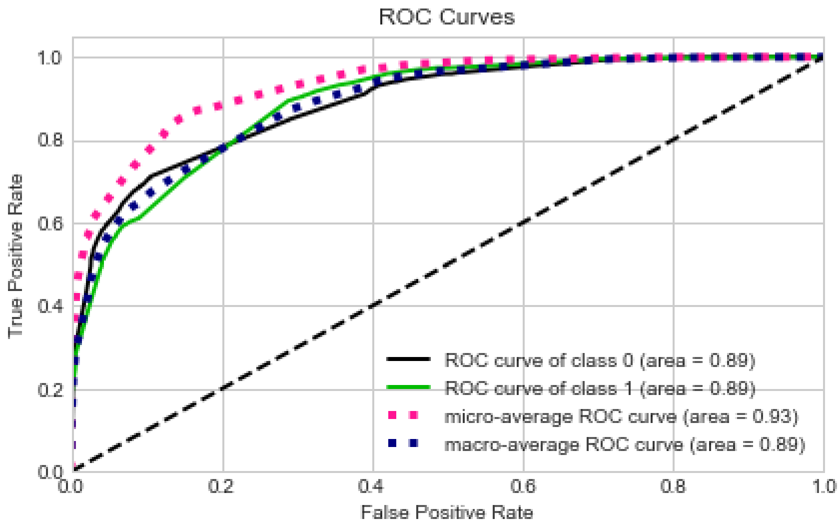
ROC

In [111]:

```
scikitplot.metrics.plot_roc(Ytest,preds_proba)
```

Out[111]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xd27ea20>
```



In [117]:

```
#### Learning Curve
scikitplot.estimators.plot_learning_curve(bst,Xtest,Ytest)
```

```
ze > 0` to check that an array is not empty.
  if diff:
C:\Users\santhu\Anaconda3\lib\site-packages\sklearn\preprocessing\label.p
y:151: DeprecationWarning: The truth value of an empty array is ambiguous.
Returning False, but in future this will result in an error. Use `array.si
ze > 0` to check that an array is not empty.
  if diff:
C:\Users\santhu\Anaconda3\lib\site-packages\sklearn\preprocessing\label.p
y:151: DeprecationWarning: The truth value of an empty array is ambiguous.
Returning False, but in future this will result in an error. Use `array.si
ze > 0` to check that an array is not empty.
  if diff:
C:\Users\santhu\Anaconda3\lib\site-packages\sklearn\preprocessing\label.p
y:151: DeprecationWarning: The truth value of an empty array is ambiguous.
Returning False, but in future this will result in an error. Use `array.si
ze > 0` to check that an array is not empty.
  if diff:
C:\Users\santhu\Anaconda3\lib\site-packages\sklearn\preprocessing\label.p
y:151: DeprecationWarning: The truth value of an empty array is ambiguous.
Returning False, but in future this will result in an error. Use `array.si
```

In [113]:

```
print('AUC for XGBOOST model : ',metrics.roc_auc_score(Ytest,preds_proba[:,1]))
model_accuracy['AUC for XGBOOST model'] = metrics.roc_auc_score(Ytest,preds_proba[:,1])
```

```
AUC for XGBOOST model :  0.8896135620253921
```

In [114]:

```
features = []
scores = []
for k,v in model_accuracy.items():
    features.append(k)
    scores.append(v)
```

In [115]:

```
df_scores = pd.DataFrame({'Features':features,'Scores':scores})
feat_cols = ['Features','Scores']
df_scores = df_scores[feat_cols]
#df_scores
```

Out[115]:

	Features	Scores
0	Logistic Regression	0.800258
1	AUC_Logistic_Regression	0.611375
2	10 CV Score-Logistic Regression	0.798882
3	Accuracy Score of Decision Tree Classifier Model	0.858608
4	AUC of Decision Tree Model Classifier - depth 10	0.897965
5	10 CV Score-Decision Tree Classifier, max dept...	0.857295
6	Accuracy Score of KNN Classifier neigbors-20	0.803022
7	AUC of KNN Classifier neighbors-20	0.648625
8	10 CV Score-KNN Classifier neighbors-20	0.801134
9	Accuracy Score-Bagging-Logistic Regression	0.800197
10	AUC-Bagging-Logistic Regression	0.593131
11	Accuracy Score-Bagging-KNN neighbors -20	0.799030
12	AUC-Bagging-KNN neighbors -20	0.635693
13	Accuracy score of Random Forest Classifier	0.847737
14	AUC for Random Forest Classifier	0.879346
15	10 CV Score-Random Forest Classifier	0.849883
16	Accuracy Score of XGBOOST Model	0.853572
17	AUC for XGBOOST model	0.889614

CONCLUSION

From the above Model Estimation Score dataframe if we take into consideration the AUC value then it is evident that the Decision Tree Classifier Model and XGBOOST ensemble model have the highest accuracy for the model performance. .

AUC for XGBOOST model is 88.96%