This data was extracted from the census bureau database found at
http://www.census.gov/ftp/pub/DES/www/welcome.html (http://www.census.gov/ftp/pub/DES/www/welcome.html)
Donor: Ronny Kohavi and Barry Becker, Data Mining and Visualization Silicon Graphics. e-mail: ronnyk@sgi.com for questions. Split into train-test using MLC++ GenCVFiles (2/3, 1/3 random). 48842 instances, mix of continuous and discrete (train=32561, test=16281) 45222 if instances with unknown values are removed (train=30162, test=15060) Duplicate or conflicting instances : 6 Class probabilities for adult.all file Probability for the label '>50K' : 23.93% / 24.78% (without unknowns) Probability for the label '<=50K' : 76.07% / 75.22% (without unknowns)

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)) Prediction task is to determine whether a person makes over 50K a year. Conversion of original data as follows:

1. Discretized a gross income into two ranges with threshold 50,000.
2. Convert U.S. to US to avoid periods.
3. Convert Unknown to "?"
4. Run MLC++ GenCVFiles to generate data,test. Description of fnlwgt (final weight) The weights on the CPS files are controlled to independent estimates of the civilian noninstitutional population of the US. These are prepared monthly for us by Population Division here at the Census Bureau. We use 3 sets of controls. These are:
5. A single cell estimate of the population 16+ for each state.
6. Controls for Hispanic Origin by age and sex.
7. Controls by Race, age and sex. We use all three sets of controls in our weighting program and "rake" through them 6 times so that by the end we come back to all the controls we used. The term estimate refers to population totals derived from CPS by creating "weighted tallies" of any specified socio-economic characteristics of the population. People with similar demographic characteristics should have similar weights. There is one important caveat to remember about this statement. That is that since the CPS sample is actually a collection of 51 state samples, each with its own probability of selection, the statement only applies within state.

Dataset Link

https://archive.ics.uci.edu/ml/machine-learning-databases/adult/ (https://archive.ics.uci.edu/ml/machine-learning-databases/adult/)

Problem 1: Prediction task is to determine whether a person makes over 50K a year.

Problem 2: Which factors are important

Problem 3: Which algorithms are best for this dataset

In [1]:
```
#code to load required libraries and data:
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
# Machine learning
import sklearn.ensemble as ske
from sklearn import datasets, model_selection, tree, preprocessing, metrics, l
inear_model
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifie
r
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LinearRegression, LogisticRegression, Ridge,
Lasso
from sklearn.tree import DecisionTreeClassifier
# Grid and Random Search
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
# Metrics
from sklearn.metrics import precision_recall_fscore_support, roc_curve, auc
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Load training and test data into corresponding dataframes

train_set=pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-database
s/adult/adult.data', header = None,na_values="?")

test_set=pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-database
s/adult/adult.test', skiprows = 1, header = None,na_values="?")

col_labels = ['age', 'workclass', 'fnlwgt', 'education', 'education_num', 'mar
ital_status',
'occupation','relationship', 'race', 'sex', 'capital_gain', 'capital_loss', 'h
ours_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))
```

In [4]:
```
# View training and test data sample

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

Out[4]:

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

In [5]:
```
test_set.sample(4, random_state = 42)
```

Out[5]:

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

In [7]:
```
# Check for null values if any in training and test data sets

train_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]:
```
test_set.isnull().sum()
```

Out[8]:
```
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 [9]:
```
pd.DataFrame([train_set.dtypes, test_set.dtypes], index = ['train_set','test_set']).T
```

Out[9]:

	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

In [10]:
```
# Finding the columns having data types as object

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 [11]:
```
train_set.workclass.value_counts()
```

Out[11]:
```
 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 [12]:
```
train_set.native_country.value_counts()
```

Out[12]:
```
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
France                           29
Greece                           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 [13]:
```
train_set
```

Out[13]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationsh
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-fam
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husba
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-fam
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husba
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	W
5	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	W
6	49	Private	160187	9th	5	Married-spouse-absent	Other-service	Not-in-fam
7	52	Self-emp-not-inc	209642	HS-grad	9	Married-civ-spouse	Exec-managerial	Husba
8	31	Private	45781	Masters	14	Never-married	Prof-specialty	Not-in-fam
9	42	Private	159449	Bachelors	13	Married-civ-spouse	Exec-managerial	Husba
10	37	Private	280464	Some-college	10	Married-civ-spouse	Exec-managerial	Husba
11	30	State-gov	141297	Bachelors	13	Married-civ-spouse	Prof-specialty	Husba
12	23	Private	122272	Bachelors	13	Never-married	Adm-clerical	Own-ch
13	32	Private	205019	Assoc-acdm	12	Never-married	Sales	Not-in-fam
14	40	Private	121772	Assoc-voc	11	Married-civ-spouse	Craft-repair	Husba
15	34	Private	245487	7th-8th	4	Married-civ-spouse	Transport-moving	Husba
16	25	Self-emp-not-inc	176756	HS-grad	9	Never-married	Farming-fishing	Own-ch
17	32	Private	186824	HS-grad	9	Never-married	Machine-op-inspct	Unmarri
18	38	Private	28887	11th	7	Married-civ-spouse	Sales	Husba
19	43	Self-emp-not-inc	292175	Masters	14	Divorced	Exec-managerial	Unmarri
20	40	Private	193524	Doctorate	16	Married-civ-spouse	Prof-specialty	Husba
21	54	Private	302146	HS-grad	9	Separated	Other-service	Unmarri

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationsh
22	35	Federal-gov	76845	9th	5	Married-civ-spouse	Farming-fishing	Husba
23	43	Private	117037	11th	7	Married-civ-spouse	Transport-moving	Husba
24	59	Private	109015	HS-grad	9	Divorced	Tech-support	Unmarri
25	56	Local-gov	216851	Bachelors	13	Married-civ-spouse	Tech-support	Husba
26	19	Private	168294	HS-grad	9	Never-married	Craft-repair	Own-ch
27	54	?	180211	Some-college	10	Married-civ-spouse	?	Husba
28	39	Private	367260	HS-grad	9	Divorced	Exec-managerial	Not-in-fam
29	49	Private	193366	HS-grad	9	Married-civ-spouse	Craft-repair	Husba
...	...	...	...	...	...	...	...	
32531	30	?	33811	Bachelors	13	Never-married	?	Not-in-fam
32532	34	Private	204461	Doctorate	16	Married-civ-spouse	Prof-specialty	Husba
32533	54	Private	337992	Bachelors	13	Married-civ-spouse	Exec-managerial	Husba
32534	37	Private	179137	Some-college	10	Divorced	Adm-clerical	Unmarri
32535	22	Private	325033	12th	8	Never-married	Protective-serv	Own-ch
32536	34	Private	160216	Bachelors	13	Never-married	Exec-managerial	Not-in-fam
32537	30	Private	345898	HS-grad	9	Never-married	Craft-repair	Not-in-fam
32538	38	Private	139180	Bachelors	13	Divorced	Prof-specialty	Unmarri
32539	71	?	287372	Doctorate	16	Married-civ-spouse	?	Husba
32540	45	State-gov	252208	HS-grad	9	Separated	Adm-clerical	Own-ch
32541	41	?	202822	HS-grad	9	Separated	?	Not-in-fam
32542	72	?	129912	HS-grad	9	Married-civ-spouse	?	Husba
32543	45	Local-gov	119199	Assoc-acdm	12	Divorced	Prof-specialty	Unmarri
32544	31	Private	199655	Masters	14	Divorced	Other-service	Not-in-fam
32545	39	Local-gov	111499	Assoc-acdm	12	Married-civ-spouse	Adm-clerical	W

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationsh
32546	37	Private	198216	Assoc-acdm	12	Divorced	Tech-support	Not-in-fam
32547	43	Private	260761	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husba
32548	65	Self-emp-not-inc	99359	Prof-school	15	Never-married	Prof-specialty	Not-in-fam
32549	43	State-gov	255835	Some-college	10	Divorced	Adm-clerical	Oth relat
32550	43	Self-emp-not-inc	27242	Some-college	10	Married-civ-spouse	Craft-repair	Husba
32551	32	Private	34066	10th	6	Married-civ-spouse	Handlers-cleaners	Husba
32552	43	Private	84661	Assoc-voc	11	Married-civ-spouse	Sales	Husba
32553	32	Private	116138	Masters	14	Never-married	Tech-support	Not-in-fam
32554	53	Private	321865	Masters	14	Married-civ-spouse	Exec-managerial	Husba
32555	22	Private	310152	Some-college	10	Never-married	Protective-serv	Not-in-fam
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	W
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husba
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarri
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-ch
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	W

32561 rows × 15 columns

In [14]:
```
train_set.relationship.value_counts()
```

Out[14]:
```
 Husband           13193
 Not-in-family      8305
 Own-child          5068
 Unmarried          3446
 Wife               1568
 Other-relative      981
Name: relationship, dtype: int64
```

In [15]:
```
# Unique counts for the features

train_set.workclass.nunique(),train_set.education.nunique(),train_set.marital_
status.nunique(),train_set.native_country.nunique()
```

Out[15]:
```
(9, 16, 7, 42)
```

In [16]:
```
# Concatenate training datasets and test datasets into a common dataframe Samp
le
X_train = train_set.copy()
X_test = test_set.copy()
```

In [17]:
```
X_train.columns
```

Out[17]:
```
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 [18]:
```
Sample = X_train.append(X_test)
```

In [19]:
```
# Summary Statistics of Continuous Values

Sample.describe()
```

Out[19]:

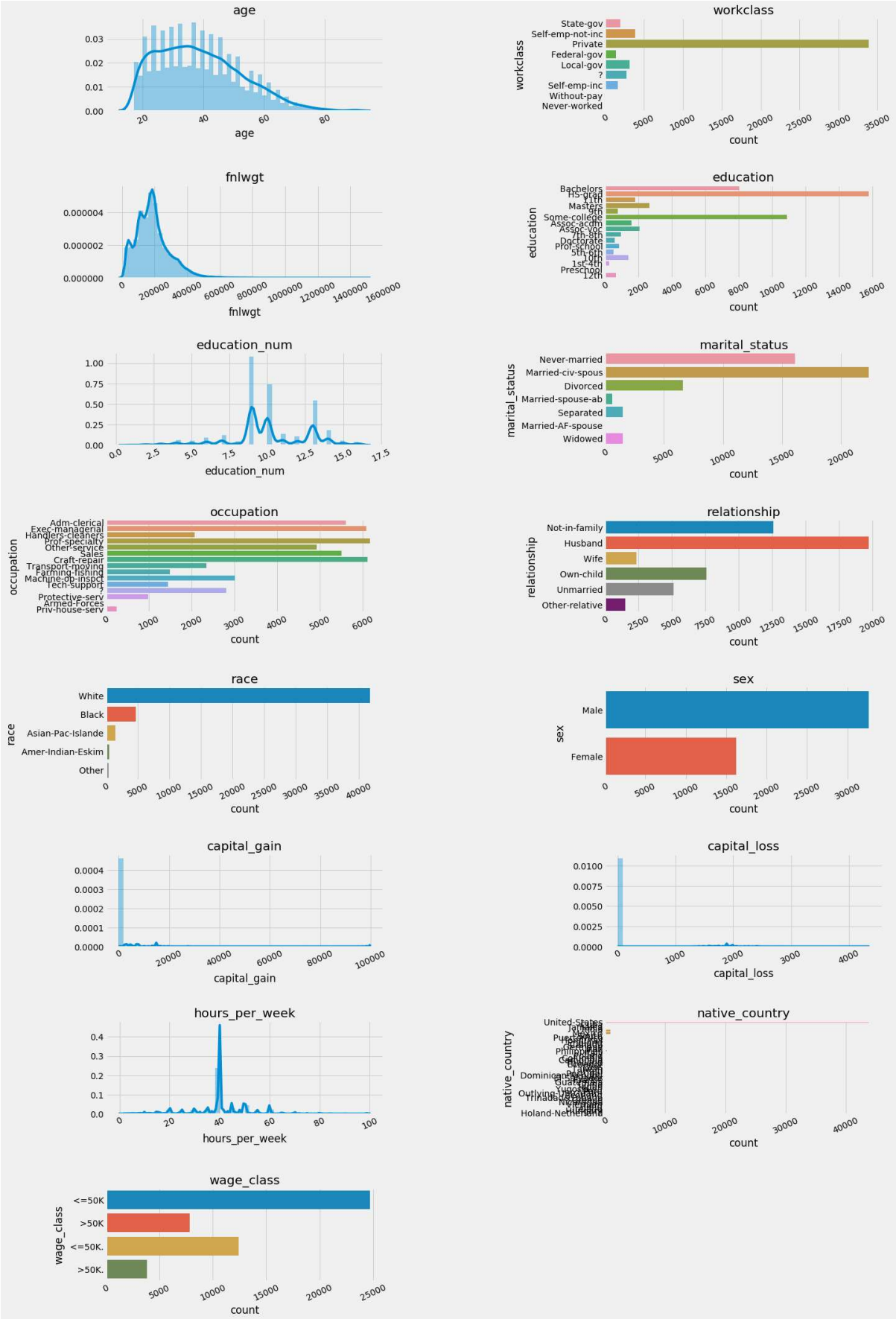
	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week
count	48842.000000	4.884200e+04	48842.000000	48842.000000	48842.000000	48842.000000
mean	38.643585	1.896641e+05	10.078089	1079.067626	87.502314	40.422382
std	13.710510	1.056040e+05	2.570973	7452.019058	403.004552	12.391444
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.175505e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.781445e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.376420e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.490400e+06	16.000000	99999.000000	4356.000000	99.000000

In [20]:
```
# Summary Statistics of Categorical Values

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

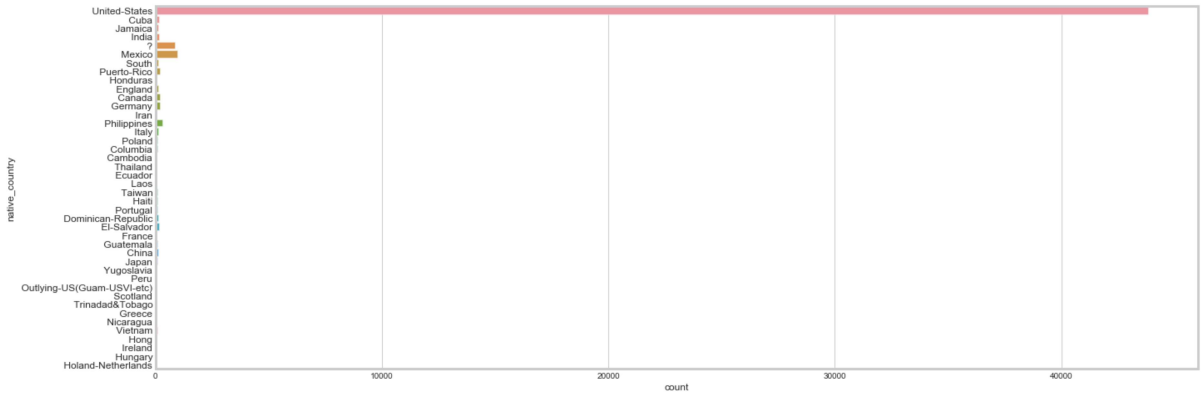
Out[20]:

	workclass	education	marital_status	occupation	relationship	race	sex	native_cou
count	48842	48842	48842	48842	48842	48842	48842	48
unique	9	16	7	15	6	5	2	
top	Private	HS-grad	Married-civ-spouse	Prof-specialty	Husband	White	Male	United-St
freq	33906	15784	22379	6172	19716	41762	32650	43

In [21]:
```
# DATA VISUALIZATION - CATEGORICAL FEATURES

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=w
space, hspace=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)
```


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
workclass
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
education
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
marital_status
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
occupation
0
1000
2000
3000
4000
5000
6000
count
relationship
Not-in-family
Husband
Wife
Own-child
Unmarried
Other-relative
relationship
0
2500
5000
7500
10000
12500
15000
17500
20000
count
race
White
Black
Asian-Pac-Islande
Amer-Indian-Eskim
Other
race
0
5000
10000
15000
20000
25000
30000
35000
40000
count
sex
Male
Female
sex
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
Dominican-Republic
Outlying-US(Guam-USVI-etc)
Holand-Netherlands
native_country
0
10000
20000
30000
40000
count
wage_class
<=50K
>50K
<=50K.
>50K.
wage_class
0
5000
10000
15000
20000
25000
count

In [22]:
```
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()
```



In [23]:
```
print("Note:The data pertaining to native country, workclass and occupation ha
s few unknown values represented by '?'.")
```

```
Note:The data pertaining to native country, workclass and occupation has few
unknown values represented by '?'.
```

In [24]:
```
# Eliminating irrelevant data vale ? from the triaining and test data set

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

In [25]:
```
train_set.isnull().sum()
```

Out[25]:
```
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 [26]:
```
test_set.isnull().sum()
```

Out[26]:
```
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 [27]:
```
test_set.dropna(inplace=True)
train_set.dropna(inplace=True)
```

In [28]:
```
test_set.isnull().sum(),train_set.isnull().sum()
```

Out[28]:
```
(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)
```

In [29]:
```
# Converting Categorical Values to Numeric Values

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

In [30]:
```
dict_workclass ={}
count = 0
for i in X_train.workclass.unique():
    dict_workclass[i] = count
    count +=1
```

In [31]:
```
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 [32]:
```
dict_sex,dict_education,dict_wage_class,dict_native_country,dict_race,dict_occ
upation ,dict_marital_status
```

Out[32]:
```
({' 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 [33]:
```
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 [34]:
```
X_train.isnull().sum()
```

Out[34]:
```
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 [35]:
```
Xtrain = X_train.astype(int)
```

In [36]:
```
X_train.head()
```

Out[36]:

	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 [37]:
```
X_train.describe()
```

Out[37]:

	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 [38]:
```
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 [39]:
```
print("Note:The dataset seems to be almost a balanced dataset with Negative class around 76% and Positive class at 24 %.")
```

```
Note:The dataset seems to be almost a balanced dataset with Negative class around 76% and Positive class at 24 %.
```

In [40]:
```
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 [41]:
```
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 [42]:
```
dict_wage_class
```

Out[42]:
```
{' <=50K.': 0, ' >50K.': 1}
```

In [43]:
```
X_test.head()
```

Out[43]:

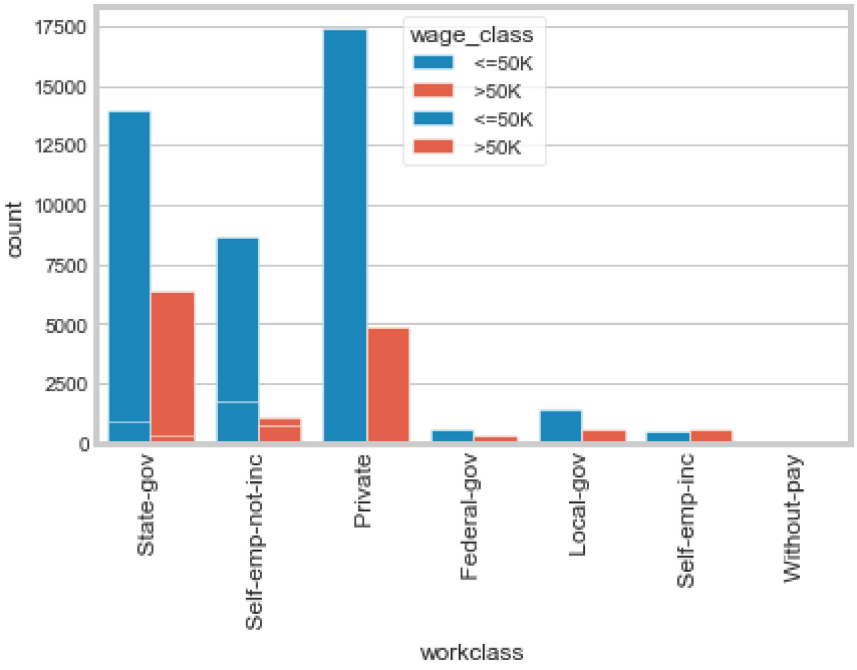
	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship
0	25	2	226802	2	7	0	9	3
1	38	2	89814	1	9	1	8	1
2	28	4	336951	6	12	1	12	1
3	44	2	160323	5	10	1	9	1
4	18	5	103497	5	10	0	11	3

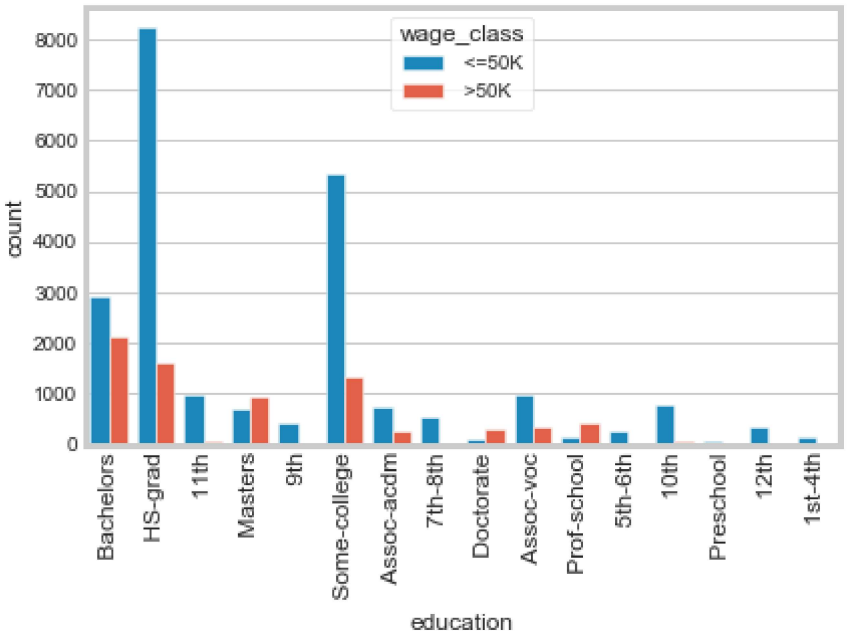
In [44]:
```
X_test.describe()
```

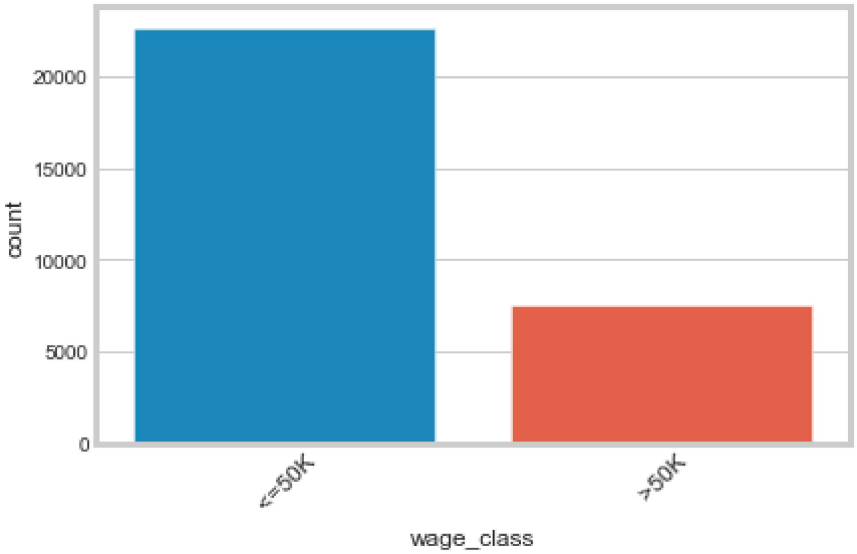
Out[44]:

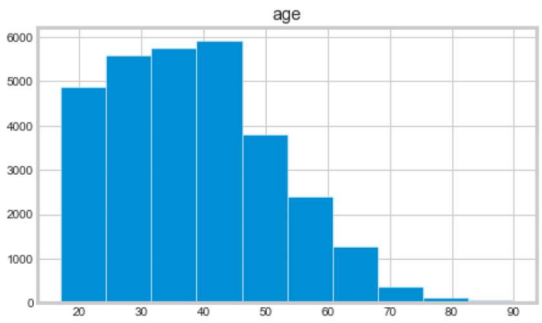
	age	workclass	fnlwgt	education	education_num	marital_status
count	16281.000000	16281.000000	1.628100e+04	16281.000000	16281.000000	16281.000000
mean	38.767459	2.315030	1.894357e+05	3.386954	10.072907	1.084270
std	13.849187	1.246499	1.057149e+05	3.440725	2.567545	1.269622
min	17.000000	0.000000	1.349200e+04	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

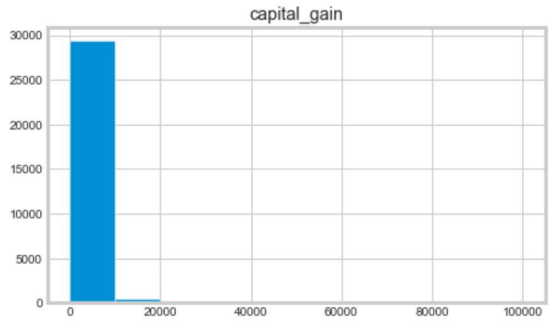
In [45]:
```
# Annual Income Data Analysis using Visualization

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()
```

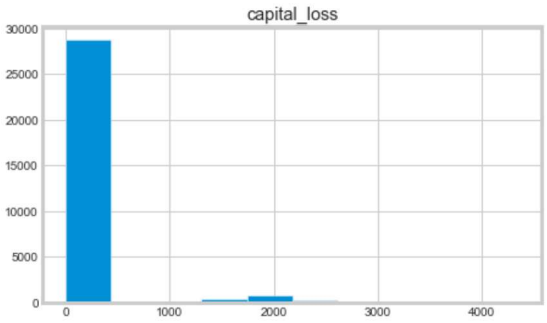

wage_class
<=50K
>50K
<=50K
>50K
count
17500
15000
12500
10000
7500
5000
2500
0
State-gov
Self-emp-not-inc
Private
Federal-gov
Local-gov
Self-emp-inc
Without-pay
workclass

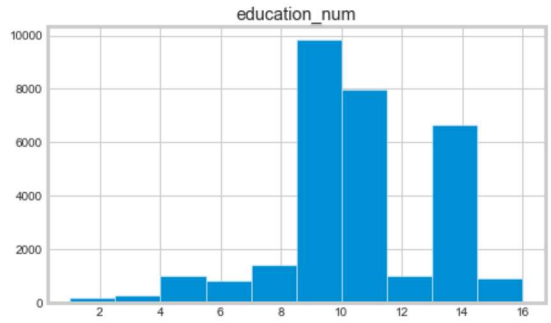

wage_class
<=50K
>50K
count
8000
7000
6000
5000
4000
3000
2000
1000
0
Bachelors
HS-grad
11th
Masters
9th
Some-college
Assoc-acdm
7th-8th
Doctorate
Assoc-voc
Prof-school
5th-6th
10th
Preschool
12th
1st-4th
education

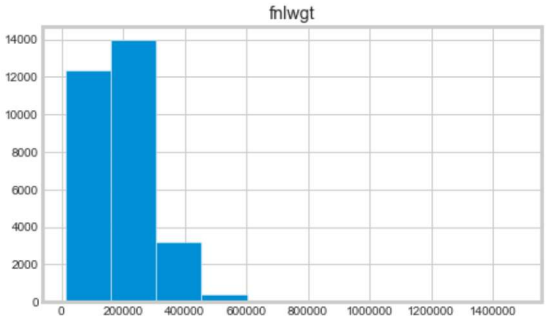

20000
15000
10000
5000
0
count
<=50K
>50K
wage_class

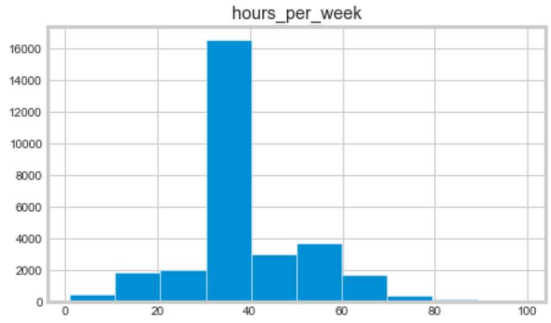

age
6000
5000
4000
3000
2000
1000
0
20
30
40
50
60
70
80
90


capital_gain
30000
25000
20000
15000
10000
5000
0
0
20000
40000
60000
80000
100000


capital_loss
30000
25000
20000
15000
10000
5000
0
0
1000
2000
3000
4000


education_num
10000
8000
6000
4000
2000
0
2
4
6
8
10
12
14
16


fnlwgt
14000
12000
10000
8000
6000
4000
2000
0
0
200000
400000
600000
800000
1000000
1200000
1400000


hours_per_week
16000
14000
12000
10000
8000
6000
4000
2000
0
0
20
40
60
80
100

In [46]:
```
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[46]:
```
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)
```

In [47]:
```
# Separating the Training Label and Test Label from the the training and test
 Features variables
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 [48]:
```
# Training Features and Labels
X = x_train.values
Y = y_train.values
```

In [49]:
```
# Validation Features and Labels

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

In [50]:
```
x_train.shape,y_train.shape,X.shape,Y.shape,Xtest.shape,Ytest.shape
```

Out[50]:
```
((32561, 14), (32561,), (32561, 14), (32561,), (16281, 14), (16281,))
```

In [51]:
```
Xtest
```

Out[51]:
```
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)
```

In [52]:
```
# Applying Logistic Regression
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)
```

```
0.8003193907008169
```

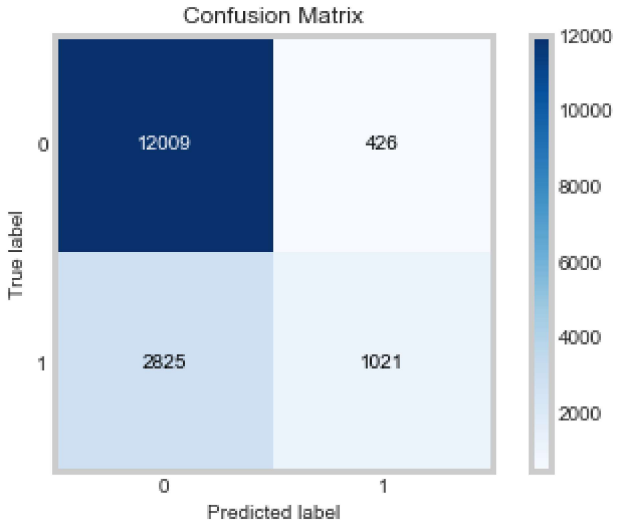
In [53]:
```
test_data =[]
for i in x_test.columns:
    test_data.append(x_test[i].max())
print(test_data)
test = np.array(test_data).reshape(-1,14)
print(test.shape)
print("Predicted Label \n ")
print(LR.predict(test))
print('Prediction Probabilities \n ')
print(LR.predict_proba(test))
print('coefficients = ',LR.coef_)
```

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

[1]
Prediction Probabilities

[[5.90416604e-13 1.00000000e+00]]
coefficients =  [[-2.91608163e-03 -2.17836060e-03 -4.29056481e-06 -6.43016145
e-03
  -1.51688129e-03 -1.15630342e-03 -9.85087599e-03 -5.26420502e-03
  -8.64047197e-04 -1.88225973e-03  3.30353419e-04  7.61250318e-04
  -6.66780735e-03 -2.84932147e-03]]
```

In [54]:
```
# problem 1 - Prediction task is to determine whether a person makes over 50K
 a year.
print("Note:That implies that there is a 100% possibility of the person to hav
e a income > 50k for the input features as evident from prediction probabilit
y.")
```

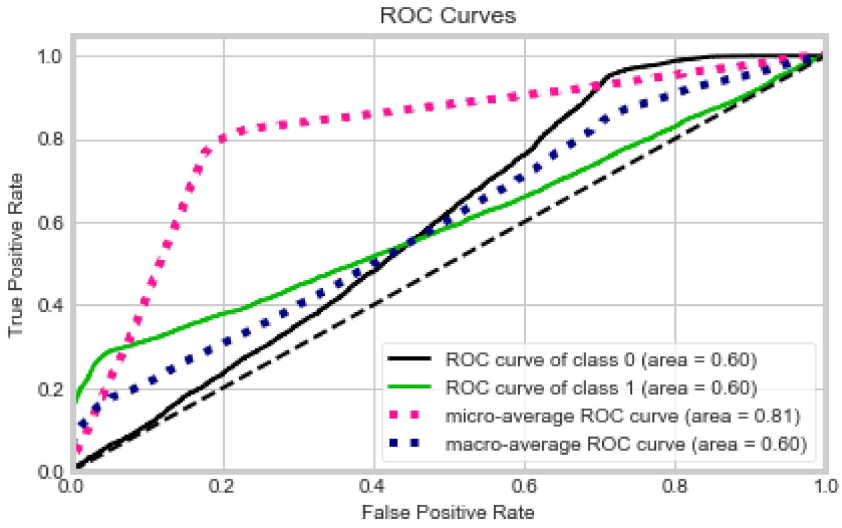
```
Note:That implies that there is a 100% possibility of the person to have a in
come > 50k for the input features as evident from prediction probability.
```

In [55]:
```
# Evaluation of Logistic Regression Model
# Creating the confusion matrix

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



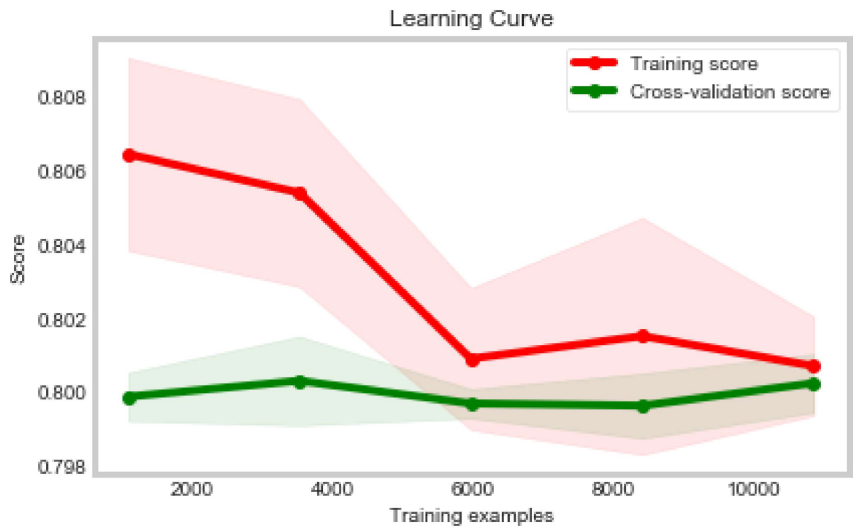
In [56]:
```
# Receiver Operating Characteristic Curve

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

Out[56]: <matplotlib.axes._subplots.AxesSubplot at 0x238aa4f4898>



In [57]:
```
# Learning Curve of Logistic Regression Classifier
scikitplot.estimators.plot_learning_curve(LR,Xtest,Ytest)
```

Out[57]: <matplotlib.axes._subplots.AxesSubplot at 0x238ac9ceef0>



In [58]:
```
print("ROC : ",(metrics.roc_curve(Ytest,Ypred_proba[:,1])))
print("AUC : ",(metrics.roc_auc_score(Ytest,Ypred_proba[:,1])))
```

```
ROC :  (array([0.        , 0.        , 0.        , ..., 0.99943707, 0.9994370
7,
       1.        ]), array([0.00000000e+00, 5.20020801e-04, 2.08008320e-03,
...,
       9.99739990e-01, 1.00000000e+00, 1.00000000e+00]), array([2.00000000e+0
0, 1.00000000e+00, 1.00000000e+00, ...,
       1.18343043e-02, 1.18273605e-02, 1.10227404e-03]))
AUC :  0.5996124935467865
```

In [59]:
```
model_accuracy['AUC_Logistic_Regression'] = metrics.roc_auc_score(Ytest,Ypred_
proba[:,1])
```

In [60]:
```
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      0.27      0.39      3846

   micro avg       0.80      0.80      0.80     16281
   macro avg       0.76      0.62      0.63     16281
weighted avg       0.79      0.80      0.76     16281
```

In [61]:
```
# Applying 10 Fold Cross Validation to Logistic Regression Model
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 0.79488
229 0.80118755
 0.80405405 0.79909891 0.79582224 0.79848454]
Average accuracy:  0.7988821300510068
```

In [62]:
```
# Feature Selection for Logistic Regression Model
# The goal of recursive feature elimination (RFE) is to select features by rec
ursively considering smaller and smaller
# sets of features. First, the estimator is trained on the initial set of feat
ures and the importance of each feature is
# obtained either through a coef_ attribute or through a featureimportances at
tribute. Then, the least important features
# are pruned from current set of features. That procedure is recursively repea
ted on the pruned set until the desired number
# of features to select is eventually reached.

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

Out[62]:
```
RFE(estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_in
tercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False),
  n_features_to_select=None, step=1, verbose=0)
```

In [63]:
```
# ranking_ : array of shape [n_features]
# The feature ranking, such that ranking_[i] corresponds to the ranking positi
on of the i-th feature.
# Selected (i.e., estimated best) features are assigned rank 1.

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

Out[63]:
```
[2, 1, 8, 4, 1, 1, 1, 1, 1, 1, 7, 6, 3, 5]
```

In [64]:
```
df_rank = pd.DataFrame({'Feature':Features.columns,'Rank':ranks})
df_rank
```

Out[64]:

	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 [65]:
```
#Problem 2 - What factors are imporatnt
imp= df_rank.Feature[df_rank.Rank == 1]
print("The important Features in the sample data are as follows :-\n",imp.valu
es)
```

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

In [66]:
```
# Feature ranking with recursive feature elimination and cross-validated selec
tion of the best number of features.

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[66]:

	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 [67]:
```
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.values)
```

```
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']
```

In [68]:
```
# Applying Decison Tree Classifier Model to the sample Data
#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, Y
pred))
#model_accuracy['Decision Tree Classifier'] = metrics.accuracy_score(Ytest, Yp
red)
```

```
Accuracy of Decision Tree Classifier : 0.8160432405871875
```

In [69]:
```
# Decision Tree Classifier Model with max_depth in a range of 1 to 20
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," : ",m
etrics.accuracy_score(Ytest, Ypred))

    #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  :  0.8586081935999017
Accuracy of Decision Tree Classifier for max_depth  11  :  0.8569498188071986
Accuracy of Decision Tree Classifier for max_depth  12  :  0.8557828143234445
Accuracy of Decision Tree Classifier for max_depth  13  :  0.8533259627786991
Accuracy of Decision Tree Classifier for max_depth  14  :  0.848289417111971
Accuracy of Decision Tree Classifier for max_depth  15  :  0.8444812972176157
Accuracy of Decision Tree Classifier for max_depth  16  :  0.8404889134574043
Accuracy of Decision Tree Classifier for max_depth  17  :  0.8381549044898962
Accuracy of Decision Tree Classifier for max_depth  18  :  0.8361280019654812
Accuracy of Decision Tree Classifier for max_depth  19  :  0.8313371414532277
Accuracy of Decision Tree Classifier for max_depth  20  :  0.8272219151157791
```

In [70]:
```
print("The model gives a better accuracy with depth = 10")
```

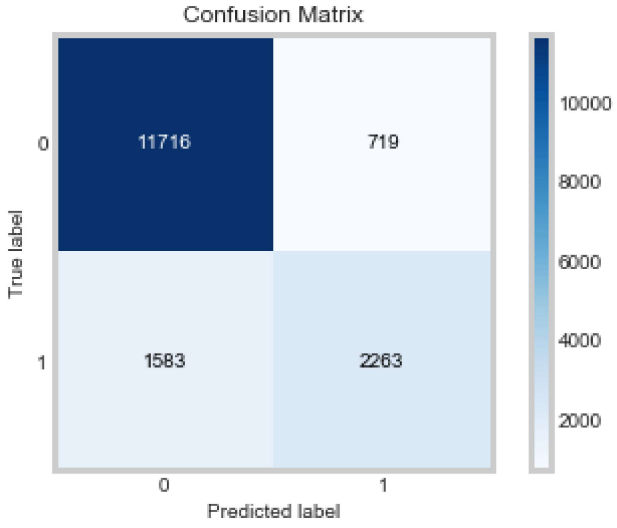
```
The model gives a better accuracy with depth = 10
```

In [71]:
```
#since the model has best accuracy for max_depth 10 so retraining the model wi
th max_depth as 10
#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, Y
pred))
model_accuracy['Accuracy Score of Decision Tree Classifier Model'] = metrics.a
ccuracy_score(Ytest, Ypred)
model_accuracy['AUC of Decision Tree Model Classifier - depth 10'] = metrics.r
oc_auc_score(Ytest,Ypred_proba[:,1])
```

```
Accuracy of Decision Tree Classifier : 0.8586081935999017
```

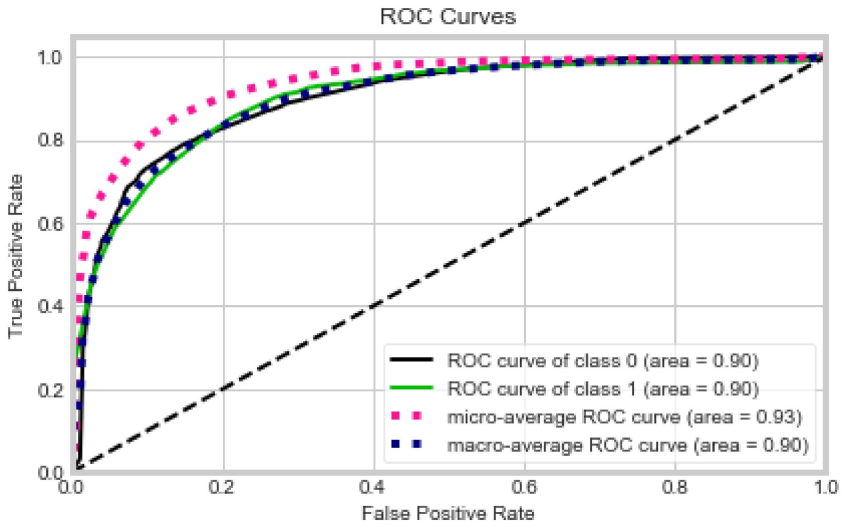
In [72]:
```
# Evaluation of the Decision Tree Model trained with max_depth as 10
# creating the confusion matrix

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

Out[72]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x238aca25f28>
```



In [73]:
```
# Receiver Operating Characteristic Curve
scikitplot.metrics.plot_roc(Ytest,Ypred_proba)
```

Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x238ab624588>

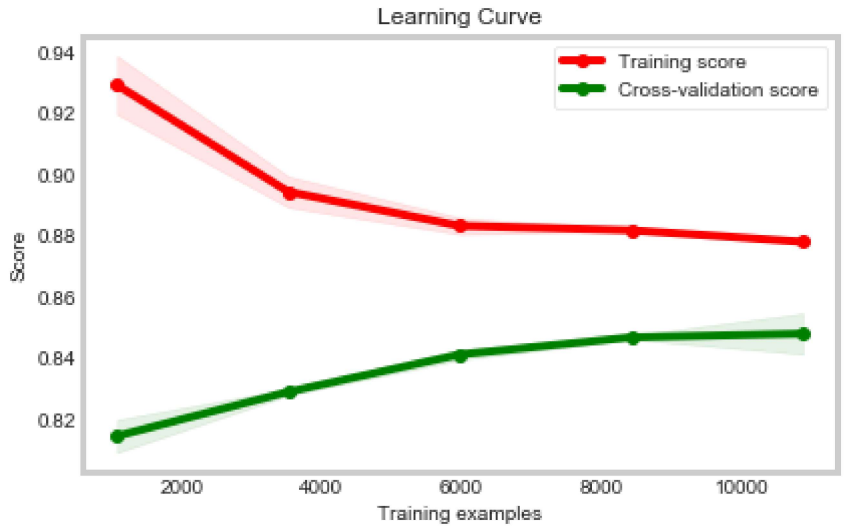


In [74]:
```
print("Generally a highly skillfull model is represented by a curve that blows up to the left of the plot.")
```

```
Generally a highly skillfull model is represented by a curve that blows up to the left of the plot.
```

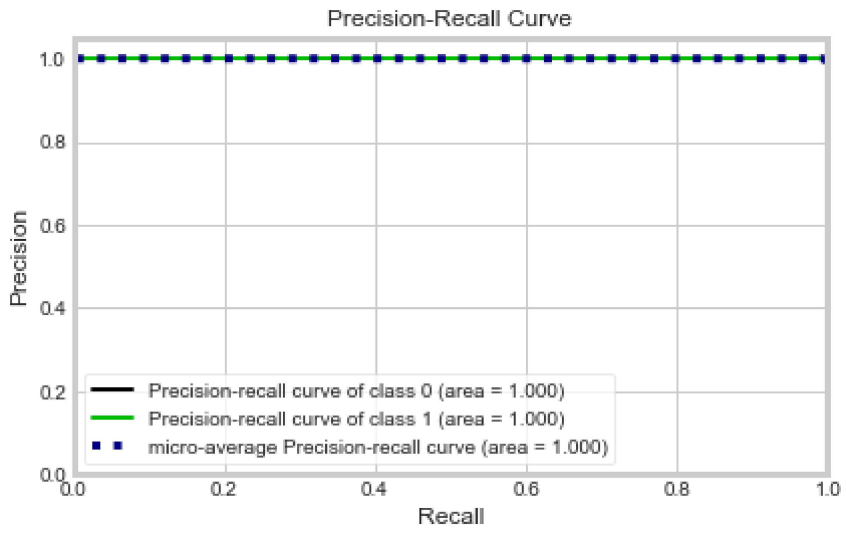
In [75]:
```
# Learning Curve for Decision Tree Classifier
scikitplot.estimators.plot_learning_curve(DT,Xtest,Ytest)
```

Out[75]: <matplotlib.axes._subplots.AxesSubplot at 0x238aa2b5e80>



In [76]:
```
# Precision -REcall Curve for Decision Tree Classifier
scikitplot.metrics.plot_precision_recall(Ypred,Ypred_proba)
```

Out[76]: <matplotlib.axes._subplots.AxesSubplot at 0x238abb90400>



In [77]:
```
# Applying 10 Fold cross validation to DEcision Tree Classifier Model
from sklearn.model_selection import cross_val_score

scores = cross_val_score(estimator= DecisionTreeClassifier(random_state=0),
# 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())
```

```
Accuracy per fold:
Cross Validation score:  [0.82006141 0.80900716 0.81453429 0.81576254 0.82006
141 0.81900082
 0.81511057 0.82387876 0.80749539 0.80810977]
Average accuracy:  0.8153022124714788
```

In [78]:
```
for depth in range(20):
    depth = depth + 1
    scores = cross_val_score(estimator= DecisionTreeClassifier(max_depth=depth
,random_state=0),     # Model to test
                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 [79]:
```
print("The cross validation score for Decision Tree Classifier is the most opt
imum for max_depth =12")
```

```
The cross validation score for Decision Tree Classifier is the most optimum f
or max_depth =12
```

In [80]:
```
scores = cross_val_score(estimator= DecisionTreeClassifier(max_depth=12,random
_state=0),     # 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-Decision Tree Classifier, max depth  12'] = scores
.mean()
```

```
Accuracy per fold:
Cross Validation score:  [0.8493347  0.8616172  0.85301945 0.85875128 0.86612
078 0.85892711
 0.85749386 0.86053656 0.85091133 0.85623592]
Average accuracy:  0.8572948165520635
```

In [81]:
```
# Feature Selection using REFCV for Decision Tree Classifier
selector =  RFECV(estimator=DecisionTreeClassifier(max_depth=12,random_state=0
), 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[81]:

	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 [82]:
```
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.values)
```

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

In [83]:
```
# Applying K- Nearest Neighbor Model to sample Data
from sklearn.neighbors import KNeighborsClassifier
```

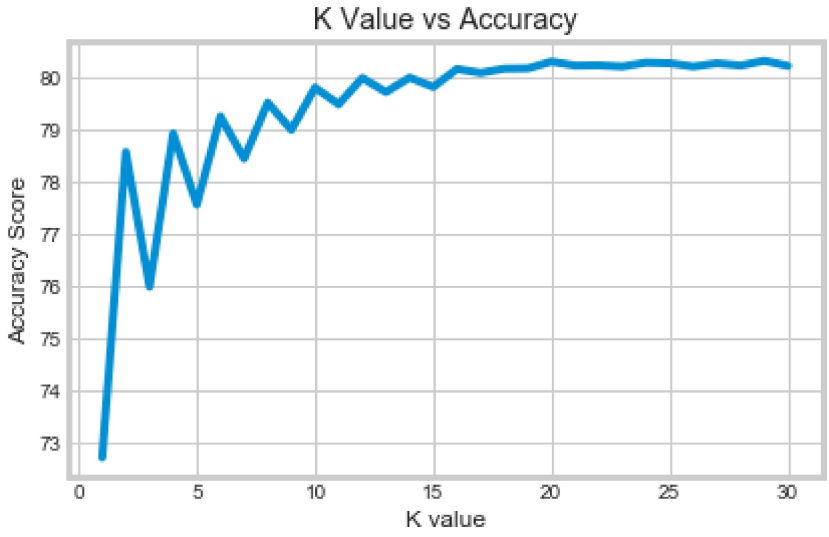
In [84]:
```
k = []
scores = []
errors = []
for K in range(30):
    K_value = K+1
    neigh = KNeighborsClassifier(n_neighbors = K_value, weights='uniform', alg
orithm='auto')
    neigh.fit(X,Y)
    y_pred = neigh.predict(Xtest)
    print("Accuracy is ", metrics.accuracy_score(Ytest,y_pred)*100,"% for K-Va
lue:",K_value)
    print("Error  is ", 100 - metrics.accuracy_score(Ytest,y_pred)*100,"% for
 K-Value:",K_value)
    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  19.771512806338677 % 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 [85]:
```
plt.plot(k,scores)
plt.xlabel('K value')
plt.ylabel('Accuracy Score')
plt.title('K Value vs Accuracy')
```

Out[85]:
```
Text(0.5, 1.0, 'K Value vs Accuracy')
```

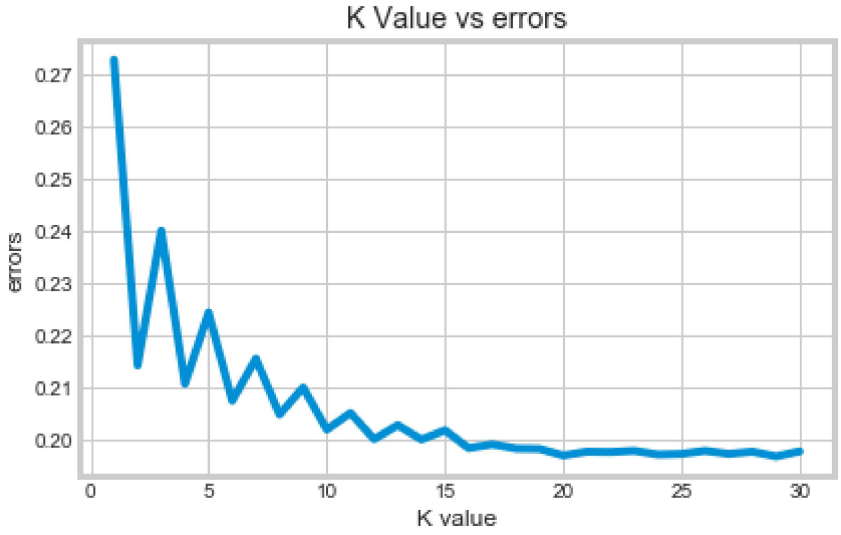


In [86]:
```
print("It is evident that from the plot that k value after 20 yields almost th
e same performance. Hence k =20 is optimum.")
```

```
It is evident that from the plot that k value after 20 yields almost the same
performance. Hence k =20 is optimum.
```

In [87]:
```
plt.plot(k,errors)
plt.xlabel('K value')
plt.ylabel('errors')
plt.title('K Value vs errors')
```

Out[87]: Text(0.5, 1.0, 'K Value vs errors')



In [88]:
```
print("The errors gradually decreases with increase in K values and does not r
educe further after K =20 .It becomes constant. Hence optimum K value = 20")
```

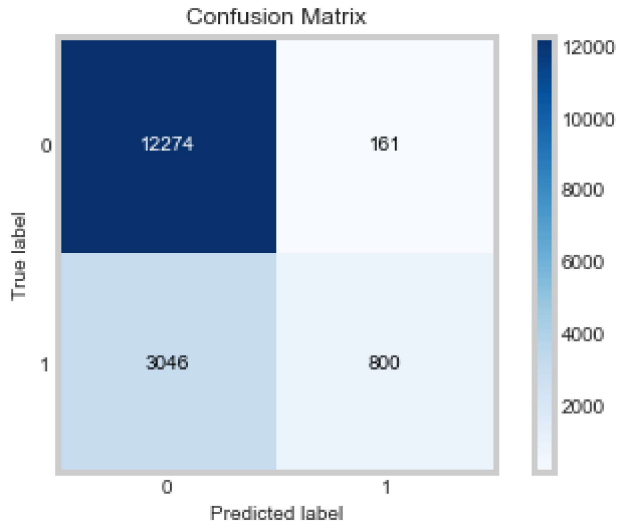
```
The errors gradually decreases with increase in K values and does not reduce
further after K =20 .It becomes constant. Hence optimum K value = 20
```

In [89]:
```
knn = KNeighborsClassifier(n_neighbors = 20, weights='uniform', algorithm='aut
o')
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.accur
acy_score(Ytest, Ypred)

model_accuracy['AUC of KNN Classifier neighbors-20'] = metrics.roc_auc_score(Y
test,Ypred_proba[:,1])
```

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

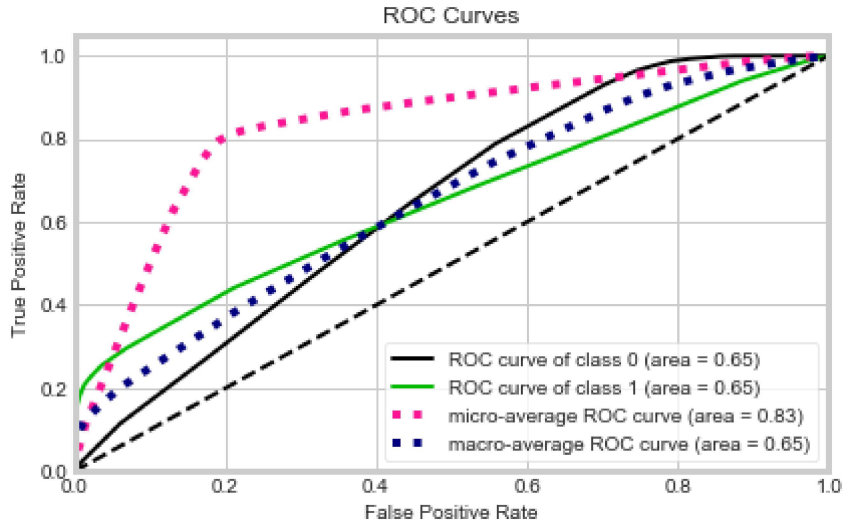
In [90]:
```
# Evaluation of K Nearest Neighbor for K = 20
# Creating confusion matrix

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

Out[90]: <matplotlib.axes._subplots.AxesSubplot at 0x238ac9f6828>



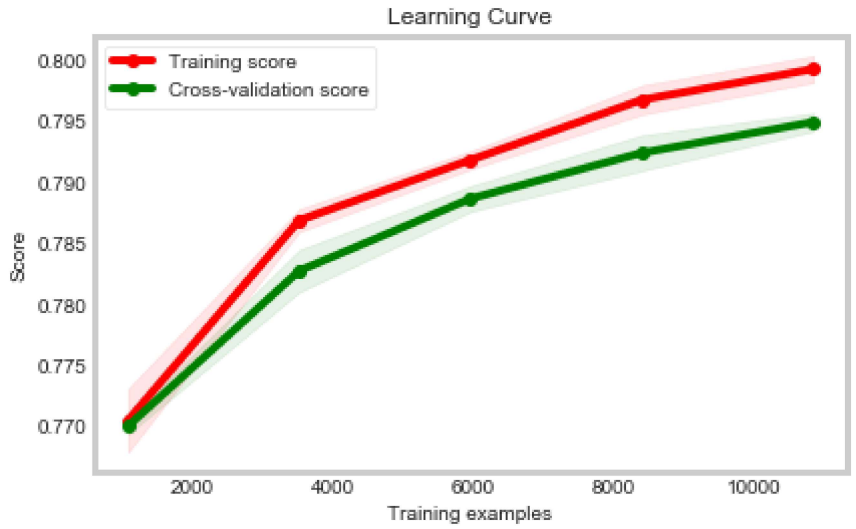
In [91]:
```
# Receiver operating characteristic curve
scikitplot.metrics.plot_roc(Ytest,Ypred_proba)
```

Out[91]: <matplotlib.axes._subplots.AxesSubplot at 0x238ab6b16d8>



In [92]:
```
# Learning Curve of KNN Classifier Model
scikitplot.estimators.plot_learning_curve(knn,Xtest,Ytest)
```

Out[92]: <matplotlib.axes._subplots.AxesSubplot at 0x238aca6b198>



In [93]:
```
# 10 Fold Cross Validation applied to K nearest neighbor model
from sklearn import datasets, linear_model
from sklearn.model_selection import cross_val_score
scores = cross_val_score(estimator= KNeighborsClassifier(n_neighbors = 20, wei
ghts='uniform', algorithm='auto'),
# 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-KNN Classifier neighbors-20'] = scores.mean()
```

```
Accuracy per fold:
Cross Validation score:  [0.80143296 0.80225179 0.80429887 0.80266121 0.79959
058 0.7993448
 0.8046683  0.80032767 0.79746058 0.79930371]
Average accuracy:  0.8011340470216448
```

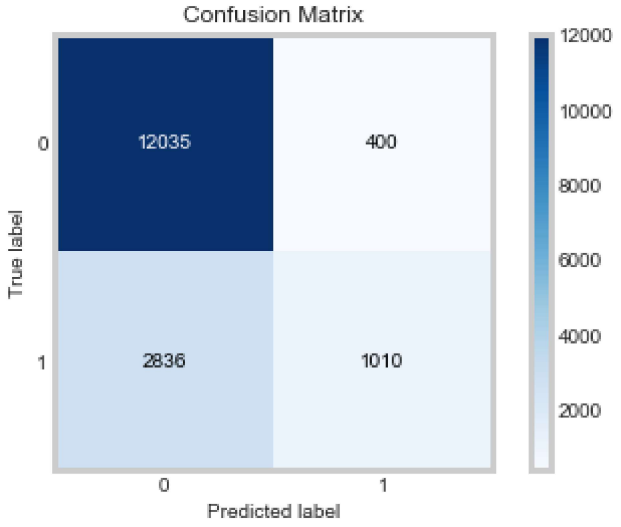
In [94]:
```
print("knn does not provide logic to do feature selection")
```

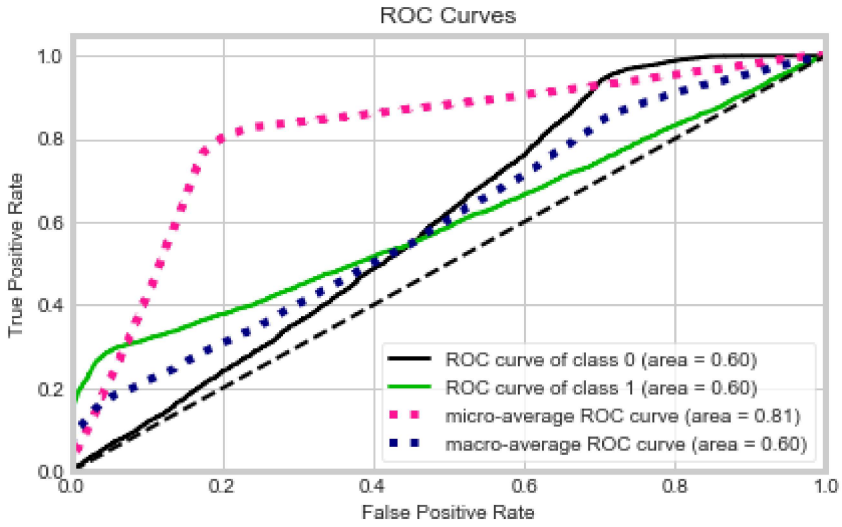
```
knn does not provide logic to do feature selection
```

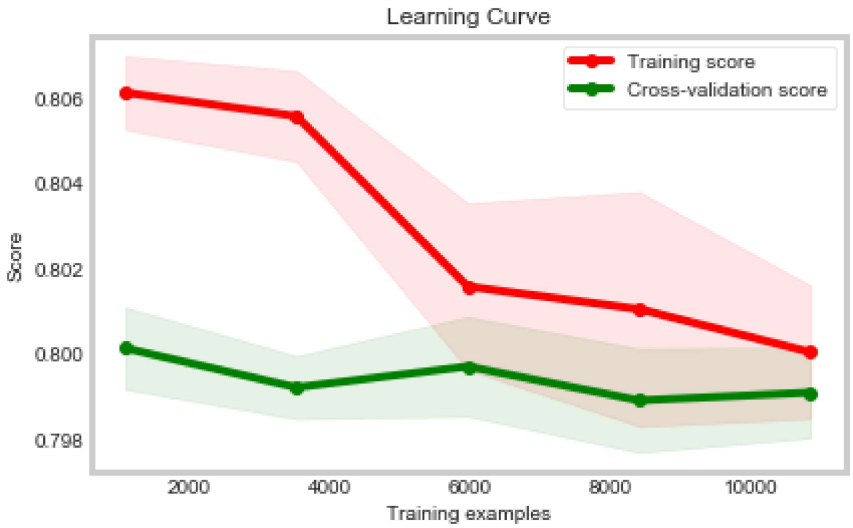
In [95]:
```
# ENSEMBLE MODEL - BAGGING TECHNIQUE
# Bagging with Logistic Regression

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 :",metri
cs.accuracy_score(Ytest,bag_preds))

#### 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.accurac
y_score(Ytest,bag_preds)
model_accuracy['AUC-Bagging-Logistic Regression'] = metrics.roc_auc_score(Ytes
t,bag_preds_proba[:,1])

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

```
Predictions :  [0 0 0 ... 0 1 0]
Prediction Probabilities :  [[0.81156283 0.18843717]
 [0.73510814 0.26489186]
 [0.87842468 0.12157532]
 ...
 [0.87698788 0.12301212]
 [0.26852472 0.73147528]
 [0.78501132 0.21498868]]
Accuracy Score of Bagging for single Logistic Regression Model : 0.8012407100
300964
```

Out[95]: <matplotlib.axes._subplots.AxesSubplot at 0x238aca56f28>



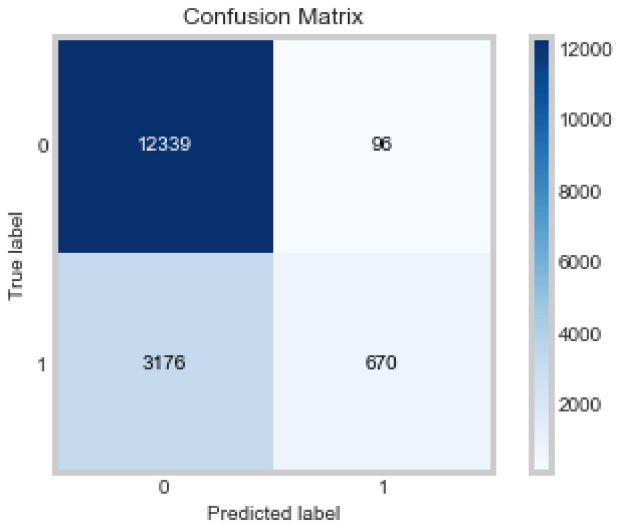


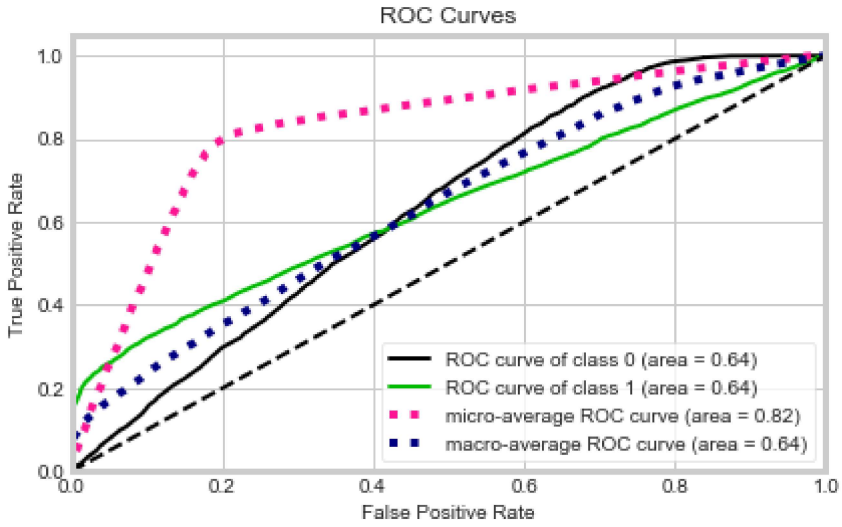

Learning Curve
Training score
Cross-validation score
0.806
0.804
0.802
0.800
0.798
Score
2000
4000
6000
8000
10000
Training examples

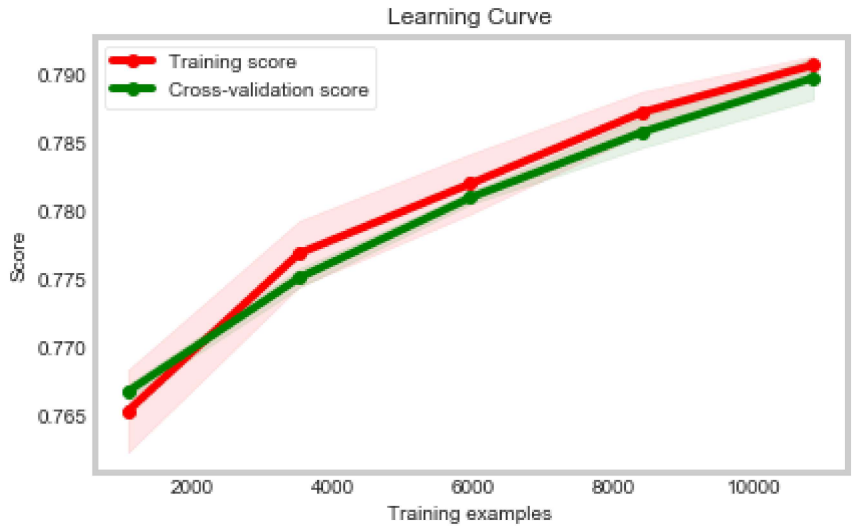
In [96]:
```
# Bagging with KNN Model
from sklearn.ensemble import BaggingClassifier
bag_KNN = BaggingClassifier(KNeighborsClassifier(n_neighbors = 20, weights='un
iform', algorithm='auto'),
                            n_estimators=10, max_samples=0.5,
                            bootstrap=True, random_state=3)

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_scor
e(Ytest,bag_preds))

#### 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,bag_preds)
model_accuracy['AUC-Bagging-KNN neighbors -20'] = metrics.roc_auc_score(Ytest,
bag_preds_proba[:,1])
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[96]: <matplotlib.axes._subplots.AxesSubplot at 0x238ab1a9e10>





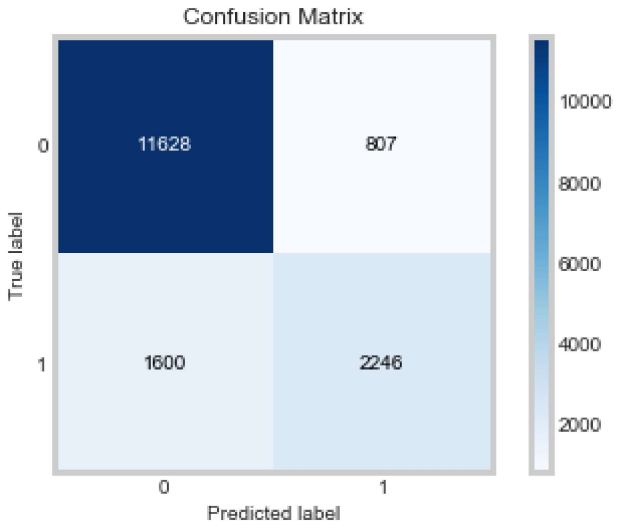


In [97]:
```
# RANDOM FOREST CLASSIFIER model
from sklearn.ensemble import  RandomForestClassifier
RF = RandomForestClassifier()
RF.fit(X,Y)
Ypred = RF.predict(Xtest)
Ypred_proba = RF.predict_proba(Xtest)
print("accuracy of Random Forest Classifier :",metrics.accuracy_score(Ytest, Y
pred))
model_accuracy['Accuracy score of Random Forest Classifier'] = metrics.accurac
y_score(Ytest, Ypred)
```

```
accuracy of Random Forest Classifier : 0.852158958294945
```

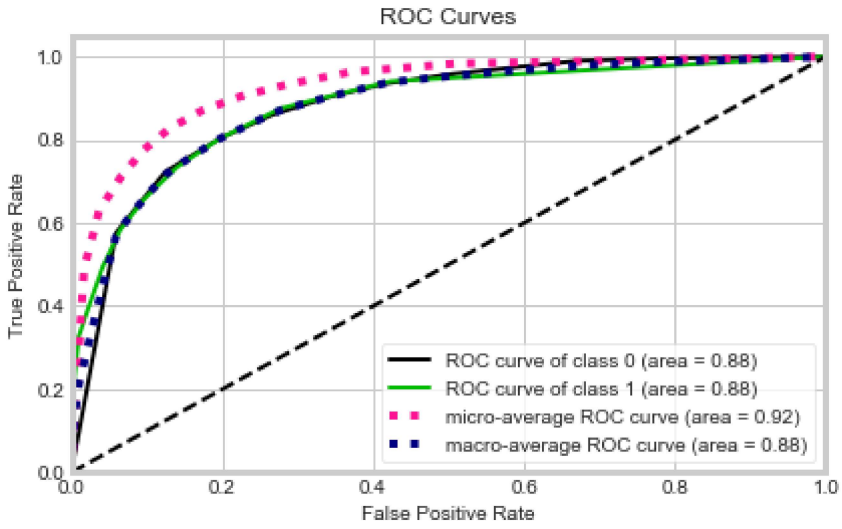
In [98]:
```
# Evaluate the Random Forest Model
# Creating confusion matrix

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

Out[98]: <matplotlib.axes._subplots.AxesSubplot at 0x238ab4b8c50>



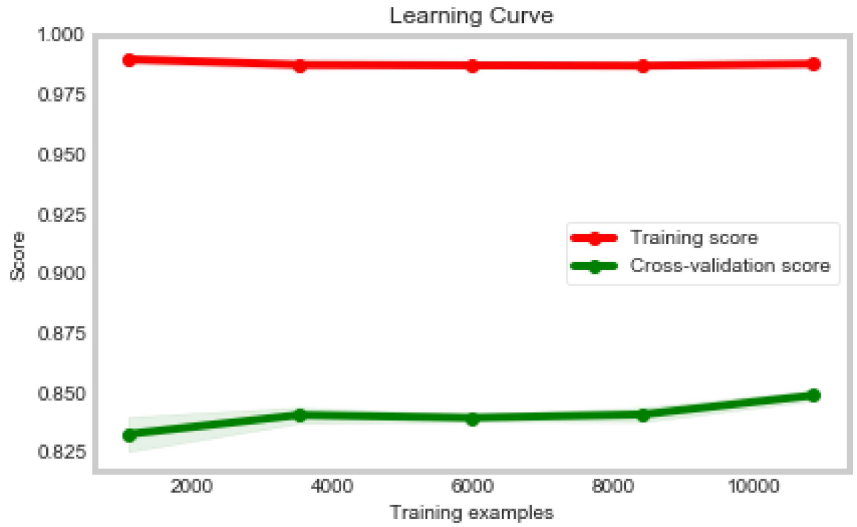
In [99]:
```
# Reciver Operating Characteristic Curve for Random Forest Model
scikitplot.metrics.plot_roc(Ytest,Ypred_proba)
```

Out[99]: <matplotlib.axes._subplots.AxesSubplot at 0x238acaeb9b0>



In [100]:
```
# Learning Curve
scikitplot.estimators.plot_learning_curve(RF,Xtest,Ytest)
```

Out[100]: <matplotlib.axes._subplots.AxesSubplot at 0x238ab132550>



In [101]:
```
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(Yte
st,Ypred_proba[:,1])
```

```
AUC for Random Forest Classifier :  0.8809871341375569
```

In [102]:
```
# Feature Selection using featureimportances parameter of Random Forest Model
RF.fit(Xtest, Ytest)
print("Features sorted by their score:")
print(sorted(zip(map(lambda x: round(x, 4), RF.feature_importances_), Features
.columns),reverse=True))
```

```
Features sorted by their score:
[(0.1573, 'fnlwgt'), (0.1432, 'age'), (0.1209, 'capital_gain'), (0.1079, 'rel
ationship'), (0.0921, 'education_num'), (0.083, 'hours_per_week'), (0.0753,
'marital_status'), (0.0738, 'occupation'), (0.0399, 'workclass'), (0.0355, 'c
apital_loss'), (0.0291, 'education'), (0.0173, 'native_country'), (0.0139, 'r
ace'), (0.011, 'sex')]
```

In [103]:
```
# Feature Selection using RFECV - Recursive Feature Elimination Using Cross Va
lidation

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[103]:

	Feature	Rank
0	age	1
1	workclass	1
2	fnlwgt	1
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 [104]:
```
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.values)
```

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

In [105]:
```
# 10 Fold Cross Validation for Random Forest Classifier
from sklearn import datasets, linear_model
from sklearn.model_selection import cross_val_score
scores = cross_val_score(estimator= RandomForestClassifier(),     # Model to t
est
                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:  [0.84728762 0.85568066 0.84851586 0.85486182 0.85731
832 0.86015561
 0.84889435 0.84968257 0.85009216 0.854188  ]
Average accuracy:  0.8526676965284485
```

In [106]:
```
# Using Boosting Method of Ensemble model to predict the annual income
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']
```

In [107]:
```
# Train the XGBClassifier model
bst = XGBClassifier(**params).fit(X,Y)
```

In [108]:
```
# Predict the annual income
preds = bst.predict(Xtest)
preds
```

Out[108]:
```
array([0, 0, 0, ..., 1, 0, 1], dtype=int64)
```

In [109]:
```
preds_proba = bst.predict_proba(Xtest)
preds_proba
```

Out[109]:
```
array([[0.9862895 , 0.01371049],
       [0.6448917 , 0.35510832],
       [0.8749048 , 0.1250952 ],
       ...,
       [0.282     , 0.718     ],
       [0.71667016, 0.28332984],
       [0.17598617, 0.8240138 ]], dtype=float32)
```

In [110]:
```
# Measure the accuracy of the model
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
```

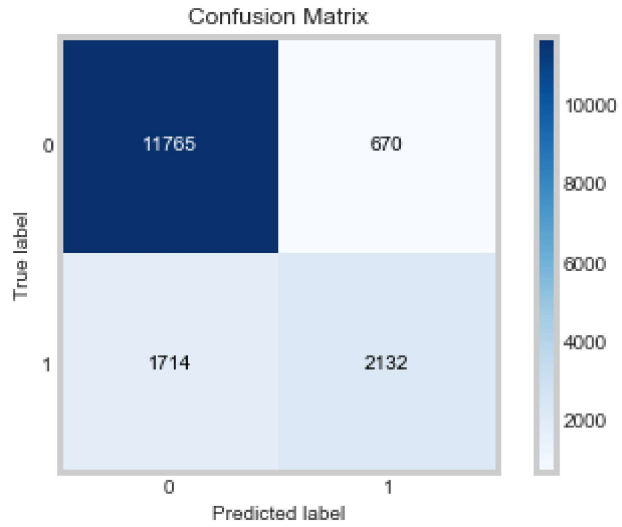
```
Predicted correctly: 13897/16281
Accuracy Score :0.8536
Error: 0.1464
```

In [111]:
```
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

   micro avg       0.85      0.85      0.85     16281
   macro avg       0.82      0.75      0.77     16281
weighted avg       0.85      0.85      0.85     16281
```

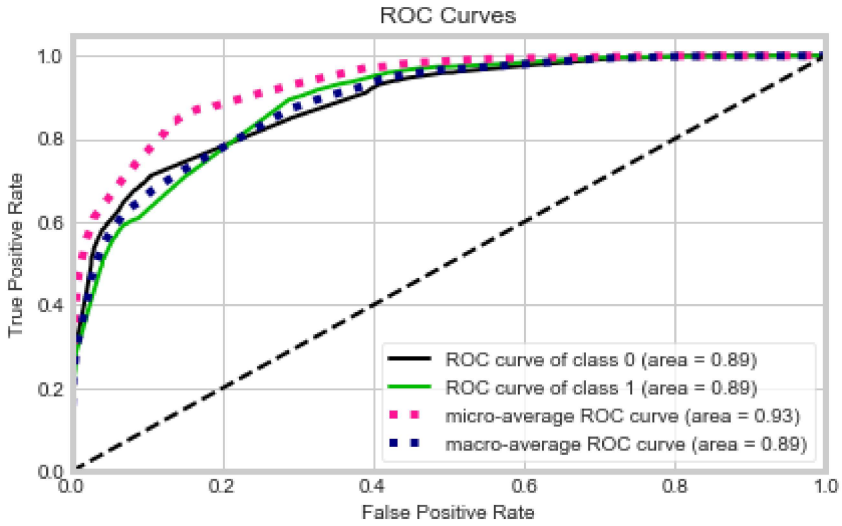
In [112]:
```
# Confusion matrix
import scikitplot
scikitplot.metrics.plot_confusion_matrix(Ytest, preds)
```

Out[112]: <matplotlib.axes._subplots.AxesSubplot at 0x238ab4e5518>



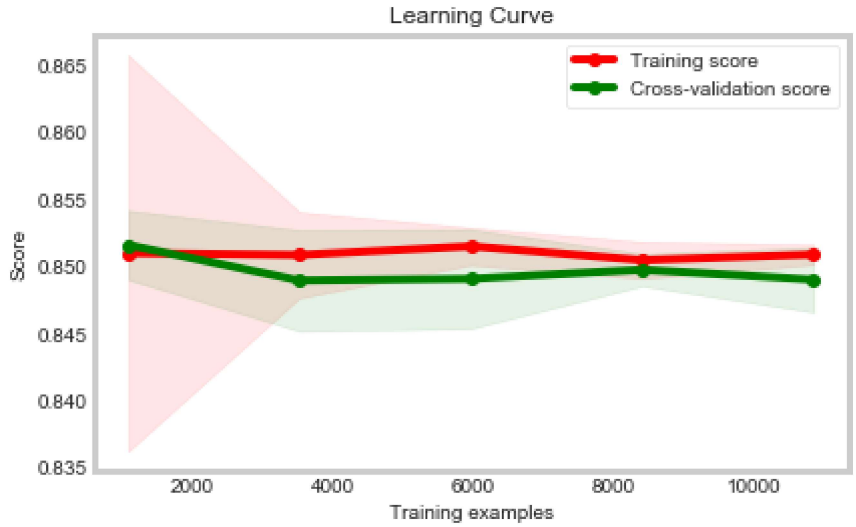
In [113]:
```
scikitplot.metrics.plot_roc(Ytest,preds_proba)
```

Out[113]: <matplotlib.axes._subplots.AxesSubplot at 0x238ab59a5c0>



In [114]:
```
#### Learning Curve
scikitplot.estimators.plot_learning_curve(bst,Xtest,Ytest)
```

Out[114]: <matplotlib.axes._subplots.AxesSubplot at 0x238abf485c0>



In [115]:
```
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_pr
oba[:,1])
```

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

In [116]:
```
features = []
scores = []
for k,v in model_accuracy.items():
    features.append(k)
    scores.append(v)
```

In [117]:
```
df_scores = pd.DataFrame({'Features':features,'Scores':scores})
feat_cols = ['Features','Scores']
df_scores = df_scores[feat_cols]
df_scores
```

Out[117]:

	Features	Scores
0	Logistic Regression	0.800319
1	AUC_Logistic_Regression	0.599612
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	0.801241
10	AUC-Bagging-Logistic Regression	0.601070
11	Accuracy Score-Bagging-KNN neighbors -20	0.799030
12	AUC-Bagging-KNN neighbors -20	0.635693
13	Accuracy score of Random Forest Classifier	0.852159
14	AUC for Random Forest Classifier	0.880987
15	10 CV Score-Random Forest Classifier	0.852668
16	Accuracy Score of XGBOOST Model	0.853572
17	AUC for XGBOOST model	0.889614

In [118]:
```
# problem 3 - Which algorithms are best for this dataset

print("From the above Model Estimation Score dataframe if we take into conside
ration the AUC value then it is evident that the Decision Tree Classifier Mode
l and XGBOOST ensemble model have the highest accuracy for the model performan
ce.")

print('AUC of Decision Tree Model Classifier - depth-10 is 89.79%')
print('AUC for XGBOOST model is 88.96%')
print('AUC for Random Forest Classifier is 88.02%')
print('AUC of KNN Classifier neighbors-20 is 64.86%')
print('AUC-Bagging-KNN neighbors -20 is 63.56%')
print('AUC_Logistic_Regression is 61.13%')
print('AUC-Bagging-Logistic Regression is 59.31%')
```

```
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 of Decision Tree Model Classifier - depth-10 is 89.79%
AUC for XGBOOST model is 88.96%
AUC for Random Forest Classifier is 88.02%
AUC of KNN Classifier neighbors-20 is 64.86%
AUC-Bagging-KNN neighbors -20 is 63.56%
AUC_Logistic_Regression is 61.13%
AUC-Bagging-Logistic Regression is 59.31%
```