income_classification

January 23, 2019

1 Project: Income Prediction of Individuals Using Machine Learning

1.1 Overview:

In this project I have used machine learning and analysis techinque to predict income of an individaul. This data was extracted from the census bureau database found at http://www.census.gov/ftp/pub/DES/www/welcome.html donated by Ronny Kohavi and Barry Becker, Data Mining and Visualization (Email: ronnyk@sgi.com)

Dataset Link: https://archive.ics.uci.edu/ml/machine-learning-databases/adult/

I this project I tried to address three important problems as given below:

- Problem 1: Prediction task is to determine whether a person makes over 50K a year.
- Problem 2: Which factors are important for predcition
- Problem 3: Which algorithms are best for this dataset

1.2 Preprocess Datasets:

In this step I have first all the loaded datasets given in adult.data.csv which is used for training purpose. I also loaded adult.test.csv which is used for training/validation purpose.

```
In [171]: import numpy as np
          import pandas as pd
          import matplotlib
          import matplotlib.pyplot as plt
          import seaborn as sns
          %matplotlib inline
          import statsmodels.api as sm
          import matplotlib.pyplot as plt
          from patsy import dmatrices
          from sklearn.linear_model import LogisticRegression
          from sklearn.cross_validation import train_test_split
          from sklearn import metrics
          from sklearn.cross_validation import cross_val_score
          from sklearn.metrics import accuracy_score
```

```
In [172]: pd.set_option('display.height', 1000)
          pd.set_option('display.max_rows', 1000)
```

```
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1500)
```

In [173]:
```
# load adult.data.csv into application_train_data dataframe
application_train_data = pd.read_csv('adult.data.csv', header=None)
print('Training data shape:',application_train_data.shape)
```

```
Training data shape: (32561, 15)
```

In [174]:
```
application_train_data.head()
```

Out[174]:

	0	1	2	3	4	5	
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-cle
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-manag
2	38	Private	215646	HS-grad	9	Divorced	Handlers-clea
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-clea
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-spec

In [175]:
```
column_list = ['AGE', 'WORKCLASS', 'FNLWGT', 'EDUCATION', 'EDUCATION_NUM', 'MARITAL_
application_train_data.columns = column_list
application_train_data['TARGET'] = application_train_data['TARGET'].apply(lambda x:
```

In [176]:
```
application_train_data.head()
```

Out[176]:

	AGE	WORKCLASS	FNLWGT	EDUCATION	EDUCATION_NUM	MARITAL_STATUS
0	39	State-gov	77516	Bachelors	13	Never-married
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse
2	38	Private	215646	HS-grad	9	Divorced
3	53	Private	234721	11th	7	Married-civ-spouse
4	28	Private	338409	Bachelors	13	Married-civ-spouse

In [177]:
```
# load adult.test.csv into application_test_data dataframe
application_test_data = pd.read_csv('adult.test.csv', header=None)
print('Test data shape:',application_test_data.shape)
```

```
Test data shape: (16281, 15)
```

In [178]:
```
application_test_data.columns = column_list
application_test_data['TARGET'] = application_test_data['TARGET'].apply(lambda x: 1
```

In [179]:
```
application_test_data.head()
```

Out[179]:

	AGE	WORKCLASS	FNLWGT	EDUCATION	EDUCATION_NUM	MARITAL_STATUS	
0	25	Private	226802	11th	7	Never-married	Machi
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Fa
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Pro
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machi
4	18	?	103497	Some-college	10	Never-married	

1.3 Missing Data Analysis

In this step, we first get which all columns have missing values and then calculate percentage of records which have missing values in each column.

In [180]:
```
application_train_data.isnull().any()
```

Out[180]:
```
AGE               False
WORKCLASS         False
FNLWGT            False
EDUCATION         False
EDUCATION_NUM     False
MARITAL_STATUS    False
OCCUPATION        False
RELATIONSHIP      False
RACE              False
SEX               False
CAPITAL_GAIN      False
CAPITAL_LOSS      False
HOURS_PER_WEEK    False
NATIVE_COUNTRY    False
TARGET            False
dtype: bool
```

In [181]:
```
application_test_data.isnull().any()
```

Out[181]:
```
AGE               False
WORKCLASS         False
FNLWGT            False
EDUCATION         False
EDUCATION_NUM     False
MARITAL_STATUS    False
OCCUPATION        False
RELATIONSHIP      False
RACE              False
SEX               False
CAPITAL_GAIN      False
CAPITAL_LOSS      False
HOURS_PER_WEEK    False
NATIVE_COUNTRY    False
TARGET            False
dtype: bool
```

1.4 Interpretation

As there are no missing value columns, we can skip this

1.5 Analyze the data

1.6 Get statistical parameters of the training and test data

In [14]:
```
application_train_data.describe()
```

Out[14]:

	AGE	FNLWGT	EDUCATION_NUM	CAPITAL_GAIN	CAPITAL_LOSS	HOURS_PE
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99

In [15]:
```
application_test_data.describe()
```

Out[15]:

	AGE	FNLWGT	EDUCATION_NUM	CAPITAL_GAIN	CAPITAL_LOSS	HOURS_PE
count	16281.000000	1.628100e+04	16281.000000	16281.000000	16281.000000	16281
mean	38.767459	1.894357e+05	10.072907	1081.905104	87.899269	40
std	13.849187	1.057149e+05	2.567545	7583.935968	403.105286	12
min	17.000000	1.349200e+04	1.000000	0.000000	0.000000	1
25%	28.000000	1.167360e+05	9.000000	0.000000	0.000000	40
50%	37.000000	1.778310e+05	10.000000	0.000000	0.000000	40
75%	48.000000	2.383840e+05	12.000000	0.000000	0.000000	45
max	90.000000	1.490400e+06	16.000000	99999.000000	3770.000000	99

1.7 Interpretation:

From the statistics parameters, mean of fnlwght is very high value 1.894357e+05, so we need log transformation

1.8 Get the event rate

Event rate percentage is calculated by dividing number of 1 in INCOME field by total number of records multiplied by 100

In [16]:
```
event_rate = (sum(application_train_data.loc[application_train_data['TARGET']==1,'TARG
print("Event_Rate: " + str(event_rate) + "%")
```

```
Event_Rate: 24.080955744602438%
```

1.9 Interpretation:

From the Event Rate, it is clear that target income is imbalanced, hence we need to consider recall, precision in addition to accuracy

1.10 Analyze WORKCLASS vs TARGET

- Create count of each type of WORKCLASS
- Create a cross-tabulation bar plot between WORKCLASS vs TARGET

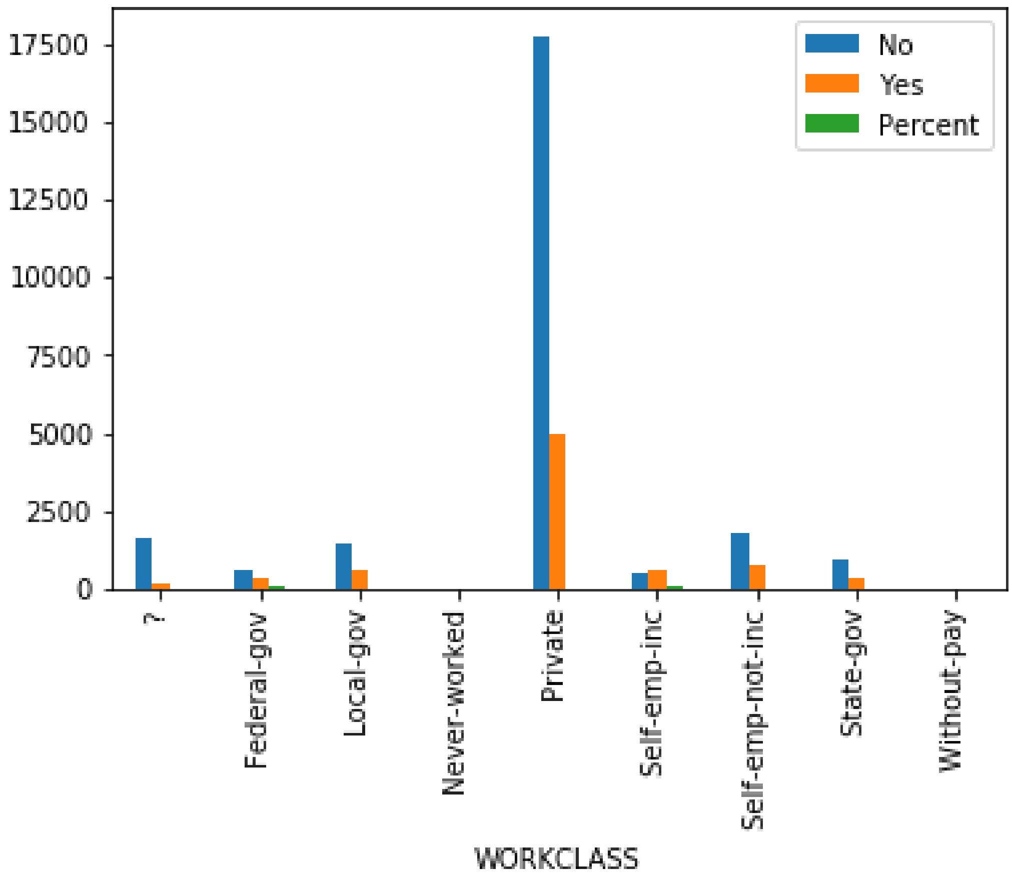
In [17]:
```
application_train_data['WORKCLASS'].value_counts()
```

Out[17]:
```
 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 [18]:
```
tab = pd.crosstab(index=application_train_data['WORKCLASS'],columns=application_train_
tab.columns = ['No','Yes']
tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
print(tab)
tab.plot(kind='bar')
```

```
                      No   Yes    Percent
WORKCLASS
 ?                  1645   191  10.403050
 Federal-gov         589   371  38.645833
 Local-gov          1476   617  29.479216
 Never-worked          7     0   0.000000
 Private           17733  4963  21.867289
 Self-emp-inc        494   622  55.734767
 Self-emp-not-inc   1817   724  28.492719
 State-gov           945   353  27.195686
 Without-pay          14     0   0.000000
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1cee3c79278>
```



1.11 Interpretation:

In terms of absolute numbers Private workclass have most number of people (4963) having > 50K income. However, percentage wise 21 percent of total private employees have mre tahn 50K income

% of Yes of Private employed 21.86% dividing by event rate, lift value = 21.86/24.08 = 0.91
% of Yes Self-emp-inc is 55.73 divinding by event rate, lift value = 55.73/24.08 = 2.31
% of Yes of Federal-gov is 38.65, dividing by event rate, lift value = 38.65/24.08 = 1.60
% of Yes of Local-gov is 29.48 dividing by event rate, lift value =29.48/24.08= 1.22
From the lift values, workcalss could be significant

1.12 Analyze EDUCATION vs TARGET

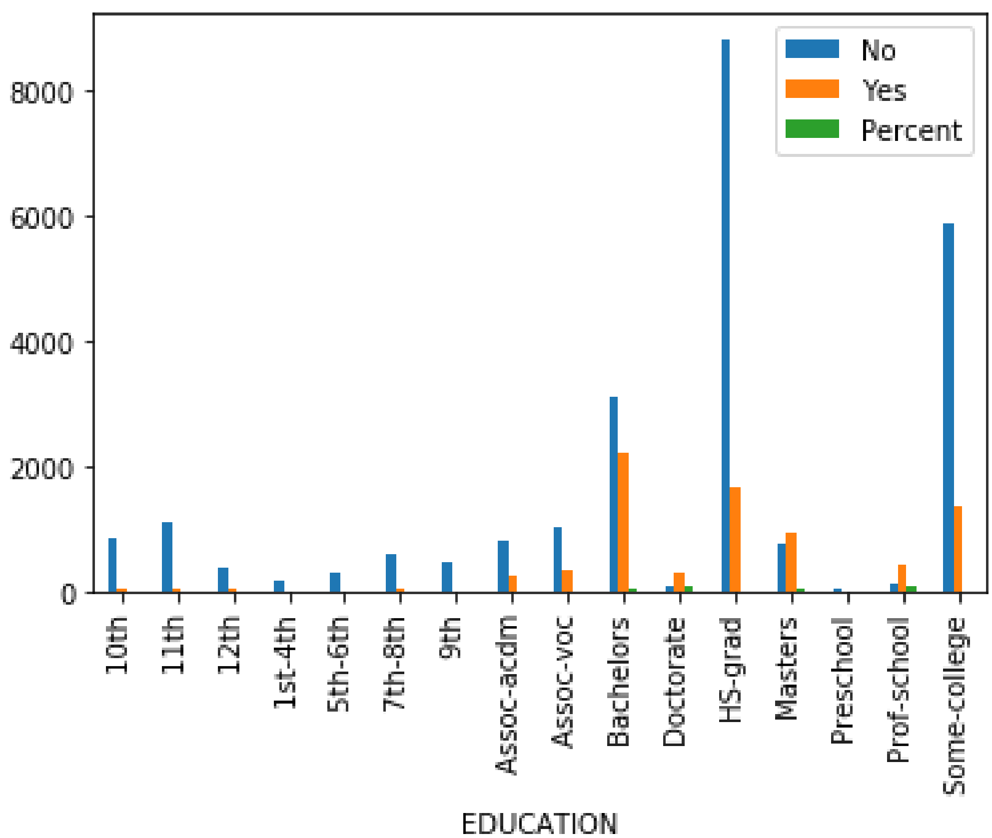
Create count of each type of EDUCATION Create a cross-tabulation bar plot between WORK-CLASS vs TARGET

```
In [19]: tab = pd.crosstab(index=application_train_data['EDUCATION'],columns=application_train_
         tab.columns = ['No','Yes']
         tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
```

```python
print(tab)
tab.plot(kind='bar')
```

```
                No   Yes    Percent
EDUCATION
 10th          871    62   6.645230
 11th         1115    60   5.106383
 12th          400    33   7.621247
 1st-4th       162     6   3.571429
 5th-6th       317    16   4.804805
 7th-8th       606    40   6.191950
 9th           487    27   5.252918
 Assoc-acdm    802   265  24.835989
 Assoc-voc    1021   361  26.121563
 Bachelors    3134  2221  41.475257
 Doctorate     107   306  74.092010
 HS-grad      8826  1675  15.950862
 Masters       764   959  55.658735
 Preschool      51     0   0.000000
 Prof-school   153   423  73.437500
 Some-college 5904  1387  19.023454
```

```
Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1cee41e2240>
```



1.13 Interpretation

In terms of absoulute numbers bachelors degree have highest number of persons having > 50K income. In terms of percentage, doctorates have highest percentage of people having > 50K
% of Yes of Doctorate 74.09 dividing by event rate, lift value = 74.09/24.08 = 3.07
% of Yes Masters is 55.65 divinding by event rate, lift value = 55.65/24.08 = 2.31
% of Yes of Bachelors is 41.48, dividing by event rate, lift value = 41.48/24.08 = 1.72
% of Yes of HS-grad is 15.95 dividing by event rate, lift value = 15.95/24.08= 0.66
From the lift values, education could be significant

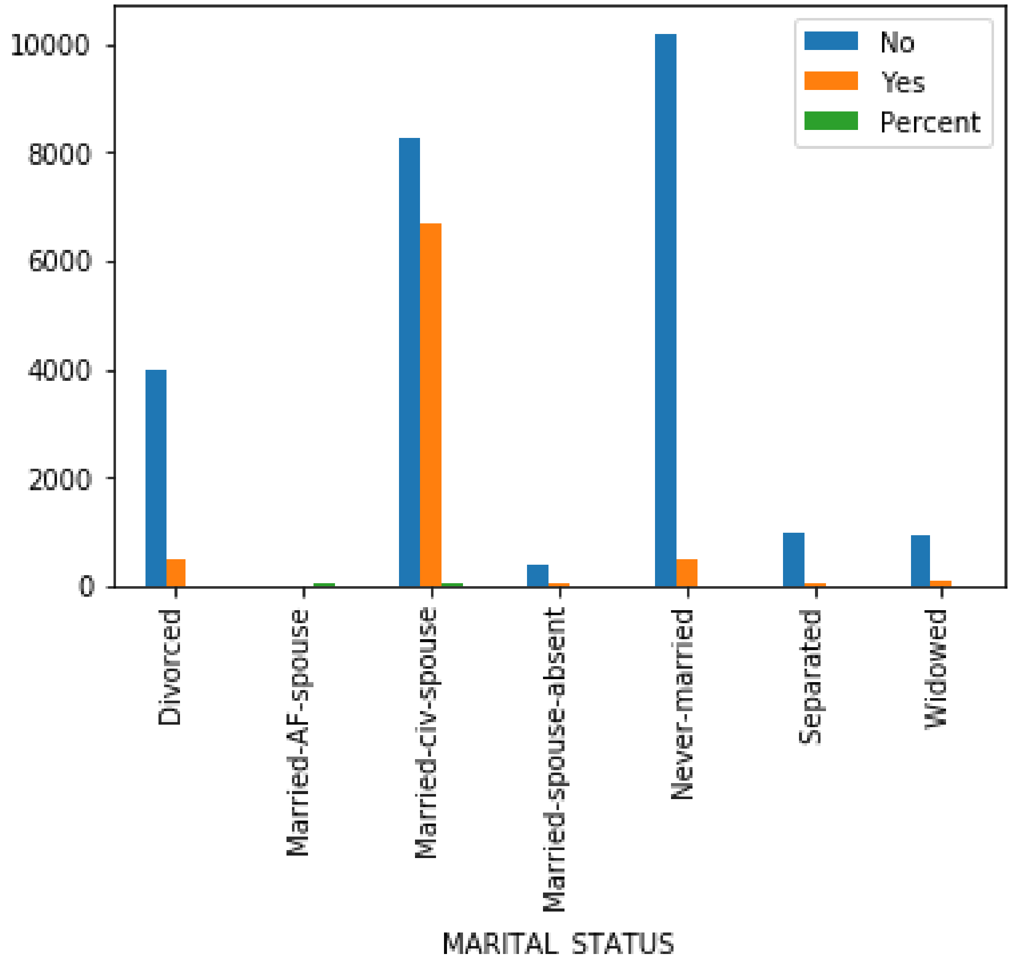
1.14 Analyze MARITAL_STATUS vs TARGET

Create count of each type of MARITAL_STATUS Create a cross-tabulation bar plot between MARITAL_STATUS vs TARGET

```
In [20]: tab = pd.crosstab(index=application_train_data['MARITAL_STATUS'],columns=application_
         tab.columns = ['No','Yes']
         tab['Percent'] = tab.Yes/(tab.No+tab.Yes) * 100
         print(tab)
         tab.plot(kind='bar')
```

```
                          No   Yes    Percent
MARITAL_STATUS
 Divorced               3980   463  10.420887
 Married-AF-spouse        13    10  43.478261
 Married-civ-spouse     8284  6692  44.684829
 Married-spouse-absent   384    34   8.133971
 Never-married         10192   491   4.596087
 Separated               959    66   6.439024
 Widowed                 908    85   8.559919
```

```
Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1cee42452b0>
```



Interpretation:

% of Yes of Married-civ-spouse 44.68 dividing by event rate, lift value = 44.68/24.08 = 1.86 % of Yes of Never-married 4.60 dividing by event rate, lift value = 4.60/24.08 = 0.19 % of Yes of Divorced 10.42 dividing by event rate, lift value = 10.42/24.08 = 0.43

From the lift values MARITAL_STATUS is a significant feature in determining person having income > 50K

1.15 Linear correlation analysis of fields:

TARGET, CAPITAL_GAIN, CAPITAL_LOSS, HOURS_PER_WEEK

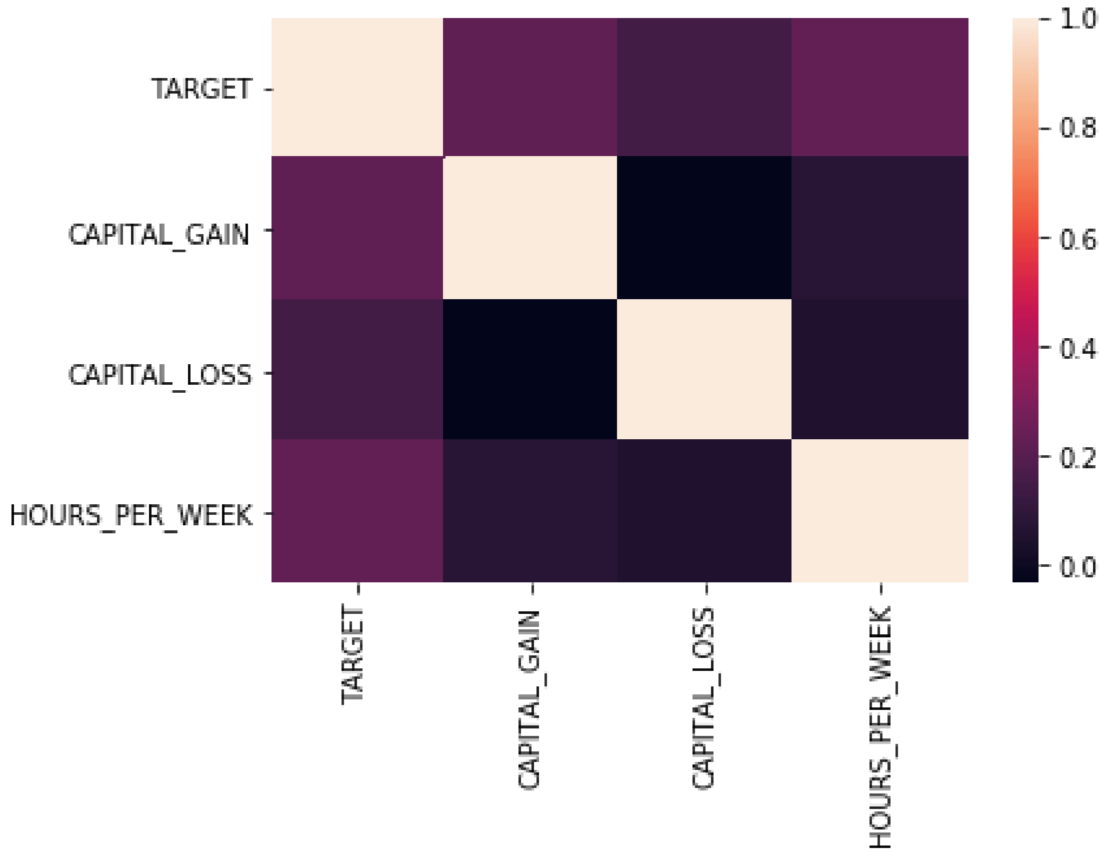
- First calculate correlation coefficinets
- Draw the heatmap

```
In [21]: cor = application_train_data[['TARGET','CAPITAL_GAIN','CAPITAL_LOSS','HOURS_PER_WEEK']
         print( "Correlation coefficients are:")
         print(str(cor))
```

```
sns.heatmap(cor)

Correlation coefficients are:
                  TARGET  CAPITAL_GAIN  CAPITAL_LOSS  HOURS_PER_WEEK
TARGET          1.000000      0.223329      0.150526        0.229689
CAPITAL_GAIN    0.223329      1.000000     -0.031615        0.078409
CAPITAL_LOSS    0.150526     -0.031615      1.000000        0.054256
HOURS_PER_WEEK  0.229689      0.078409      0.054256        1.000000
```

```
Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x1cee433fcc0>
```



1.16 Interpretation:

As all the correlation coefficients are low value, the fields TARGET, CAPITAL_GAIN, CAPITAL_LOSS, HOURS_PER_WEEK do not have correlation

1.17 Linear correlation analysis of fields:

TARGET, AGE, FNLWGT, EDUCATION_NUM

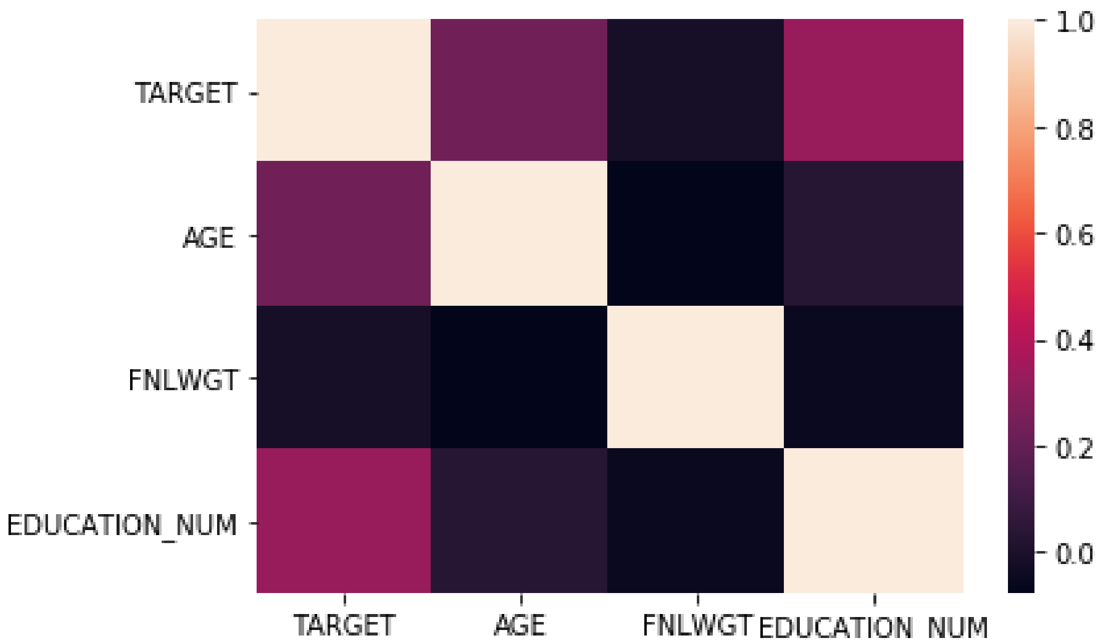
- First calculate correlation coefficinets
- Draw the heatmap

```
In [22]: cor = application_train_data[['TARGET','AGE','FNLWGT','EDUCATION_NUM']].corr()
         print( "Correlation coefficients are:")
         print(str(cor))

         sns.heatmap(cor)
```

```
Correlation coefficients are:
                 TARGET       AGE    FNLWGT  EDUCATION_NUM
TARGET         1.000000  0.234037 -0.009463       0.335154
AGE            0.234037  1.000000 -0.076646       0.036527
FNLWGT        -0.009463 -0.076646  1.000000      -0.043195
EDUCATION_NUM  0.335154  0.036527 -0.043195       1.000000
```

```
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1cee4403978>
```



1.18 Interpretation:

From the heatmap it is clear that EDUCATION_NUM there is some corrrelation to TARGET

1.19 Field Transformations

i. Logarithmic Transformation: For highly-skewed feature distributions such as FNLWGT, CAPITAL_GAIN CAPITAL_LOSS, logarithmic transformation is done on the data so that the very large and very small values do not negatively affect the performance of a learning algorithm. Using a logarithmic transformation significantly reduces the range of values caused by outliers. Care must be taken when applying this transformation however: The logarithm of 0 is undefined, so we must translate the values by a small amount above 0 to apply the the logarithm successfully.

ii. Normalizing Numerical Features

In addition to performing transformations on features that are highly skewed, it is often good practice to perform some type of scaling on numerical features. Applying a scaling to the data does not change the shape of each feature's distribution (such as EDUCATION_NUM, AGE above); however, normalization ensures that each feature is treated equally when applying supervised learners. Note that once scaling is applied, observing the data in its raw form will no longer have the same original meaning, as exampled below.

iii.One hot encoding for categorical features Categorical variables having more than two possible vlaues are encoded using the one-hot encoding scheme. One-hot encoding creates a "dummy" variable for each possible category of each non-numeric feature. For example, assume someFeature has three possible entries: A, B, or C. We then encode this feature into someFeature_A, someFeature_B and someFeature_C.

iv. Label Encoding: Categorical variables having more than two possible are encoded using Label Encode to have values 0 and 1

In [182]:
```
# Perform log transformation
log_transform_fields = [ 'FNLWGT', 'CAPITAL_GAIN', 'CAPITAL_LOSS']

train_data = pd.DataFrame(data = application_train_data)
train_data[log_transform_fields] = application_train_data[log_transform_fields].apply

test_data = pd.DataFrame(data = application_test_data)
test_data[log_transform_fields] = application_test_data[log_transform_fields].apply(
```

In [183]:
```
from sklearn.preprocessing import MinMaxScaler

# Initialize a scaler, then apply it to the features
scaler = MinMaxScaler() # default=(0, 1)

numerical = ['AGE','FNLWGT','EDUCATION_NUM', 'CAPITAL_GAIN', 'CAPITAL_LOSS']

temp1 = pd.DataFrame(data = train_data)
temp1[numerical] = scaler.fit_transform( train_data[numerical])
train_data = temp1

temp2 = pd.DataFrame(data = test_data)
temp2[numerical] = scaler.fit_transform( test_data[numerical])
test_data = temp2
```

In [184]:
```
train_data.head()
```

Out[184]:

	AGE	WORKCLASS	FNLWGT	EDUCATION	EDUCATION_NUM	MARITAL_STA
0	0.301370	State-gov	0.384197	Bachelors	0.800000	Never-mar
1	0.452055	Self-emp-not-inc	0.399234	Bachelors	0.800000	Married-civ-spo
2	0.287671	Private	0.597596	HS-grad	0.533333	Divo
3	0.493151	Private	0.615275	11th	0.400000	Married-civ-spo
4	0.150685	Private	0.691582	Bachelors	0.800000	Married-civ-spo

In [26]:
```
import statsmodels.api as sm
import scipy.stats
from patsy import dmatrices



# create dataframes with an intercept column and dummy variables for
# WORKCLASS, MARITAL_STATUS,  OCCUPATION, RELATIONSHIP, RACE, SEX, NATIVE_COUNTRY
Y, X = dmatrices('TARGET ~ AGE + C(WORKCLASS) + FNLWGT +  C(EDUCATION) \
                  + EDUCATION_NUM + C(MARITAL_STATUS) + C(OCCUPATION) + C(RELATIONSHI
                  + C(RACE) + C(SEX) + CAPITAL_GAIN + CAPITAL_LOSS + HOURS_PER_WEEK +
                  train_data, return_type="dataframe")
X.columns
```

Out[26]:
```
Index(['Intercept', 'C(WORKCLASS)[T. Federal-gov]', 'C(WORKCLASS)[T. Local-gov]', 'C(W
       ...
       'C(NATIVE_COUNTRY)[T. Trinadad&Tobago]', 'C(NATIVE_COUNTRY)[T. United-States]'
```

In [27]:
```
X = X.rename(columns = {
'C(WORKCLASS)[T. Federal-gov]':'WORKCLASS_Federal_gov',
'C(WORKCLASS)[T. Local-gov]' : 'WORKCLASS_Local_gov',
'C(WORKCLASS)[T. Never-worked]' : 'WORKCLASS_Never_worked',
'C(WORKCLASS)[T. Private]' :  'WORKCLASS_Private',
'C(WORKCLASS)[T. Self-emp-inc]' : 'WORKCLASS_Self_emp_inc',
'C(WORKCLASS)[T. Self-emp-not-inc]' : 'WORKCLASS_Self_emp_not_inc',
'C(WORKCLASS)[T. State-gov]' :    'WORKCLASS_State_gov',
'C(WORKCLASS)[T. Without-pay]' :  'WORKCLASS_Without_pay',
'C(EDUCATION)[T. 11th]' :  'EDUCATION_11th',
'C(EDUCATION)[T. 12th]' :   'EDUCATION_12th',
'C(EDUCATION)[T. 1st-4th]' : 'EDUCATION_1st_4th',
'C(EDUCATION)[T. 5th-6th]' : 'EDUCATION_5th_6th',
'C(EDUCATION)[T. 7th-8th]' : 'EDUCATION_7th_8th',
'C(EDUCATION)[T. 9th]' :     'EDUCATION_9th',
'C(EDUCATION)[T. Assoc-acdm]' : 'EDUCATION_Assoc_acdm',
'C(EDUCATION)[T. Assoc-voc]' :   'EDUCATION_voc',
'C(EDUCATION)[T. Bachelors]' :   'EDUCATION_Bachelors',
'C(EDUCATION)[T. Doctorate]' :   'EDUCATION_Doctorate',
'C(EDUCATION)[T. HS-grad]' :     'EDUCATION_HS_grad',
'C(EDUCATION)[T. Masters]' :     'EDUCATION_Masters',
'C(EDUCATION)[T. Preschool]' :   'EDUCATION_Preschool',
'C(EDUCATION)[T. Prof-school]' : 'EDUCATION_Prof_school',
```

```
'C(EDUCATION)[T. Some-college]' : 'EDUCATION_Some_college',
'C(MARITAL_STATUS)[T. Married-AF-spouse]' : 'MARITAL_STATUS_Married_AF_spouse',
'C(MARITAL_STATUS)[T. Married-civ-spouse]' : 'MARITAL_STATUS_Married_civ_spouse',
'C(MARITAL_STATUS)[T. Married-spouse-absent]' : 'MARITAL_STATUS_spouse_absent',
'C(MARITAL_STATUS)[T. Never-married]' :  'MARITAL_STATUS_Never_married',
'C(MARITAL_STATUS)[T. Separated]' :   'MARITAL_STATUS_Separated',
'C(MARITAL_STATUS)[T. Widowed]' :      'MARITAL_ Widowed',
'C(OCCUPATION)[T. Adm-clerical]' :     'OCCUPATION_Adm_clerical',
'C(OCCUPATION)[T. Armed-Forces]' :     'OCCUPATION_Armed-Forces',
'C(OCCUPATION)[T. Craft-repair]' :     'OCCUPATION_Craft_repair',
'C(OCCUPATION)[T. Exec-managerial]' :  'OCCUPATION_ Exec_managerial',
'C(OCCUPATION)[T. Farming-fishing]' :  'OCCUPATION_Farming_fishing',
'C(OCCUPATION)[T. Handlers-cleaners]' : 'OCCUPATION_Handlers_cleaners',
'C(OCCUPATION)[T. Machine-op-inspct]' : 'OCCUPATION_Machine_op_inspct',
'C(OCCUPATION)[T. Other-service]' :   'OCCUPATION_Other_service',
'C(OCCUPATION)[T. Priv-house-serv]' :  'OCCUPATION_Priv_house_serv',
'C(OCCUPATION)[T. Prof-specialty]' :   'OCCUPATION_Prof_specialty',
'C(OCCUPATION)[T. Protective-serv]' :   'OCCUPATION_Protective_serv',
'C(OCCUPATION)[T. Sales]' :           'OCCUPATION_Sales',
'C(OCCUPATION)[T. Tech-support]' :     'OCCUPATION_Tech_support',
'C(OCCUPATION)[T. Transport-moving]' :  'OCCUPATION_Transport_moving',
'C(RELATIONSHIP)[T. Not-in-family]' :  'RELATIONSHIP_Not_in_family',
'C(RELATIONSHIP)[T. Other-relative]' :  'RELATIONSHIP_Other_relative',
'C(RELATIONSHIP)[T. Own-child]' :  'RELATIONSHIP_Own_child',
'C(RELATIONSHIP)[T. Unmarried]' :   'RELATIONSHIP_Unmarried',
'C(RELATIONSHIP)[T. Wife]' :    'RELATIONSHIP_Wife',
'C(RACE)[T. Asian-Pac-Islander]' : 'RACE_Asian_Pac_Islander',
'C(RACE)[T. Black]' :   'RACE_Black',
'C(RACE)[T. Other]' :   'RACE_Other',
'C(RACE)[T. White]' :   'RACE_White',
'C(SEX)[T. Male]' :     'SEX_Male',
'C(NATIVE_COUNTRY)[T. Cambodia]' :  'NATIVE_COUNTRY_Cambodia',
'C(NATIVE_COUNTRY)[T. Canada]' :   'NATIVE_COUNTRY_Canada',
'C(NATIVE_COUNTRY)[T. China]' :   'NATIVE_COUNTRY_China',
'C(NATIVE_COUNTRY)[T. Columbia]' : 'NATIVE_COUNTRY_Columbia',
'C(NATIVE_COUNTRY)[T. Cuba]' :     'NATIVE_COUNTRY_Cuba',
'C(NATIVE_COUNTRY)[T. Dominican-Republic]' :  'NATIVE_COUNTRY_Dominican_Republic',
'C(NATIVE_COUNTRY)[T. Ecuador]' :   'NATIVE_COUNTRY_Ecuador',
'C(NATIVE_COUNTRY)[T. El-Salvador]' :  'NATIVE_COUNTRY_El-Salvador',
'C(NATIVE_COUNTRY)[T. England]' :  'NATIVE_COUNTRY_England',
'C(NATIVE_COUNTRY)[T. France]' :   'NATIVE_COUNTRY_France',
'C(NATIVE_COUNTRY)[T. Germany]' :   'NATIVE_COUNTRY_Germany',
'C(NATIVE_COUNTRY)[T. Greece]' :    'NATIVE_COUNTRY_Greece',
'C(NATIVE_COUNTRY)[T. Guatemala]' :  'NATIVE_COUNTRY_Guatemala',
'C(NATIVE_COUNTRY)[T. Haiti]' :       'NATIVE_COUNTRY_Haiti',
'C(NATIVE_COUNTRY)[T. Holand-Netherlands]' :  'NATIVE_COUNTRY_Holand-Netherlands',
'C(NATIVE_COUNTRY)[T. Honduras]' :   'NATIVE_COUNTRY_Honduras',
'C(NATIVE_COUNTRY)[T. Hong]' :    'NATIVE_COUNTRY_Hong',
```

```
'C(NATIVE_COUNTRY)[T. Hungary]' :  'NATIVE_COUNTRY_Hungary',
'C(NATIVE_COUNTRY)[T. India]' :   'NATIVE_COUNTRY_India',
'C(NATIVE_COUNTRY)[T. Iran]' :     'NATIVE_COUNTRY_Iran',
'C(NATIVE_COUNTRY)[T. Ireland]' :  'NATIVE_COUNTRY_Ireland',
'C(NATIVE_COUNTRY)[T. Italy]' :   'NATIVE_COUNTRY_Italy',
'C(NATIVE_COUNTRY)[T. Jamaica]' :  'NATIVE_COUNTRY_Jamaica',
'C(NATIVE_COUNTRY)[T. Japan]' :   'NATIVE_COUNTRY_Japan',
'C(NATIVE_COUNTRY)[T. Laos]' :    'NATIVE_COUNTRY_Laos',
'C(NATIVE_COUNTRY)[T. Mexico]' :   'NATIVE_COUNTRY_Mexico',
'C(NATIVE_COUNTRY)[T. Nicaragua]' : 'NATIVE_COUNTRY_Nicaragua',
'C(NATIVE_COUNTRY)[T. Outlying-US(Guam-USVI-etc)]' : 'NATIVE_COUNTRY_Outlying-US_Guam_
'C(NATIVE_COUNTRY)[T. Peru]' :   'NATIVE_COUNTRY_Peru',
'C(NATIVE_COUNTRY)[T. Philippines]' : 'NATIVE_COUNTRY_Philippines',
'C(NATIVE_COUNTRY)[T. Poland]' :    'NATIVE_COUNTRY_Poland',
'C(NATIVE_COUNTRY)[T. Portugal]' :   'NATIVE_COUNTRY_Portugal',
'C(NATIVE_COUNTRY)[T. Puerto-Rico]' : 'NATIVE_COUNTRY_Puerto_Rico',
'C(NATIVE_COUNTRY)[T. Scotland]' : 'NATIVE_COUNTRY_Scotland',
'C(NATIVE_COUNTRY)[T. South]' :  'NATIVE_COUNTRY_South',
'C(NATIVE_COUNTRY)[T. Taiwan]' :  'NATIVE_COUNTRY_Taiwan',
'C(NATIVE_COUNTRY)[T. Thailand]' :     'NATIVE_COUNTRY_Thailand',
'C(NATIVE_COUNTRY)[T. Trinadad&Tobago]' :  'NATIVE_COUNTRY_Trinadad_Tobago',
'C(NATIVE_COUNTRY)[T. United-States]' :   'NATIVE_COUNTRY_United_States',
'C(NATIVE_COUNTRY)[T. Vietnam]' :     'NATIVE_COUNTRY_Vietnam',
'C(NATIVE_COUNTRY)[T. Yugoslavia]' :   'NATIVE_COUNTRY_Yugoslavia'
})
```

In [28]:
```
X.head()
```

Out[28]:

	Intercept	WORKCLASS_Federal_gov	WORKCLASS_Local_gov	WORKCLASS_Never_worked	WORI
0	1.0	0.0	0.0	0.0	
1	1.0	0.0	0.0	0.0	
2	1.0	0.0	0.0	0.0	
3	1.0	0.0	0.0	0.0	
4	1.0	0.0	0.0	0.0	

	NATIVE_COUNTRY_Germany	NATIVE_COUNTRY_Greece	NATIVE_COUNTRY_Guatemala	NATIVE_COU
0	0.0	0.0	0.0	
1	0.0	0.0	0.0	
2	0.0	0.0	0.0	
3	0.0	0.0	0.0	
4	0.0	0.0	0.0	

In [29]:
```
Y
```

Out[29]:

	TARGET
0	0.0
1	0.0
2	0.0
3	0.0

4	0.0
5	0.0
6	0.0
7	1.0
8	1.0
9	1.0
10	1.0
11	1.0
12	0.0
13	0.0
14	1.0
15	0.0
16	0.0
17	0.0
18	0.0
19	1.0
20	1.0
21	0.0
22	0.0
23	0.0
24	0.0
25	1.0
26	0.0
27	1.0
28	0.0
29	0.0
30	0.0
31	0.0
32	0.0
33	0.0
34	0.0
35	0.0
36	0.0
37	0.0
38	1.0
39	0.0
40	0.0
41	0.0
42	0.0
43	0.0
44	0.0
45	1.0
46	0.0
47	0.0
48	0.0
49	0.0
50	0.0
51	0.0

52	1.0
53	1.0
54	0.0
55	1.0
56	0.0
57	0.0
58	0.0
59	0.0
60	0.0
61	0.0
62	0.0
63	1.0
64	0.0
65	0.0
66	0.0
67	1.0
68	1.0
69	0.0
70	0.0
71	0.0
72	1.0
73	0.0
74	0.0
75	0.0
76	0.0
77	0.0
78	0.0
79	0.0
80	0.0
81	0.0
82	0.0
83	0.0
84	1.0
85	0.0
86	1.0
87	0.0
88	0.0
89	1.0
90	0.0
91	0.0
92	0.0
93	0.0
94	1.0
95	0.0
96	1.0
97	1.0
98	0.0
99	0.0

100	1.0
101	1.0
102	0.0
103	0.0
104	0.0
105	1.0
106	0.0
107	0.0
108	0.0
109	0.0
110	0.0
111	1.0
112	1.0
113	0.0
114	0.0
115	0.0
116	0.0
117	1.0
118	0.0
119	0.0
120	0.0
121	0.0
122	0.0
123	1.0
124	0.0
125	1.0
126	0.0
127	1.0
128	0.0
129	0.0
130	0.0
131	0.0
132	0.0
133	0.0
134	0.0
135	1.0
136	0.0
137	0.0
138	0.0
139	1.0
140	0.0
141	0.0
142	0.0
143	1.0
144	0.0
145	0.0
146	0.0
147	0.0

148	0.0
149	0.0
150	0.0
151	0.0
152	0.0
153	0.0
154	1.0
155	0.0
156	0.0
157	0.0
158	0.0
159	0.0
160	0.0
161	0.0
162	0.0
163	0.0
164	1.0
165	0.0
166	0.0
167	0.0
168	0.0
169	0.0
170	0.0
171	0.0
172	1.0
173	0.0
174	1.0
175	0.0
176	0.0
177	0.0
178	0.0
179	0.0
180	1.0
181	0.0
182	0.0
183	1.0
184	1.0
185	0.0
186	0.0
187	0.0
188	0.0
189	1.0
190	0.0
191	0.0
192	0.0
193	0.0
194	0.0
195	0.0

196	0.0
197	1.0
198	1.0
199	0.0
200	0.0
201	0.0
202	1.0
203	0.0
204	0.0
205	0.0
206	1.0
207	0.0
208	1.0
209	0.0
210	0.0
211	1.0
212	0.0
213	0.0
214	1.0
215	1.0
216	0.0
217	0.0
218	0.0
219	0.0
220	0.0
221	0.0
222	0.0
223	0.0
224	0.0
225	0.0
226	0.0
227	0.0
228	0.0
229	0.0
230	0.0
231	0.0
232	0.0
233	0.0
234	1.0
235	0.0
236	0.0
237	1.0
238	1.0
239	0.0
240	1.0
241	0.0
242	0.0
243	0.0

244	0.0
245	0.0
246	1.0
247	0.0
248	1.0
249	0.0
250	1.0
251	0.0
252	0.0
253	0.0
254	0.0
255	1.0
256	0.0
257	0.0
258	0.0
259	0.0
260	0.0
261	0.0
262	0.0
263	0.0
264	0.0
265	1.0
266	0.0
267	1.0
268	0.0
269	1.0
270	1.0
271	0.0
272	0.0
273	0.0
274	0.0
275	0.0
276	1.0
277	0.0
278	0.0
279	1.0
280	0.0
281	1.0
282	0.0
283	0.0
284	0.0
285	1.0
286	1.0
287	0.0
288	0.0
289	0.0
290	0.0
291	0.0

292	0.0
293	0.0
294	0.0
295	0.0
296	0.0
297	0.0
298	0.0
299	0.0
300	1.0
301	0.0
302	1.0
303	1.0
304	0.0
305	0.0
306	0.0
307	0.0
308	1.0
309	1.0
310	0.0
311	1.0
312	0.0
313	1.0
314	0.0
315	0.0
316	0.0
317	0.0
318	0.0
319	0.0
320	0.0
321	0.0
322	0.0
323	0.0
324	0.0
325	0.0
326	0.0
327	1.0
328	0.0
329	0.0
330	0.0
331	0.0
332	0.0
333	0.0
334	0.0
335	0.0
336	0.0
337	0.0
338	0.0
339	0.0

340	0.0
341	1.0
342	1.0
343	0.0
344	0.0
345	0.0
346	0.0
347	0.0
348	0.0
349	0.0
350	1.0
351	0.0
352	1.0
353	0.0
354	1.0
355	1.0
356	0.0
357	1.0
358	0.0
359	0.0
360	0.0
361	1.0
362	0.0
363	1.0
364	0.0
365	0.0
366	0.0
367	0.0
368	1.0
369	0.0
370	0.0
371	0.0
372	1.0
373	0.0
374	0.0
375	0.0
376	0.0
377	0.0
378	0.0
379	0.0
380	0.0
381	0.0
382	0.0
383	0.0
384	0.0
385	0.0
386	0.0
387	1.0

388	0.0
389	0.0
390	1.0
391	0.0
392	0.0
393	1.0
394	0.0
395	0.0
396	0.0
397	0.0
398	1.0
399	1.0
400	0.0
401	0.0
402	1.0
403	0.0
404	0.0
405	1.0
406	0.0
407	0.0
408	1.0
409	0.0
410	0.0
411	0.0
412	0.0
413	1.0
414	0.0
415	1.0
416	0.0
417	0.0
418	0.0
419	0.0
420	0.0
421	0.0
422	1.0
423	0.0
424	0.0
425	0.0
426	0.0
427	0.0
428	1.0
429	0.0
430	0.0
431	0.0
432	0.0
433	1.0
434	0.0
435	1.0

436	0.0
437	0.0
438	0.0
439	0.0
440	0.0
441	0.0
442	0.0
443	1.0
444	0.0
445	0.0
446	0.0
447	0.0
448	0.0
449	0.0
450	0.0
451	1.0
452	1.0
453	1.0
454	1.0
455	1.0
456	0.0
457	0.0
458	0.0
459	0.0
460	0.0
461	0.0
462	0.0
463	0.0
464	0.0
465	0.0
466	0.0
467	0.0
468	1.0
469	1.0
470	1.0
471	0.0
472	0.0
473	0.0
474	0.0
475	0.0
476	0.0
477	0.0
478	0.0
479	0.0
480	0.0
481	0.0
482	0.0
483	0.0

```
484         0.0
485         0.0
486         0.0
487         0.0
488         0.0
489         1.0
490         0.0
491         0.0
492         0.0
493         0.0
494         0.0
495         0.0
496         0.0
497         0.0
498         0.0
499         0.0
...         ...
32061       0.0
32062       0.0
32063       1.0
32064       0.0
32065       0.0
32066       0.0
32067       1.0
32068       0.0
32069       0.0
32070       0.0
32071       0.0
32072       0.0
32073       0.0
32074       0.0
32075       0.0
32076       1.0
32077       1.0
32078       0.0
32079       1.0
32080       0.0
32081       1.0
32082       0.0
32083       0.0
32084       1.0
32085       0.0
32086       0.0
32087       0.0
32088       0.0
32089       0.0
32090       1.0
32091       0.0
```

32092	1.0
32093	0.0
32094	0.0
32095	0.0
32096	0.0
32097	0.0
32098	1.0
32099	0.0
32100	0.0
32101	0.0
32102	1.0
32103	0.0
32104	0.0
32105	0.0
32106	0.0
32107	0.0
32108	0.0
32109	1.0
32110	0.0
32111	0.0
32112	0.0
32113	0.0
32114	1.0
32115	1.0
32116	0.0
32117	1.0
32118	0.0
32119	1.0
32120	0.0
32121	0.0
32122	0.0
32123	1.0
32124	0.0
32125	1.0
32126	0.0
32127	0.0
32128	0.0
32129	1.0
32130	1.0
32131	1.0
32132	0.0
32133	1.0
32134	0.0
32135	1.0
32136	0.0
32137	0.0
32138	1.0
32139	1.0

32140	0.0
32141	0.0
32142	0.0
32143	0.0
32144	0.0
32145	0.0
32146	0.0
32147	0.0
32148	0.0
32149	0.0
32150	0.0
32151	0.0
32152	0.0
32153	0.0
32154	0.0
32155	0.0
32156	1.0
32157	0.0
32158	0.0
32159	0.0
32160	0.0
32161	1.0
32162	0.0
32163	0.0
32164	1.0
32165	0.0
32166	0.0
32167	0.0
32168	0.0
32169	0.0
32170	0.0
32171	0.0
32172	0.0
32173	1.0
32174	0.0
32175	0.0
32176	0.0
32177	0.0
32178	0.0
32179	1.0
32180	0.0
32181	1.0
32182	0.0
32183	1.0
32184	0.0
32185	1.0
32186	0.0
32187	0.0

32188	0.0
32189	0.0
32190	0.0
32191	0.0
32192	0.0
32193	0.0
32194	0.0
32195	1.0
32196	0.0
32197	0.0
32198	1.0
32199	1.0
32200	0.0
32201	0.0
32202	0.0
32203	1.0
32204	0.0
32205	0.0
32206	0.0
32207	0.0
32208	0.0
32209	0.0
32210	0.0
32211	0.0
32212	0.0
32213	0.0
32214	1.0
32215	0.0
32216	1.0
32217	0.0
32218	0.0
32219	0.0
32220	0.0
32221	1.0
32222	1.0
32223	0.0
32224	0.0
32225	0.0
32226	0.0
32227	0.0
32228	1.0
32229	0.0
32230	1.0
32231	0.0
32232	0.0
32233	0.0
32234	1.0
32235	0.0

32236	1.0
32237	0.0
32238	1.0
32239	0.0
32240	0.0
32241	0.0
32242	0.0
32243	0.0
32244	0.0
32245	1.0
32246	1.0
32247	1.0
32248	0.0
32249	1.0
32250	1.0
32251	0.0
32252	0.0
32253	0.0
32254	1.0
32255	0.0
32256	1.0
32257	0.0
32258	1.0
32259	0.0
32260	1.0
32261	0.0
32262	0.0
32263	0.0
32264	0.0
32265	1.0
32266	1.0
32267	1.0
32268	0.0
32269	0.0
32270	0.0
32271	1.0
32272	1.0
32273	0.0
32274	0.0
32275	0.0
32276	0.0
32277	0.0
32278	0.0
32279	0.0
32280	0.0
32281	0.0
32282	0.0
32283	0.0

32284	0.0
32285	0.0
32286	0.0
32287	1.0
32288	0.0
32289	0.0
32290	0.0
32291	0.0
32292	1.0
32293	0.0
32294	0.0
32295	1.0
32296	0.0
32297	1.0
32298	0.0
32299	0.0
32300	0.0
32301	0.0
32302	1.0
32303	0.0
32304	0.0
32305	1.0
32306	0.0
32307	1.0
32308	1.0
32309	0.0
32310	0.0
32311	1.0
32312	0.0
32313	0.0
32314	0.0
32315	0.0
32316	0.0
32317	0.0
32318	1.0
32319	0.0
32320	0.0
32321	0.0
32322	0.0
32323	0.0
32324	0.0
32325	0.0
32326	1.0
32327	0.0
32328	0.0
32329	0.0
32330	0.0
32331	0.0

32332	0.0
32333	1.0
32334	0.0
32335	0.0
32336	0.0
32337	0.0
32338	0.0
32339	0.0
32340	0.0
32341	1.0
32342	0.0
32343	0.0
32344	0.0
32345	0.0
32346	0.0
32347	1.0
32348	0.0
32349	0.0
32350	0.0
32351	0.0
32352	0.0
32353	0.0
32354	0.0
32355	0.0
32356	0.0
32357	0.0
32358	1.0
32359	0.0
32360	0.0
32361	0.0
32362	0.0
32363	0.0
32364	0.0
32365	1.0
32366	0.0
32367	0.0
32368	0.0
32369	0.0
32370	1.0
32371	1.0
32372	1.0
32373	0.0
32374	0.0
32375	1.0
32376	0.0
32377	0.0
32378	0.0
32379	0.0

32380	0.0
32381	0.0
32382	0.0
32383	0.0
32384	0.0
32385	0.0
32386	1.0
32387	0.0
32388	0.0
32389	1.0
32390	0.0
32391	0.0
32392	0.0
32393	0.0
32394	0.0
32395	0.0
32396	0.0
32397	0.0
32398	0.0
32399	1.0
32400	1.0
32401	0.0
32402	1.0
32403	0.0
32404	0.0
32405	0.0
32406	0.0
32407	0.0
32408	0.0
32409	0.0
32410	0.0
32411	0.0
32412	0.0
32413	0.0
32414	0.0
32415	1.0
32416	1.0
32417	0.0
32418	0.0
32419	1.0
32420	0.0
32421	0.0
32422	0.0
32423	0.0
32424	0.0
32425	0.0
32426	0.0
32427	0.0

32428	0.0
32429	1.0
32430	0.0
32431	0.0
32432	0.0
32433	0.0
32434	0.0
32435	0.0
32436	1.0
32437	0.0
32438	1.0
32439	0.0
32440	1.0
32441	1.0
32442	0.0
32443	0.0
32444	0.0
32445	0.0
32446	0.0
32447	0.0
32448	0.0
32449	1.0
32450	0.0
32451	0.0
32452	0.0
32453	1.0
32454	0.0
32455	1.0
32456	0.0
32457	1.0
32458	1.0
32459	0.0
32460	0.0
32461	0.0
32462	1.0
32463	0.0
32464	1.0
32465	0.0
32466	1.0
32467	0.0
32468	0.0
32469	0.0
32470	0.0
32471	0.0
32472	0.0
32473	0.0
32474	1.0
32475	0.0

32476	0.0
32477	0.0
32478	0.0
32479	0.0
32480	1.0
32481	0.0
32482	0.0
32483	0.0
32484	0.0
32485	0.0
32486	0.0
32487	0.0
32488	0.0
32489	0.0
32490	0.0
32491	0.0
32492	0.0
32493	0.0
32494	0.0
32495	0.0
32496	0.0
32497	0.0
32498	0.0
32499	0.0
32500	0.0
32501	0.0
32502	0.0
32503	0.0
32504	1.0
32505	0.0
32506	1.0
32507	0.0
32508	0.0
32509	0.0
32510	1.0
32511	0.0
32512	0.0
32513	1.0
32514	0.0
32515	0.0
32516	0.0
32517	0.0
32518	1.0
32519	1.0
32520	0.0
32521	0.0
32522	0.0
32523	0.0

```
32524    0.0
32525    0.0
32526    0.0
32527    0.0
32528    0.0
32529    0.0
32530    1.0
32531    0.0
32532    1.0
32533    1.0
32534    0.0
32535    0.0
32536    1.0
32537    0.0
32538    1.0
32539    1.0
32540    0.0
32541    0.0
32542    0.0
32543    0.0
32544    0.0
32545    1.0
32546    0.0
32547    0.0
32548    0.0
32549    0.0
32550    0.0
32551    0.0
32552    0.0
32553    0.0
32554    1.0
32555    0.0
32556    0.0
32557    1.0
32558    0.0
32559    0.0
32560    1.0

[32561 rows x 1 columns]
```

In [30]:
```
# flatten y into a 1-D array
y = np.ravel(Y)
```

In [31]:
```
import statsmodels.discrete.discrete_model as sm
```

In [32]:
```
logit = sm.Logit(y, X)
logit.fit().params
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.327679
```

```
Iterations: 35
```

```
E:\anaconda\lib\site-packages\statsmodels\base\model.py:496: ConvergenceWarning: Maximum Likeli
  "Check mle_retvals", ConvergenceWarning)
```

Out[32]:

```
Intercept                                   -9.595447
WORKCLASS_Federal_gov                        1.865412
WORKCLASS_Local_gov                          1.170437
WORKCLASS_Never_worked                      -4.355701
WORKCLASS_Private                            1.374251
WORKCLASS_Self_emp_inc                       1.579517
WORKCLASS_Self_emp_not_inc                   0.909755
WORKCLASS_State_gov                          1.049001
WORKCLASS_Without_pay                      -21.355073
EDUCATION_11th                              -0.117762
EDUCATION_12th                               0.093894
EDUCATION_1st_4th                            0.300387
EDUCATION_5th_6th                            0.365038
EDUCATION_7th_8th                           -0.160913
EDUCATION_9th                               -0.058430
EDUCATION_Assoc_acdm                         0.095845
EDUCATION_voc                                0.314577
EDUCATION_Bachelors                          0.501487
EDUCATION_Doctorate                          0.936896
EDUCATION_HS_grad                            0.183562
EDUCATION_Masters                            0.654729
EDUCATION_Preschool                        -13.089857
EDUCATION_Prof_school                        0.972347
EDUCATION_Some_college                       0.327718
MARITAL_STATUS_Married_AF_spouse             2.533233
MARITAL_STATUS_Married_civ_spouse            2.074459
MARITAL_STATUS_spouse_absent                -0.000180
MARITAL_STATUS_Never_married                -0.444847
MARITAL_STATUS_Separated                    -0.126701
MARITAL_ Widowed                             0.147044
OCCUPATION_Adm_clerical                     -0.693359
OCCUPATION_Armed-Forces                     -1.868967
OCCUPATION_Craft_repair                     -0.609063
OCCUPATION_ Exec_managerial                  0.104999
OCCUPATION_Farming_fishing                  -1.628142
OCCUPATION_Handlers_cleaners                -1.391866
OCCUPATION_Machine_op_inspct                -1.016729
OCCUPATION_Other_service                    -1.534701
OCCUPATION_Priv_house_serv                  -3.237536
OCCUPATION_Prof_specialty                   -0.170065
OCCUPATION_Protective_serv                  -0.104775
```

OCCUPATION_Sales	-0.406155
OCCUPATION_Tech_support	-0.054774
OCCUPATION_Transport_moving	-0.795559
RELATIONSHIP_Not_in_family	0.494330
RELATIONSHIP_Other_relative	-0.364730
RELATIONSHIP_Own_child	-0.696104
RELATIONSHIP_Unmarried	0.317579
RELATIONSHIP_Wife	1.337373
RACE_Asian_Pac_Islander	0.569655
RACE_Black	0.324126
RACE_Other	0.048644
RACE_White	0.463186
SEX_Male	0.832070
NATIVE_COUNTRY_Cambodia	1.421082
NATIVE_COUNTRY_Canada	0.523793
NATIVE_COUNTRY_China	-0.497925
NATIVE_COUNTRY_Columbia	-2.041456
NATIVE_COUNTRY_Cuba	0.499447
NATIVE_COUNTRY_Dominican_Republic	-0.912492
NATIVE_COUNTRY_Ecuador	-0.083268
NATIVE_COUNTRY_El-Salvador	-0.432740
NATIVE_COUNTRY_England	0.534696
NATIVE_COUNTRY_France	0.723336
NATIVE_COUNTRY_Germany	0.629252
NATIVE_COUNTRY_Greece	-0.758449
NATIVE_COUNTRY_Guatemala	-0.099210
NATIVE_COUNTRY_Haiti	0.059656
NATIVE_COUNTRY_Holand-Netherlands	-31.371527
NATIVE_COUNTRY_Honduras	-1.068044
NATIVE_COUNTRY_Hong	0.069803
NATIVE_COUNTRY_Hungary	-0.184788
NATIVE_COUNTRY_India	-0.239361
NATIVE_COUNTRY_Iran	0.266874
NATIVE_COUNTRY_Ireland	0.603071
NATIVE_COUNTRY_Italy	0.967893
NATIVE_COUNTRY_Jamaica	0.170760
NATIVE_COUNTRY_Japan	0.520462
NATIVE_COUNTRY_Laos	-0.505922
NATIVE_COUNTRY_Mexico	-0.361869
NATIVE_COUNTRY_Nicaragua	-0.569817
NATIVE_COUNTRY_Outlying-US_Guam_USVI_etc	-16.093396
NATIVE_COUNTRY_Peru	-0.632805
NATIVE_COUNTRY_Philippines	0.595483
NATIVE_COUNTRY_Poland	0.153873
NATIVE_COUNTRY_Portugal	0.104827
NATIVE_COUNTRY_Puerto_Rico	-0.150226
NATIVE_COUNTRY_Scotland	0.153128
NATIVE_COUNTRY_South	-0.719521

```
        NATIVE_COUNTRY_Taiwan                      0.186672
        NATIVE_COUNTRY_Thailand                   -0.421411
        NATIVE_COUNTRY_Trinadad_Tobago            -0.213550
        NATIVE_COUNTRY_United_States               0.390228
        NATIVE_COUNTRY_Vietnam                    -1.079569
        NATIVE_COUNTRY_Yugoslavia                  0.864316
        AGE                                        1.882189
        FNLWGT                                     0.613053
        EDUCATION_NUM                              3.100250
        CAPITAL_GAIN                               2.368338
        CAPITAL_LOSS                               1.270714
        HOURS_PER_WEEK                             0.029830
        dtype: float64
```

In [33]:
```
result_ = logit.fit()
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.327679
         Iterations: 35


E:\anaconda\lib\site-packages\statsmodels\base\model.py:496: ConvergenceWarning: Maximum Likel
  "Check mle_retvals", ConvergenceWarning)
```

In [34]:
```
print(result_.summary2())
```

```
                                              Results: Logit
=================================================================================================
Model:                          Logit                        No. Iterations:
Dependent Variable:             y                            Pseudo R-squared:
Date:                           2018-11-15 23:43             AIC:
No. Observations:               32561                        BIC:
Df Model:                       98                           Log-Likelihood:
Df Residuals:                   32462                        LL-Null:
Converged:                      0.0000                       Scale:
-------------------------------------------------------------------------------------------------
                                    Coef.      Std.Err.      z    P>|z|       [0.025
-------------------------------------------------------------------------------------------------
Intercept                          -9.5954          nan      nan    nan            nan
WORKCLASS_Federal_gov               1.8654          nan      nan    nan            nan
WORKCLASS_Local_gov                 1.1704          nan      nan    nan            nan
WORKCLASS_Never_worked             -4.3557      20.5145 -0.2123 0.8319        -44.5635
WORKCLASS_Private                   1.3743          nan      nan    nan            nan
WORKCLASS_Self_emp_inc              1.5795          nan      nan    nan            nan
WORKCLASS_Self_emp_not_inc          0.9098          nan      nan    nan            nan
WORKCLASS_State_gov                 1.0490          nan      nan    nan            nan
WORKCLASS_Without_pay             -21.3551          nan      nan    nan            nan
EDUCATION_11th                     -0.1178          nan      nan    nan            nan
```

EDUCATION_12th	0.0939	nan	nan	nan	nan
EDUCATION_1st_4th	0.3004	nan	nan	nan	nan
EDUCATION_5th_6th	0.3650	nan	nan	nan	nan
EDUCATION_7th_8th	-0.1609	nan	nan	nan	nan
EDUCATION_9th	-0.0584	nan	nan	nan	nan
EDUCATION_Assoc_acdm	0.0958	nan	nan	nan	nan
EDUCATION_voc	0.3146	nan	nan	nan	nan
EDUCATION_Bachelors	0.5015	nan	nan	nan	nan
EDUCATION_Doctorate	0.9369	nan	nan	nan	nan
EDUCATION_HS_grad	0.1836	nan	nan	nan	nan
EDUCATION_Masters	0.6547	nan	nan	nan	nan
EDUCATION_Preschool	-13.0899	nan	nan	nan	nan
EDUCATION_Prof_school	0.9723	nan	nan	nan	nan
EDUCATION_Some_college	0.3277	nan	nan	nan	nan
MARITAL_STATUS_Married_AF_spouse	2.5332	0.5531	4.5797	0.0000	1.4491
MARITAL_STATUS_Married_civ_spouse	2.0745	0.2578	8.0478	0.0000	1.5692
MARITAL_STATUS_spouse_absent	-0.0002	0.2193	-0.0008	0.9993	-0.4300
MARITAL_STATUS_Never_married	-0.4448	0.0831	-5.3558	0.0000	-0.6076
MARITAL_STATUS_Separated	-0.1267	0.1565	-0.8095	0.4182	-0.4335
MARITAL_ Widowed	0.1470	0.1447	1.0163	0.3095	-0.1365
OCCUPATION_Adm_clerical	-0.6934	nan	nan	nan	nan
OCCUPATION_Armed-Forces	-1.8690	nan	nan	nan	nan
OCCUPATION_Craft_repair	-0.6091	nan	nan	nan	nan
OCCUPATION_ Exec_managerial	0.1050	nan	nan	nan	nan
OCCUPATION_Farming_fishing	-1.6281	nan	nan	nan	nan
OCCUPATION_Handlers_cleaners	-1.3919	nan	nan	nan	nan
OCCUPATION_Machine_op_inspct	-1.0167	nan	nan	nan	nan
OCCUPATION_Other_service	-1.5347	nan	nan	nan	nan
OCCUPATION_Priv_house_serv	-3.2375	nan	nan	nan	nan
OCCUPATION_Prof_specialty	-0.1701	nan	nan	nan	nan
OCCUPATION_Protective_serv	-0.1048	nan	nan	nan	nan
OCCUPATION_Sales	-0.4062	nan	nan	nan	nan
OCCUPATION_Tech_support	-0.0548	nan	nan	nan	nan
OCCUPATION_Transport_moving	-0.7956	nan	nan	nan	nan
RELATIONSHIP_Not_in_family	0.4943	0.2554	1.9356	0.0529	-0.0062
RELATIONSHIP_Other_relative	-0.3647	0.2375	-1.5359	0.1246	-0.8302
RELATIONSHIP_Own_child	-0.6961	0.2541	-2.7393	0.0062	-1.1942
RELATIONSHIP_Unmarried	0.3176	0.2703	1.1749	0.2400	-0.2122
RELATIONSHIP_Wife	1.3374	0.0989	13.5212	0.0000	1.1435
RACE_Asian_Pac_Islander	0.5697	0.2611	2.1814	0.0292	0.0578
RACE_Black	0.3241	0.2254	1.4383	0.1503	-0.1176
RACE_Other	0.0486	0.3445	0.1412	0.8877	-0.6267
RACE_White	0.4632	0.2143	2.1619	0.0306	0.0433
SEX_Male	0.8321	0.0748	11.1175	0.0000	0.6854
NATIVE_COUNTRY_Cambodia	1.4211	0.6287	2.2602	0.0238	0.1888
NATIVE_COUNTRY_Canada	0.5238	0.2905	1.8033	0.0713	-0.0455
NATIVE_COUNTRY_China	-0.4979	0.3926	-1.2683	0.2047	-1.2674
NATIVE_COUNTRY_Columbia	-2.0415	0.8183	-2.4949	0.0126	-3.6452

```
NATIVE_COUNTRY_Cuba                          0.4994        0.3309  1.5093 0.1312          -0.1491
NATIVE_COUNTRY_Dominican_Republic           -0.9125        0.7727 -1.1810 0.2376          -2.4269
NATIVE_COUNTRY_Ecuador                      -0.0833        0.7078 -0.1177 0.9063          -1.4704
NATIVE_COUNTRY_El-Salvador                  -0.4327        0.4813 -0.8992 0.3686          -1.3760
NATIVE_COUNTRY_England                       0.5347        0.3261  1.6398 0.1010          -0.1044
NATIVE_COUNTRY_France                        0.7233        0.5259  1.3753 0.1690          -0.3075
NATIVE_COUNTRY_Germany                       0.6293        0.2772  2.2699 0.0232           0.0859
NATIVE_COUNTRY_Greece                       -0.7584        0.5528 -1.3720 0.1701          -1.8420
NATIVE_COUNTRY_Guatemala                    -0.0992        0.7505 -0.1322 0.8948          -1.5701
NATIVE_COUNTRY_Haiti                         0.0597        0.6857  0.0870 0.9307          -1.2843
NATIVE_COUNTRY_Holand-Netherlands          -31.3715 62950672.8850 -0.0000 1.0000 -123381083.0287
NATIVE_COUNTRY_Honduras                     -1.0680        2.2872 -0.4670 0.6405          -5.5510
NATIVE_COUNTRY_Hong                          0.0698        0.6796  0.1027 0.9182          -1.2622
NATIVE_COUNTRY_Hungary                      -0.1848        0.7826 -0.2361 0.8133          -1.7186
NATIVE_COUNTRY_India                        -0.2394        0.3245 -0.7375 0.4608          -0.8755
NATIVE_COUNTRY_Iran                          0.2669        0.4341  0.6148 0.5387          -0.5840
NATIVE_COUNTRY_Ireland                       0.6031        0.6442  0.9362 0.3492          -0.6595
NATIVE_COUNTRY_Italy                         0.9679        0.3414  2.8348 0.0046           0.2987
NATIVE_COUNTRY_Jamaica                       0.1708        0.4551  0.3752 0.7075          -0.7213
NATIVE_COUNTRY_Japan                         0.5205        0.4101  1.2691 0.2044          -0.2833
NATIVE_COUNTRY_Laos                         -0.5059        0.8610 -0.5876 0.5568          -2.1934
NATIVE_COUNTRY_Mexico                       -0.3619        0.2516 -1.4384 0.1503          -0.8550
NATIVE_COUNTRY_Nicaragua                    -0.5698        0.7995 -0.7127 0.4760          -2.1368
NATIVE_COUNTRY_Outlying-US_Guam_USVI_etc   -16.0934     2533.2211 -0.0064 0.9949       -4981.1156
NATIVE_COUNTRY_Peru                         -0.6328        0.8530 -0.7418 0.4582          -2.3047
NATIVE_COUNTRY_Philippines                   0.5955        0.2762  2.1558 0.0311           0.0541
NATIVE_COUNTRY_Poland                        0.1539        0.4155  0.3704 0.7111          -0.6604
NATIVE_COUNTRY_Portugal                      0.1048        0.6350  0.1651 0.8689          -1.1398
NATIVE_COUNTRY_Puerto_Rico                  -0.1502        0.3978 -0.3776 0.7057          -0.9300
NATIVE_COUNTRY_Scotland                      0.1531        0.7919  0.1934 0.8467          -1.3989
NATIVE_COUNTRY_South                        -0.7195        0.4205 -1.7113 0.0870          -1.5436
NATIVE_COUNTRY_Taiwan                        0.1867        0.4660  0.4006 0.6887          -0.7267
NATIVE_COUNTRY_Thailand                     -0.4214        0.8331 -0.5058 0.6130          -2.0543
NATIVE_COUNTRY_Trinadad_Tobago              -0.2135        0.8660 -0.2466 0.8052          -1.9109
NATIVE_COUNTRY_United_States                 0.3902        0.1347  2.8981 0.0038           0.1263
NATIVE_COUNTRY_Vietnam                      -1.0796        0.6265 -1.7231 0.0849          -2.3075
NATIVE_COUNTRY_Yugoslavia                    0.8643        0.6832  1.2651 0.2058          -0.4747
AGE                                          1.8822        0.1182 15.9249 0.0000           1.6505
FNLWGT                                       0.6131        0.1345  4.5580 0.0000           0.3494
EDUCATION_NUM                                3.1002           nan     nan    nan              nan
CAPITAL_GAIN                                 2.3683        0.0748 31.6659 0.0000           2.2217
CAPITAL_LOSS                                 1.2707        0.0788 16.1220 0.0000           1.1162
HOURS_PER_WEEK                               0.0298        0.0016 18.7619 0.0000           0.0267
=================================================================================================


E:\anaconda\lib\site-packages\statsmodels\base\model.py:1029: RuntimeWarning: invalid value enc
```

```
  return np.sqrt(np.diag(self.cov_params()))
E:\anaconda\lib\site-packages\scipy\stats\_distn_infrastructure.py:879: RuntimeWarning: invali
  return (self.a < x) & (x < self.b)
E:\anaconda\lib\site-packages\scipy\stats\_distn_infrastructure.py:879: RuntimeWarning: invali
  return (self.a < x) & (x < self.b)
E:\anaconda\lib\site-packages\scipy\stats\_distn_infrastructure.py:1821: RuntimeWarning: inval
  cond2 = cond0 & (x <= self.a)
```

```
In [35]: accuracy_score(y,np.where(result_.predict(X)>0.5,1,0))
```

```
Out[35]: 0.84730198703971
```

```
In [36]: print(Y)
```

```
       TARGET
0         0.0
1         0.0
2         0.0
3         0.0
4         0.0
5         0.0
6         0.0
7         1.0
8         1.0
9         1.0
10        1.0
11        1.0
12        0.0
13        0.0
14        1.0
15        0.0
16        0.0
17        0.0
18        0.0
19        1.0
20        1.0
21        0.0
22        0.0
23        0.0
24        0.0
25        1.0
26        0.0
27        1.0
28        0.0
29        0.0
30        0.0
31        0.0
32        0.0
```

33	0.0
34	0.0
35	0.0
36	0.0
37	0.0
38	1.0
39	0.0
40	0.0
41	0.0
42	0.0
43	0.0
44	0.0
45	1.0
46	0.0
47	0.0
48	0.0
49	0.0
50	0.0
51	0.0
52	1.0
53	1.0
54	0.0
55	1.0
56	0.0
57	0.0
58	0.0
59	0.0
60	0.0
61	0.0
62	0.0
63	1.0
64	0.0
65	0.0
66	0.0
67	1.0
68	1.0
69	0.0
70	0.0
71	0.0
72	1.0
73	0.0
74	0.0
75	0.0
76	0.0
77	0.0
78	0.0
79	0.0
80	0.0

81	0.0
82	0.0
83	0.0
84	1.0
85	0.0
86	1.0
87	0.0
88	0.0
89	1.0
90	0.0
91	0.0
92	0.0
93	0.0
94	1.0
95	0.0
96	1.0
97	1.0
98	0.0
99	0.0
100	1.0
101	1.0
102	0.0
103	0.0
104	0.0
105	1.0
106	0.0
107	0.0
108	0.0
109	0.0
110	0.0
111	1.0
112	1.0
113	0.0
114	0.0
115	0.0
116	0.0
117	1.0
118	0.0
119	0.0
120	0.0
121	0.0
122	0.0
123	1.0
124	0.0
125	1.0
126	0.0
127	1.0
128	0.0

129	0.0
130	0.0
131	0.0
132	0.0
133	0.0
134	0.0
135	1.0
136	0.0
137	0.0
138	0.0
139	1.0
140	0.0
141	0.0
142	0.0
143	1.0
144	0.0
145	0.0
146	0.0
147	0.0
148	0.0
149	0.0
150	0.0
151	0.0
152	0.0
153	0.0
154	1.0
155	0.0
156	0.0
157	0.0
158	0.0
159	0.0
160	0.0
161	0.0
162	0.0
163	0.0
164	1.0
165	0.0
166	0.0
167	0.0
168	0.0
169	0.0
170	0.0
171	0.0
172	1.0
173	0.0
174	1.0
175	0.0
176	0.0

177	0.0
178	0.0
179	0.0
180	1.0
181	0.0
182	0.0
183	1.0
184	1.0
185	0.0
186	0.0
187	0.0
188	0.0
189	1.0
190	0.0
191	0.0
192	0.0
193	0.0
194	0.0
195	0.0
196	0.0
197	1.0
198	1.0
199	0.0
200	0.0
201	0.0
202	1.0
203	0.0
204	0.0
205	0.0
206	1.0
207	0.0
208	1.0
209	0.0
210	0.0
211	1.0
212	0.0
213	0.0
214	1.0
215	1.0
216	0.0
217	0.0
218	0.0
219	0.0
220	0.0
221	0.0
222	0.0
223	0.0
224	0.0

225	0.0
226	0.0
227	0.0
228	0.0
229	0.0
230	0.0
231	0.0
232	0.0
233	0.0
234	1.0
235	0.0
236	0.0
237	1.0
238	1.0
239	0.0
240	1.0
241	0.0
242	0.0
243	0.0
244	0.0
245	0.0
246	1.0
247	0.0
248	1.0
249	0.0
250	1.0
251	0.0
252	0.0
253	0.0
254	0.0
255	1.0
256	0.0
257	0.0
258	0.0
259	0.0
260	0.0
261	0.0
262	0.0
263	0.0
264	0.0
265	1.0
266	0.0
267	1.0
268	0.0
269	1.0
270	1.0
271	0.0
272	0.0

273	0.0
274	0.0
275	0.0
276	1.0
277	0.0
278	0.0
279	1.0
280	0.0
281	1.0
282	0.0
283	0.0
284	0.0
285	1.0
286	1.0
287	0.0
288	0.0
289	0.0
290	0.0
291	0.0
292	0.0
293	0.0
294	0.0
295	0.0
296	0.0
297	0.0
298	0.0
299	0.0
300	1.0
301	0.0
302	1.0
303	1.0
304	0.0
305	0.0
306	0.0
307	0.0
308	1.0
309	1.0
310	0.0
311	1.0
312	0.0
313	1.0
314	0.0
315	0.0
316	0.0
317	0.0
318	0.0
319	0.0
320	0.0

321	0.0
322	0.0
323	0.0
324	0.0
325	0.0
326	0.0
327	1.0
328	0.0
329	0.0
330	0.0
331	0.0
332	0.0
333	0.0
334	0.0
335	0.0
336	0.0
337	0.0
338	0.0
339	0.0
340	0.0
341	1.0
342	1.0
343	0.0
344	0.0
345	0.0
346	0.0
347	0.0
348	0.0
349	0.0
350	1.0
351	0.0
352	1.0
353	0.0
354	1.0
355	1.0
356	0.0
357	1.0
358	0.0
359	0.0
360	0.0
361	1.0
362	0.0
363	1.0
364	0.0
365	0.0
366	0.0
367	0.0
368	1.0

369	0.0
370	0.0
371	0.0
372	1.0
373	0.0
374	0.0
375	0.0
376	0.0
377	0.0
378	0.0
379	0.0
380	0.0
381	0.0
382	0.0
383	0.0
384	0.0
385	0.0
386	0.0
387	1.0
388	0.0
389	0.0
390	1.0
391	0.0
392	0.0
393	1.0
394	0.0
395	0.0
396	0.0
397	0.0
398	1.0
399	1.0
400	0.0
401	0.0
402	1.0
403	0.0
404	0.0
405	1.0
406	0.0
407	0.0
408	1.0
409	0.0
410	0.0
411	0.0
412	0.0
413	1.0
414	0.0
415	1.0
416	0.0

417	0.0
418	0.0
419	0.0
420	0.0
421	0.0
422	1.0
423	0.0
424	0.0
425	0.0
426	0.0
427	0.0
428	1.0
429	0.0
430	0.0
431	0.0
432	0.0
433	1.0
434	0.0
435	1.0
436	0.0
437	0.0
438	0.0
439	0.0
440	0.0
441	0.0
442	0.0
443	1.0
444	0.0
445	0.0
446	0.0
447	0.0
448	0.0
449	0.0
450	0.0
451	1.0
452	1.0
453	1.0
454	1.0
455	1.0
456	0.0
457	0.0
458	0.0
459	0.0
460	0.0
461	0.0
462	0.0
463	0.0
464	0.0

465	0.0
466	0.0
467	0.0
468	1.0
469	1.0
470	1.0
471	0.0
472	0.0
473	0.0
474	0.0
475	0.0
476	0.0
477	0.0
478	0.0
479	0.0
480	0.0
481	0.0
482	0.0
483	0.0
484	0.0
485	0.0
486	0.0
487	0.0
488	0.0
489	1.0
490	0.0
491	0.0
492	0.0
493	0.0
494	0.0
495	0.0
496	0.0
497	0.0
498	0.0
499	0.0
...	...
32061	0.0
32062	0.0
32063	1.0
32064	0.0
32065	0.0
32066	0.0
32067	1.0
32068	0.0
32069	0.0
32070	0.0
32071	0.0
32072	0.0

32073	0.0
32074	0.0
32075	0.0
32076	1.0
32077	1.0
32078	0.0
32079	1.0
32080	0.0
32081	1.0
32082	0.0
32083	0.0
32084	1.0
32085	0.0
32086	0.0
32087	0.0
32088	0.0
32089	0.0
32090	1.0
32091	0.0
32092	1.0
32093	0.0
32094	0.0
32095	0.0
32096	0.0
32097	0.0
32098	1.0
32099	0.0
32100	0.0
32101	0.0
32102	1.0
32103	0.0
32104	0.0
32105	0.0
32106	0.0
32107	0.0
32108	0.0
32109	1.0
32110	0.0
32111	0.0
32112	0.0
32113	0.0
32114	1.0
32115	1.0
32116	0.0
32117	1.0
32118	0.0
32119	1.0
32120	0.0

32121	0.0
32122	0.0
32123	1.0
32124	0.0
32125	1.0
32126	0.0
32127	0.0
32128	0.0
32129	1.0
32130	1.0
32131	1.0
32132	0.0
32133	1.0
32134	0.0
32135	1.0
32136	0.0
32137	0.0
32138	1.0
32139	1.0
32140	0.0
32141	0.0
32142	0.0
32143	0.0
32144	0.0
32145	0.0
32146	0.0
32147	0.0
32148	0.0
32149	0.0
32150	0.0
32151	0.0
32152	0.0
32153	0.0
32154	0.0
32155	0.0
32156	1.0
32157	0.0
32158	0.0
32159	0.0
32160	0.0
32161	1.0
32162	0.0
32163	0.0
32164	1.0
32165	0.0
32166	0.0
32167	0.0
32168	0.0

32169	0.0
32170	0.0
32171	0.0
32172	0.0
32173	1.0
32174	0.0
32175	0.0
32176	0.0
32177	0.0
32178	0.0
32179	1.0
32180	0.0
32181	1.0
32182	0.0
32183	1.0
32184	0.0
32185	1.0
32186	0.0
32187	0.0
32188	0.0
32189	0.0
32190	0.0
32191	0.0
32192	0.0
32193	0.0
32194	0.0
32195	1.0
32196	0.0
32197	0.0
32198	1.0
32199	1.0
32200	0.0
32201	0.0
32202	0.0
32203	1.0
32204	0.0
32205	0.0
32206	0.0
32207	0.0
32208	0.0
32209	0.0
32210	0.0
32211	0.0
32212	0.0
32213	0.0
32214	1.0
32215	0.0
32216	1.0

32217	0.0
32218	0.0
32219	0.0
32220	0.0
32221	1.0
32222	1.0
32223	0.0
32224	0.0
32225	0.0
32226	0.0
32227	0.0
32228	1.0
32229	0.0
32230	1.0
32231	0.0
32232	0.0
32233	0.0
32234	1.0
32235	0.0
32236	1.0
32237	0.0
32238	1.0
32239	0.0
32240	0.0
32241	0.0
32242	0.0
32243	0.0
32244	0.0
32245	1.0
32246	1.0
32247	1.0
32248	0.0
32249	1.0
32250	1.0
32251	0.0
32252	0.0
32253	0.0
32254	1.0
32255	0.0
32256	1.0
32257	0.0
32258	1.0
32259	0.0
32260	1.0
32261	0.0
32262	0.0
32263	0.0
32264	0.0

32265	1.0
32266	1.0
32267	1.0
32268	0.0
32269	0.0
32270	0.0
32271	1.0
32272	1.0
32273	0.0
32274	0.0
32275	0.0
32276	0.0
32277	0.0
32278	0.0
32279	0.0
32280	0.0
32281	0.0
32282	0.0
32283	0.0
32284	0.0
32285	0.0
32286	0.0
32287	1.0
32288	0.0
32289	0.0
32290	0.0
32291	0.0
32292	1.0
32293	0.0
32294	0.0
32295	1.0
32296	0.0
32297	1.0
32298	0.0
32299	0.0
32300	0.0
32301	0.0
32302	1.0
32303	0.0
32304	0.0
32305	1.0
32306	0.0
32307	1.0
32308	1.0
32309	0.0
32310	0.0
32311	1.0
32312	0.0

32313	0.0
32314	0.0
32315	0.0
32316	0.0
32317	0.0
32318	1.0
32319	0.0
32320	0.0
32321	0.0
32322	0.0
32323	0.0
32324	0.0
32325	0.0
32326	1.0
32327	0.0
32328	0.0
32329	0.0
32330	0.0
32331	0.0
32332	0.0
32333	1.0
32334	0.0
32335	0.0
32336	0.0
32337	0.0
32338	0.0
32339	0.0
32340	0.0
32341	1.0
32342	0.0
32343	0.0
32344	0.0
32345	0.0
32346	0.0
32347	1.0
32348	0.0
32349	0.0
32350	0.0
32351	0.0
32352	0.0
32353	0.0
32354	0.0
32355	0.0
32356	0.0
32357	0.0
32358	1.0
32359	0.0
32360	0.0

32361	0.0
32362	0.0
32363	0.0
32364	0.0
32365	1.0
32366	0.0
32367	0.0
32368	0.0
32369	0.0
32370	1.0
32371	1.0
32372	1.0
32373	0.0
32374	0.0
32375	1.0
32376	0.0
32377	0.0
32378	0.0
32379	0.0
32380	0.0
32381	0.0
32382	0.0
32383	0.0
32384	0.0
32385	0.0
32386	1.0
32387	0.0
32388	0.0
32389	1.0
32390	0.0
32391	0.0
32392	0.0
32393	0.0
32394	0.0
32395	0.0
32396	0.0
32397	0.0
32398	0.0
32399	1.0
32400	1.0
32401	0.0
32402	1.0
32403	0.0
32404	0.0
32405	0.0
32406	0.0
32407	0.0
32408	0.0

32409	0.0
32410	0.0
32411	0.0
32412	0.0
32413	0.0
32414	0.0
32415	1.0
32416	1.0
32417	0.0
32418	0.0
32419	1.0
32420	0.0
32421	0.0
32422	0.0
32423	0.0
32424	0.0
32425	0.0
32426	0.0
32427	0.0
32428	0.0
32429	1.0
32430	0.0
32431	0.0
32432	0.0
32433	0.0
32434	0.0
32435	0.0
32436	1.0
32437	0.0
32438	1.0
32439	0.0
32440	1.0
32441	1.0
32442	0.0
32443	0.0
32444	0.0
32445	0.0
32446	0.0
32447	0.0
32448	0.0
32449	1.0
32450	0.0
32451	0.0
32452	0.0
32453	1.0
32454	0.0
32455	1.0
32456	0.0

32457	1.0
32458	1.0
32459	0.0
32460	0.0
32461	0.0
32462	1.0
32463	0.0
32464	1.0
32465	0.0
32466	1.0
32467	0.0
32468	0.0
32469	0.0
32470	0.0
32471	0.0
32472	0.0
32473	0.0
32474	1.0
32475	0.0
32476	0.0
32477	0.0
32478	0.0
32479	0.0
32480	1.0
32481	0.0
32482	0.0
32483	0.0
32484	0.0
32485	0.0
32486	0.0
32487	0.0
32488	0.0
32489	0.0
32490	0.0
32491	0.0
32492	0.0
32493	0.0
32494	0.0
32495	0.0
32496	0.0
32497	0.0
32498	0.0
32499	0.0
32500	0.0
32501	0.0
32502	0.0
32503	0.0
32504	1.0

32505	0.0
32506	1.0
32507	0.0
32508	0.0
32509	0.0
32510	1.0
32511	0.0
32512	0.0
32513	1.0
32514	0.0
32515	0.0
32516	0.0
32517	0.0
32518	1.0
32519	1.0
32520	0.0
32521	0.0
32522	0.0
32523	0.0
32524	0.0
32525	0.0
32526	0.0
32527	0.0
32528	0.0
32529	0.0
32530	1.0
32531	0.0
32532	1.0
32533	1.0
32534	0.0
32535	0.0
32536	1.0
32537	0.0
32538	1.0
32539	1.0
32540	0.0
32541	0.0
32542	0.0
32543	0.0
32544	0.0
32545	1.0
32546	0.0
32547	0.0
32548	0.0
32549	0.0
32550	0.0
32551	0.0
32552	0.0

```
32553     0.0
32554     1.0
32555     0.0
32556     0.0
32557     1.0
32558     0.0
32559     0.0
32560     1.0

[32561 rows x 1 columns]
```

```
In [37]: pd.merge(X,Y, left_index=True, right_index=True)[['TARGET']]
```

```
Out[37]:        TARGET
         0         0.0
         1         0.0
         2         0.0
         3         0.0
         4         0.0
         5         0.0
         6         0.0
         7         1.0
         8         1.0
         9         1.0
         10        1.0
         11        1.0
         12        0.0
         13        0.0
         14        1.0
         15        0.0
         16        0.0
         17        0.0
         18        0.0
         19        1.0
         20        1.0
         21        0.0
         22        0.0
         23        0.0
         24        0.0
         25        1.0
         26        0.0
         27        1.0
         28        0.0
         29        0.0
         30        0.0
         31        0.0
         32        0.0
```

33	0.0
34	0.0
35	0.0
36	0.0
37	0.0
38	1.0
39	0.0
40	0.0
41	0.0
42	0.0
43	0.0
44	0.0
45	1.0
46	0.0
47	0.0
48	0.0
49	0.0
50	0.0
51	0.0
52	1.0
53	1.0
54	0.0
55	1.0
56	0.0
57	0.0
58	0.0
59	0.0
60	0.0
61	0.0
62	0.0
63	1.0
64	0.0
65	0.0
66	0.0
67	1.0
68	1.0
69	0.0
70	0.0
71	0.0
72	1.0
73	0.0
74	0.0
75	0.0
76	0.0
77	0.0
78	0.0
79	0.0
80	0.0

81	0.0
82	0.0
83	0.0
84	1.0
85	0.0
86	1.0
87	0.0
88	0.0
89	1.0
90	0.0
91	0.0
92	0.0
93	0.0
94	1.0
95	0.0
96	1.0
97	1.0
98	0.0
99	0.0
100	1.0
101	1.0
102	0.0
103	0.0
104	0.0
105	1.0
106	0.0
107	0.0
108	0.0
109	0.0
110	0.0
111	1.0
112	1.0
113	0.0
114	0.0
115	0.0
116	0.0
117	1.0
118	0.0
119	0.0
120	0.0
121	0.0
122	0.0
123	1.0
124	0.0
125	1.0
126	0.0
127	1.0
128	0.0

129	0.0
130	0.0
131	0.0
132	0.0
133	0.0
134	0.0
135	1.0
136	0.0
137	0.0
138	0.0
139	1.0
140	0.0
141	0.0
142	0.0
143	1.0
144	0.0
145	0.0
146	0.0
147	0.0
148	0.0
149	0.0
150	0.0
151	0.0
152	0.0
153	0.0
154	1.0
155	0.0
156	0.0
157	0.0
158	0.0
159	0.0
160	0.0
161	0.0
162	0.0
163	0.0
164	1.0
165	0.0
166	0.0
167	0.0
168	0.0
169	0.0
170	0.0
171	0.0
172	1.0
173	0.0
174	1.0
175	0.0
176	0.0

177	0.0
178	0.0
179	0.0
180	1.0
181	0.0
182	0.0
183	1.0
184	1.0
185	0.0
186	0.0
187	0.0
188	0.0
189	1.0
190	0.0
191	0.0
192	0.0
193	0.0
194	0.0
195	0.0
196	0.0
197	1.0
198	1.0
199	0.0
200	0.0
201	0.0
202	1.0
203	0.0
204	0.0
205	0.0
206	1.0
207	0.0
208	1.0
209	0.0
210	0.0
211	1.0
212	0.0
213	0.0
214	1.0
215	1.0
216	0.0
217	0.0
218	0.0
219	0.0
220	0.0
221	0.0
222	0.0
223	0.0
224	0.0

225	0.0
226	0.0
227	0.0
228	0.0
229	0.0
230	0.0
231	0.0
232	0.0
233	0.0
234	1.0
235	0.0
236	0.0
237	1.0
238	1.0
239	0.0
240	1.0
241	0.0
242	0.0
243	0.0
244	0.0
245	0.0
246	1.0
247	0.0
248	1.0
249	0.0
250	1.0
251	0.0
252	0.0
253	0.0
254	0.0
255	1.0
256	0.0
257	0.0
258	0.0
259	0.0
260	0.0
261	0.0
262	0.0
263	0.0
264	0.0
265	1.0
266	0.0
267	1.0
268	0.0
269	1.0
270	1.0
271	0.0
272	0.0

273	0.0
274	0.0
275	0.0
276	1.0
277	0.0
278	0.0
279	1.0
280	0.0
281	1.0
282	0.0
283	0.0
284	0.0
285	1.0
286	1.0
287	0.0
288	0.0
289	0.0
290	0.0
291	0.0
292	0.0
293	0.0
294	0.0
295	0.0
296	0.0
297	0.0
298	0.0
299	0.0
300	1.0
301	0.0
302	1.0
303	1.0
304	0.0
305	0.0
306	0.0
307	0.0
308	1.0
309	1.0
310	0.0
311	1.0
312	0.0
313	1.0
314	0.0
315	0.0
316	0.0
317	0.0
318	0.0
319	0.0
320	0.0

321	0.0
322	0.0
323	0.0
324	0.0
325	0.0
326	0.0
327	1.0
328	0.0
329	0.0
330	0.0
331	0.0
332	0.0
333	0.0
334	0.0
335	0.0
336	0.0
337	0.0
338	0.0
339	0.0
340	0.0
341	1.0
342	1.0
343	0.0
344	0.0
345	0.0
346	0.0
347	0.0
348	0.0
349	0.0
350	1.0
351	0.0
352	1.0
353	0.0
354	1.0
355	1.0
356	0.0
357	1.0
358	0.0
359	0.0
360	0.0
361	1.0
362	0.0
363	1.0
364	0.0
365	0.0
366	0.0
367	0.0
368	1.0

369	0.0
370	0.0
371	0.0
372	1.0
373	0.0
374	0.0
375	0.0
376	0.0
377	0.0
378	0.0
379	0.0
380	0.0
381	0.0
382	0.0
383	0.0
384	0.0
385	0.0
386	0.0
387	1.0
388	0.0
389	0.0
390	1.0
391	0.0
392	0.0
393	1.0
394	0.0
395	0.0
396	0.0
397	0.0
398	1.0
399	1.0
400	0.0
401	0.0
402	1.0
403	0.0
404	0.0
405	1.0
406	0.0
407	0.0
408	1.0
409	0.0
410	0.0
411	0.0
412	0.0
413	1.0
414	0.0
415	1.0
416	0.0

417	0.0
418	0.0
419	0.0
420	0.0
421	0.0
422	1.0
423	0.0
424	0.0
425	0.0
426	0.0
427	0.0
428	1.0
429	0.0
430	0.0
431	0.0
432	0.0
433	1.0
434	0.0
435	1.0
436	0.0
437	0.0
438	0.0
439	0.0
440	0.0
441	0.0
442	0.0
443	1.0
444	0.0
445	0.0
446	0.0
447	0.0
448	0.0
449	0.0
450	0.0
451	1.0
452	1.0
453	1.0
454	1.0
455	1.0
456	0.0
457	0.0
458	0.0
459	0.0
460	0.0
461	0.0
462	0.0
463	0.0
464	0.0

465	0.0
466	0.0
467	0.0
468	1.0
469	1.0
470	1.0
471	0.0
472	0.0
473	0.0
474	0.0
475	0.0
476	0.0
477	0.0
478	0.0
479	0.0
480	0.0
481	0.0
482	0.0
483	0.0
484	0.0
485	0.0
486	0.0
487	0.0
488	0.0
489	1.0
490	0.0
491	0.0
492	0.0
493	0.0
494	0.0
495	0.0
496	0.0
497	0.0
498	0.0
499	0.0
...	...
32061	0.0
32062	0.0
32063	1.0
32064	0.0
32065	0.0
32066	0.0
32067	1.0
32068	0.0
32069	0.0
32070	0.0
32071	0.0
32072	0.0

32073	0.0
32074	0.0
32075	0.0
32076	1.0
32077	1.0
32078	0.0
32079	1.0
32080	0.0
32081	1.0
32082	0.0
32083	0.0
32084	1.0
32085	0.0
32086	0.0
32087	0.0
32088	0.0
32089	0.0
32090	1.0
32091	0.0
32092	1.0
32093	0.0
32094	0.0
32095	0.0
32096	0.0
32097	0.0
32098	1.0
32099	0.0
32100	0.0
32101	0.0
32102	1.0
32103	0.0
32104	0.0
32105	0.0
32106	0.0
32107	0.0
32108	0.0
32109	1.0
32110	0.0
32111	0.0
32112	0.0
32113	0.0
32114	1.0
32115	1.0
32116	0.0
32117	1.0
32118	0.0
32119	1.0
32120	0.0

32121	0.0
32122	0.0
32123	1.0
32124	0.0
32125	1.0
32126	0.0
32127	0.0
32128	0.0
32129	1.0
32130	1.0
32131	1.0
32132	0.0
32133	1.0
32134	0.0
32135	1.0
32136	0.0
32137	0.0
32138	1.0
32139	1.0
32140	0.0
32141	0.0
32142	0.0
32143	0.0
32144	0.0
32145	0.0
32146	0.0
32147	0.0
32148	0.0
32149	0.0
32150	0.0
32151	0.0
32152	0.0
32153	0.0
32154	0.0
32155	0.0
32156	1.0
32157	0.0
32158	0.0
32159	0.0
32160	0.0
32161	1.0
32162	0.0
32163	0.0
32164	1.0
32165	0.0
32166	0.0
32167	0.0
32168	0.0

32169	0.0
32170	0.0
32171	0.0
32172	0.0
32173	1.0
32174	0.0
32175	0.0
32176	0.0
32177	0.0
32178	0.0
32179	1.0
32180	0.0
32181	1.0
32182	0.0
32183	1.0
32184	0.0
32185	1.0
32186	0.0
32187	0.0
32188	0.0
32189	0.0
32190	0.0
32191	0.0
32192	0.0
32193	0.0
32194	0.0
32195	1.0
32196	0.0
32197	0.0
32198	1.0
32199	1.0
32200	0.0
32201	0.0
32202	0.0
32203	1.0
32204	0.0
32205	0.0
32206	0.0
32207	0.0
32208	0.0
32209	0.0
32210	0.0
32211	0.0
32212	0.0
32213	0.0
32214	1.0
32215	0.0
32216	1.0

32217	0.0
32218	0.0
32219	0.0
32220	0.0
32221	1.0
32222	1.0
32223	0.0
32224	0.0
32225	0.0
32226	0.0
32227	0.0
32228	1.0
32229	0.0
32230	1.0
32231	0.0
32232	0.0
32233	0.0
32234	1.0
32235	0.0
32236	1.0
32237	0.0
32238	1.0
32239	0.0
32240	0.0
32241	0.0
32242	0.0
32243	0.0
32244	0.0
32245	1.0
32246	1.0
32247	1.0
32248	0.0
32249	1.0
32250	1.0
32251	0.0
32252	0.0
32253	0.0
32254	1.0
32255	0.0
32256	1.0
32257	0.0
32258	1.0
32259	0.0
32260	1.0
32261	0.0
32262	0.0
32263	0.0
32264	0.0

32265	1.0
32266	1.0
32267	1.0
32268	0.0
32269	0.0
32270	0.0
32271	1.0
32272	1.0
32273	0.0
32274	0.0
32275	0.0
32276	0.0
32277	0.0
32278	0.0
32279	0.0
32280	0.0
32281	0.0
32282	0.0
32283	0.0
32284	0.0
32285	0.0
32286	0.0
32287	1.0
32288	0.0
32289	0.0
32290	0.0
32291	0.0
32292	1.0
32293	0.0
32294	0.0
32295	1.0
32296	0.0
32297	1.0
32298	0.0
32299	0.0
32300	0.0
32301	0.0
32302	1.0
32303	0.0
32304	0.0
32305	1.0
32306	0.0
32307	1.0
32308	1.0
32309	0.0
32310	0.0
32311	1.0
32312	0.0

32313	0.0
32314	0.0
32315	0.0
32316	0.0
32317	0.0
32318	1.0
32319	0.0
32320	0.0
32321	0.0
32322	0.0
32323	0.0
32324	0.0
32325	0.0
32326	1.0
32327	0.0
32328	0.0
32329	0.0
32330	0.0
32331	0.0
32332	0.0
32333	1.0
32334	0.0
32335	0.0
32336	0.0
32337	0.0
32338	0.0
32339	0.0
32340	0.0
32341	1.0
32342	0.0
32343	0.0
32344	0.0
32345	0.0
32346	0.0
32347	1.0
32348	0.0
32349	0.0
32350	0.0
32351	0.0
32352	0.0
32353	0.0
32354	0.0
32355	0.0
32356	0.0
32357	0.0
32358	1.0
32359	0.0
32360	0.0

32361	0.0
32362	0.0
32363	0.0
32364	0.0
32365	1.0
32366	0.0
32367	0.0
32368	0.0
32369	0.0
32370	1.0
32371	1.0
32372	1.0
32373	0.0
32374	0.0
32375	1.0
32376	0.0
32377	0.0
32378	0.0
32379	0.0
32380	0.0
32381	0.0
32382	0.0
32383	0.0
32384	0.0
32385	0.0
32386	1.0
32387	0.0
32388	0.0
32389	1.0
32390	0.0
32391	0.0
32392	0.0
32393	0.0
32394	0.0
32395	0.0
32396	0.0
32397	0.0
32398	0.0
32399	1.0
32400	1.0
32401	0.0
32402	1.0
32403	0.0
32404	0.0
32405	0.0
32406	0.0
32407	0.0
32408	0.0

32409	0.0
32410	0.0
32411	0.0
32412	0.0
32413	0.0
32414	0.0
32415	1.0
32416	1.0
32417	0.0
32418	0.0
32419	1.0
32420	0.0
32421	0.0
32422	0.0
32423	0.0
32424	0.0
32425	0.0
32426	0.0
32427	0.0
32428	0.0
32429	1.0
32430	0.0
32431	0.0
32432	0.0
32433	0.0
32434	0.0
32435	0.0
32436	1.0
32437	0.0
32438	1.0
32439	0.0
32440	1.0
32441	1.0
32442	0.0
32443	0.0
32444	0.0
32445	0.0
32446	0.0
32447	0.0
32448	0.0
32449	1.0
32450	0.0
32451	0.0
32452	0.0
32453	1.0
32454	0.0
32455	1.0
32456	0.0

32457	1.0
32458	1.0
32459	0.0
32460	0.0
32461	0.0
32462	1.0
32463	0.0
32464	1.0
32465	0.0
32466	1.0
32467	0.0
32468	0.0
32469	0.0
32470	0.0
32471	0.0
32472	0.0
32473	0.0
32474	1.0
32475	0.0
32476	0.0
32477	0.0
32478	0.0
32479	0.0
32480	1.0
32481	0.0
32482	0.0
32483	0.0
32484	0.0
32485	0.0
32486	0.0
32487	0.0
32488	0.0
32489	0.0
32490	0.0
32491	0.0
32492	0.0
32493	0.0
32494	0.0
32495	0.0
32496	0.0
32497	0.0
32498	0.0
32499	0.0
32500	0.0
32501	0.0
32502	0.0
32503	0.0
32504	1.0

32505	0.0
32506	1.0
32507	0.0
32508	0.0
32509	0.0
32510	1.0
32511	0.0
32512	0.0
32513	1.0
32514	0.0
32515	0.0
32516	0.0
32517	0.0
32518	1.0
32519	1.0
32520	0.0
32521	0.0
32522	0.0
32523	0.0
32524	0.0
32525	0.0
32526	0.0
32527	0.0
32528	0.0
32529	0.0
32530	1.0
32531	0.0
32532	1.0
32533	1.0
32534	0.0
32535	0.0
32536	1.0
32537	0.0
32538	1.0
32539	1.0
32540	0.0
32541	0.0
32542	0.0
32543	0.0
32544	0.0
32545	1.0
32546	0.0
32547	0.0
32548	0.0
32549	0.0
32550	0.0
32551	0.0
32552	0.0

```
32553    0.0
32554    1.0
32555    0.0
32556    0.0
32557    1.0
32558    0.0
32559    0.0
32560    1.0

[32561 rows x 1 columns]
```

In [38]:
```
X_Y_combined = pd.merge(X,Y, left_index=True, right_index=True)
Y_reduced1, X_reduced1 = dmatrices('TARGET ~ MARITAL_STATUS_Married_AF_spouse + MARITA
                  + MARITAL_STATUS_Never_married + RELATIONSHIP_Own_child + RELATIONSH
                  + NATIVE_COUNTRY_Italy + NATIVE_COUNTRY_United_States + AGE + FNLWGT
                  + CAPITAL_LOSS + HOURS_PER_WEEK',
                  X_Y_combined, return_type="dataframe")
X_reduced1.columns
```

Out[38]:
```
Index(['Intercept', 'MARITAL_STATUS_Married_AF_spouse', 'MARITAL_STATUS_Married_civ_sp
```

In [39]:
```
y_reduced1 = np.ravel(Y_reduced1)
```

In [40]:
```
# instantiate a logistic regression model, and fit with X and y
model = LogisticRegression()
model = model.fit(X_reduced1, y_reduced1)

# check the accuracy on the training set
model.score(X_reduced1, y_reduced1)
```

Out[40]:
```
0.7985319861183625
```

In [41]:
```
logit = sm.Logit(y_reduced1, X_reduced1)
logit.fit().params
```

```
Optimization terminated successfully.
         Current function value: 0.388834
         Iterations 8
```

Out[41]:
```
Intercept                            -5.590698
MARITAL_STATUS_Married_AF_spouse      1.815648
MARITAL_STATUS_Married_civ_spouse     1.575228
MARITAL_STATUS_Never_married         -0.208485
RELATIONSHIP_Own_child               -1.486424
RELATIONSHIP_Wife                     1.178586
SEX_Male                              0.704855
NATIVE_COUNTRY_Italy                  0.407299
NATIVE_COUNTRY_United_States          0.376234
```

```
AGE                              1.609788
FNLWGT                           0.514097
CAPITAL_GAIN                     2.492197
CAPITAL_LOSS                     1.480956
HOURS_PER_WEEK                   0.036048
dtype: float64
```

In [42]:
```
result_ = logit.fit()
```

```
Optimization terminated successfully.
         Current function value: 0.388834
         Iterations 8
```

In [43]:
```
accuracy_score(y_reduced1,np.where(result_.predict(X_reduced1)>0.5,1,0))
```

Out[43]:
```
0.7987469672307361
```

In [44]:
```
Y_train, X_train = Y, X
```

In [64]:
```
X_train.head()
```

Out[64]:
```
   Intercept  WORKCLASS_Federal_gov  WORKCLASS_Local_gov  WORKCLASS_Never_worked  WORK
0        1.0                    0.0                  0.0                     0.0
1        1.0                    0.0                  0.0                     0.0
2        1.0                    0.0                  0.0                     0.0
3        1.0                    0.0                  0.0                     0.0
4        1.0                    0.0                  0.0                     0.0

   NATIVE_COUNTRY_Germany  NATIVE_COUNTRY_Greece  NATIVE_COUNTRY_Guatemala  NATIVE_COU
0                     0.0                    0.0                       0.0
1                     0.0                    0.0                       0.0
2                     0.0                    0.0                       0.0
3                     0.0                    0.0                       0.0
4                     0.0                    0.0                       0.0
```

1.20 Interpretation:

The accuracy score of original model is higher than reduced model. So I will stick with original model

In [45]:
```
Y_test, X_test = dmatrices('TARGET ~ AGE + C(WORKCLASS) + FNLWGT +  C(EDUCATION) \
                  + EDUCATION_NUM + C(MARITAL_STATUS) + C(OCCUPATION) + C(RELATIONSHI
                  + C(RACE) + C(SEX) + CAPITAL_GAIN + CAPITAL_LOSS + HOURS_PER_WEEK +
                  test_data, return_type="dataframe")
X_test.columns
```

Out[45]:
```
Index(['Intercept', 'C(WORKCLASS)[T. Federal-gov]', 'C(WORKCLASS)[T. Local-gov]', 'C(
       'C(RELATIONSHIP)[T. Other-relative]', 'C(RELATIONSHIP)[T. Own-child]', 'C(RELA
       'C(NATIVE_COUNTRY)[T. Trinadad&Tobago]', 'C(NATIVE_COUNTRY)[T. United-States]'
      dtype='object')
```

```
In [46]: X_test = X_test.rename(columns = {
         'C(WORKCLASS)[T. Federal-gov]':'WORKCLASS_Federal_gov',
         'C(WORKCLASS)[T. Local-gov]' : 'WORKCLASS_Local_gov',
         'C(WORKCLASS)[T. Never-worked]' : 'WORKCLASS_Never_worked',
         'C(WORKCLASS)[T. Private]' :  'WORKCLASS_Private',
         'C(WORKCLASS)[T. Self-emp-inc]' : 'WORKCLASS_Self_emp_inc',
         'C(WORKCLASS)[T. Self-emp-not-inc]' : 'WORKCLASS_Self_emp_not_inc',
         'C(WORKCLASS)[T. State-gov]' :    'WORKCLASS_State_gov',
         'C(WORKCLASS)[T. Without-pay]' :  'WORKCLASS_Without_pay',
         'C(EDUCATION)[T. 11th]' :  'EDUCATION_11th',
         'C(EDUCATION)[T. 12th]' :   'EDUCATION_12th',
         'C(EDUCATION)[T. 1st-4th]' : 'EDUCATION_1st_4th',
         'C(EDUCATION)[T. 5th-6th]' : 'EDUCATION_5th_6th',
         'C(EDUCATION)[T. 7th-8th]' : 'EDUCATION_7th_8th',
         'C(EDUCATION)[T. 9th]' :     'EDUCATION_9th',
         'C(EDUCATION)[T. Assoc-acdm]' : 'EDUCATION_Assoc_acdm',
         'C(EDUCATION)[T. Assoc-voc]' :   'EDUCATION_voc',
         'C(EDUCATION)[T. Bachelors]' :   'EDUCATION_Bachelors',
         'C(EDUCATION)[T. Doctorate]' :   'EDUCATION_Doctorate',
         'C(EDUCATION)[T. HS-grad]' :     'EDUCATION_HS_grad',
         'C(EDUCATION)[T. Masters]' :     'EDUCATION_Masters',
         'C(EDUCATION)[T. Preschool]' :   'EDUCATION_Preschool',
         'C(EDUCATION)[T. Prof-school]' : 'EDUCATION_Prof_school',
         'C(EDUCATION)[T. Some-college]' : 'EDUCATION_Some_college',
         'C(MARITAL_STATUS)[T. Married-AF-spouse]' : 'MARITAL_STATUS_Married_AF_spouse',
         'C(MARITAL_STATUS)[T. Married-civ-spouse]' : 'MARITAL_STATUS_Married_civ_spouse',
         'C(MARITAL_STATUS)[T. Married-spouse-absent]' : 'MARITAL_STATUS_spouse_absent',
         'C(MARITAL_STATUS)[T. Never-married]' :  'MARITAL_STATUS_Never_married',
         'C(MARITAL_STATUS)[T. Separated]' :   'MARITAL_STATUS_Separated',
         'C(MARITAL_STATUS)[T. Widowed]' :      'MARITAL_ Widowed',
         'C(OCCUPATION)[T. Adm-clerical]' :     'OCCUPATION_Adm_clerical',
         'C(OCCUPATION)[T. Armed-Forces]' :     'OCCUPATION_Armed-Forces',
         'C(OCCUPATION)[T. Craft-repair]' :     'OCCUPATION_Craft_repair',
         'C(OCCUPATION)[T. Exec-managerial]' :  'OCCUPATION_ Exec_managerial',
         'C(OCCUPATION)[T. Farming-fishing]' :  'OCCUPATION_Farming_fishing',
         'C(OCCUPATION)[T. Handlers-cleaners]' : 'OCCUPATION_Handlers_cleaners',
         'C(OCCUPATION)[T. Machine-op-inspct]' : 'OCCUPATION_Machine_op_inspct',
         'C(OCCUPATION)[T. Other-service]' :   'OCCUPATION_Other_service',
         'C(OCCUPATION)[T. Priv-house-serv]' :  'OCCUPATION_Priv_house_serv',
         'C(OCCUPATION)[T. Prof-specialty]' :   'OCCUPATION_Prof_specialty',
         'C(OCCUPATION)[T. Protective-serv]' :   'OCCUPATION_Protective_serv',
         'C(OCCUPATION)[T. Sales]' :           'OCCUPATION_Sales',
         'C(OCCUPATION)[T. Tech-support]' :     'OCCUPATION_Tech_support',
         'C(OCCUPATION)[T. Transport-moving]' :  'OCCUPATION_Transport_moving',
         'C(RELATIONSHIP)[T. Not-in-family]' :  'RELATIONSHIP_Not_in_family',
         'C(RELATIONSHIP)[T. Other-relative]' :  'RELATIONSHIP_Other_relative',
         'C(RELATIONSHIP)[T. Own-child]' :  'RELATIONSHIP_Own_child',
         'C(RELATIONSHIP)[T. Unmarried]' :   'RELATIONSHIP_Unmarried',
```

```
'C(RELATIONSHIP)[T. Wife]' :    'RELATIONSHIP_Wife',
'C(RACE)[T. Asian-Pac-Islander]' : 'RACE_Asian_Pac_Islander',
'C(RACE)[T. Black]' :   'RACE_Black',
'C(RACE)[T. Other]' :   'RACE_Other',
'C(RACE)[T. White]' :   'RACE_White',
'C(SEX)[T. Male]' :     'SEX_Male',
'C(NATIVE_COUNTRY)[T. Cambodia]' :  'NATIVE_COUNTRY_Cambodia',
'C(NATIVE_COUNTRY)[T. Canada]' :   'NATIVE_COUNTRY_Canada',
'C(NATIVE_COUNTRY)[T. China]' :   'NATIVE_COUNTRY_China',
'C(NATIVE_COUNTRY)[T. Columbia]' : 'NATIVE_COUNTRY_Columbia',
'C(NATIVE_COUNTRY)[T. Cuba]' :     'NATIVE_COUNTRY_Cuba',
'C(NATIVE_COUNTRY)[T. Dominican-Republic]' :  'NATIVE_COUNTRY_Dominican_Republic',
'C(NATIVE_COUNTRY)[T. Ecuador]' :   'NATIVE_COUNTRY_Ecuador',
'C(NATIVE_COUNTRY)[T. El-Salvador]' :  'NATIVE_COUNTRY_El-Salvador',
'C(NATIVE_COUNTRY)[T. England]' :  'NATIVE_COUNTRY_England',
'C(NATIVE_COUNTRY)[T. France]' :   'NATIVE_COUNTRY_France',
'C(NATIVE_COUNTRY)[T. Germany]' :   'NATIVE_COUNTRY_Germany',
'C(NATIVE_COUNTRY)[T. Greece]' :    'NATIVE_COUNTRY_Greece',
'C(NATIVE_COUNTRY)[T. Guatemala]' :  'NATIVE_COUNTRY_Guatemala',
'C(NATIVE_COUNTRY)[T. Haiti]' :       'NATIVE_COUNTRY_Haiti',
'C(NATIVE_COUNTRY)[T. Holand-Netherlands]' :  'NATIVE_COUNTRY_Holand-Netherlands',
'C(NATIVE_COUNTRY)[T. Honduras]' :   'NATIVE_COUNTRY_Honduras',
'C(NATIVE_COUNTRY)[T. Hong]' :    'NATIVE_COUNTRY_Hong',
'C(NATIVE_COUNTRY)[T. Hungary]' :  'NATIVE_COUNTRY_Hungary',
'C(NATIVE_COUNTRY)[T. India]' :   'NATIVE_COUNTRY_India',
'C(NATIVE_COUNTRY)[T. Iran]' :     'NATIVE_COUNTRY_Iran',
'C(NATIVE_COUNTRY)[T. Ireland]' :  'NATIVE_COUNTRY_Ireland',
'C(NATIVE_COUNTRY)[T. Italy]' :   'NATIVE_COUNTRY_Italy',
'C(NATIVE_COUNTRY)[T. Jamaica]' :  'NATIVE_COUNTRY_Jamaica',
'C(NATIVE_COUNTRY)[T. Japan]' :   'NATIVE_COUNTRY_Japan',
'C(NATIVE_COUNTRY)[T. Laos]' :    'NATIVE_COUNTRY_Laos',
'C(NATIVE_COUNTRY)[T. Mexico]' :   'NATIVE_COUNTRY_Mexico',
'C(NATIVE_COUNTRY)[T. Nicaragua]' : 'NATIVE_COUNTRY_Nicaragua',
'C(NATIVE_COUNTRY)[T. Outlying-US(Guam-USVI-etc)]' : 'NATIVE_COUNTRY_Outlying-US_Guam_
'C(NATIVE_COUNTRY)[T. Peru]' :   'NATIVE_COUNTRY_Peru',
'C(NATIVE_COUNTRY)[T. Philippines]' : 'NATIVE_COUNTRY_Philippines',
'C(NATIVE_COUNTRY)[T. Poland]' :    'NATIVE_COUNTRY_Poland',
'C(NATIVE_COUNTRY)[T. Portugal]' :   'NATIVE_COUNTRY_Portugal',
'C(NATIVE_COUNTRY)[T. Puerto-Rico]' : 'NATIVE_COUNTRY_Puerto_Rico',
'C(NATIVE_COUNTRY)[T. Scotland]' : 'NATIVE_COUNTRY_Scotland',
'C(NATIVE_COUNTRY)[T. South]' :  'NATIVE_COUNTRY_South',
'C(NATIVE_COUNTRY)[T. Taiwan]' :  'NATIVE_COUNTRY_Taiwan',
'C(NATIVE_COUNTRY)[T. Thailand]' :     'NATIVE_COUNTRY_Thailand',
'C(NATIVE_COUNTRY)[T. Trinadad&Tobago]' :  'NATIVE_COUNTRY_Trinadad_Tobago',
'C(NATIVE_COUNTRY)[T. United-States]' :   'NATIVE_COUNTRY_United_States',
'C(NATIVE_COUNTRY)[T. Vietnam]' :     'NATIVE_COUNTRY_Vietnam',
'C(NATIVE_COUNTRY)[T. Yugoslavia]' :   'NATIVE_COUNTRY_Yugoslavia'
})
```

```
In [63]: # Check if there is any field which is there in train_data_x but not in test_data_x a
         train_col_set = set(X_train.columns.values.tolist())
         test_col_set = set(X_test.columns.values.tolist())

         train_minus_test_list = list(train_col_set - test_col_set)
         test_minus_train_list = list(test_col_set - train_col_set)
         print("train_minus_test_list = " + str(train_minus_test_list))
         print("test_minus_train_list = " + str(test_minus_train_list))
```

```
train_minus_test_list = ['NATIVE_COUNTRY_Holand-Netherlands']
test_minus_train_list = []
```

```
In [67]: X_train = X_train.drop(['Intercept','NATIVE_COUNTRY_Holand-Netherlands'], axis=1)
         X_test = X_test.drop(['Intercept'], axis=1)
```

```
In [185]: #  One-hot encode the 'train_data' data using pandas.get_dummies()
          categorical = ['WORKCLASS','EDUCATION','MARITAL_STATUS','OCCUPATION','RELATIONSHIP',

          train_data = pd.get_dummies(data = train_data, columns = categorical)

          test_data = pd.get_dummies(data = test_data, columns = categorical)
```

```
In [186]: # Drop the fields TARGET to create dataframe train_data_x
          train_data_x = train_data.drop(['TARGET'], axis=1)

          # Get only filed TARGET to create dataframe train_data_y
          train_data_y = train_data['TARGET']
```

```
In [187]: train_data_x.head()
```

Out[187]:

	AGE	FNLWGT	EDUCATION_NUM	CAPITAL_GAIN	CAPITAL_LOSS	HOURS_PER_WEEK	WO
0	0.301370	0.384197	0.800000	0.667492	0.0	40	
1	0.452055	0.399234	0.800000	0.000000	0.0	13	
2	0.287671	0.597596	0.533333	0.000000	0.0	40	
3	0.493151	0.615275	0.400000	0.000000	0.0	40	
4	0.150685	0.691582	0.800000	0.000000	0.0	40	

	SEX_ Male	NATIVE_COUNTRY_ ?	NATIVE_COUNTRY_ Cambodia	NATIVE_COUNTRY_ Canada	N
0	1	0	0	0	
1	1	0	0	0	
2	1	0	0	0	
3	1	0	0	0	
4	0	0	0	0	

```
In [188]: train_data_y.head()
```

```
Out[188]: 0    0
          1    0
```

```
2    0
3    0
4    0
Name: TARGET, dtype: int64
```

In [189]:
```
# Drop the fields TARGET to create dataframe test_data_x
test_data_x = test_data.drop(['TARGET'], axis=1)

# Get only filed TARGET to create dataframe test_data_y
test_data_y = test_data['TARGET']
```

In [190]:
```
test_data_x.head()
```

Out[190]:
```
        AGE    FNLWGT  EDUCATION_NUM  CAPITAL_GAIN  CAPITAL_LOSS  HOURS_PER_WEEK  WO
0  0.109589  0.599816       0.400000      0.000000           0.0              40
1  0.287671  0.402918       0.533333      0.000000           0.0              50
2  0.150685  0.683958       0.733333      0.000000           0.0              40
3  0.369863  0.526083       0.600000      0.777174           0.0              40
4  0.013699  0.433059       0.600000      0.000000           0.0              30

   SEX_ Male  NATIVE_COUNTRY_ ?  NATIVE_COUNTRY_ Cambodia  NATIVE_COUNTRY_ Canada  N
0          1                  0                         0                       0
1          1                  0                         0                       0
2          1                  0                         0                       0
3          1                  0                         0                       0
4          0                  0                         0                       0
```

In [191]:
```
test_data_y.head()
```

Out[191]:
```
0    0
1    0
2    0
3    0
4    0
Name: TARGET, dtype: int64
```

In [192]:
```
# Check if there is any field which is there in train_data_x but not in test_data_x
train_col_set = set(train_data_x.columns.values.tolist())
test_col_set = set(test_data_x.columns.values.tolist())

train_minus_test_list = list(train_col_set - test_col_set)
test_minus_train_list = list(test_col_set - train_col_set)
```

In [193]:
```
train_minus_test_list
```

Out[193]:
```
['NATIVE_COUNTRY_ Holand-Netherlands']
```

In [194]:
```
test_minus_train_list
```

Out[194]:
```
[]
```

In [195]:
```
train_data_x = train_data_x.drop(['NATIVE_COUNTRY_ Holand-Netherlands'], axis=1)
```

1.21 Apply Supervised Machine Learning Models

The following six supervised learning models that are currently available in scikit-learn are used to train the data:

i. Decision Trees: Decision tree is used for prediction and assessing the relative importance of variables. for the current problem we will need to do prediction for people having >50K income, decision tree can be used

ii. Logistic Regression: logistic regression is a simple model moves with non-linear function hence can work with linearly and non-linearly separable problems

iii. Gaussian Naive Bayes (GaussianNB): Gaussian Naive Bayes is a simple but powerful algorithm for predictive modeling suitable for current problem as we are predicting

iv. Gradient Boosting: Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

Most Kaggle competition winners use stack/ensemble of various models as it gives good performance, this is because it reduces both bias and variance.

v. XGB Boosting: XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

vi. Random Forest: Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees

In [ ]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_recall_fscore_support as score


def calculate_metrics(machine_learning_algorithm, X_train, y_train, X_test, y_test):
    results = {}

    start = time() # Get start time
    machine_learning_algorithm = machine_learning_algorithm
    print( "Doing learner.fit")
    machine_learning_algorithm.fit(X_train, y_train)
    end = time() # Get end time
    print ("Done learner.fit")

    #  Calculate the training time
```

```
    results['train_time'] = end - start
    print( "training time=" + str(results['train_time']))

    # Get the predictions on the test set(X_test),
    #       then get predictions on the first 300 training samples(X_train) using .pre
    start = time() # Get start time
    print ("Doing learner.predict X_test")
    predictions_test = machine_learning_algorithm.predict(X_test)
    print( "Doing learner.predict X_train 300 samples")
    predictions_train = machine_learning_algorithm.predict(X_train)
    end = time() # Get end time

    #  Calculate the total prediction time
    results['prediction_time'] = end - start
    print("prediction time=" + str(results['prediction_time']))

    # Compute training accuracy, precision, recall, fscore, support
    print( "Calculating training accuracy_score")
    results['train_accuracy'] = accuracy_score(predictions_train, y_train)
    print("accuracy_score training data=" + str(results['train_accuracy']))
    print( "Calculating precision, recal, fscore, support")
    results["train_precision"],results["train_recall"], results["train_fscore"], resul

    # Compute test accuracy, precision, recall, fscore, support
    print( "Calculating accuracy_score")
    results['test_accuracy'] = accuracy_score(predictions_test, y_test)
    print("accuracy_score test data=" + str(results['test_accuracy']))
    results["test_precision"],results["test_recall"], results["test_fscore"], results["

    # Success
    #print("{} trained on {} samples.".format((machine_learning_algorithm.__class__.__

    return results
```

In [217]:
```
# Import the six supervised learning models from sklearn
from time import time
from IPython.display import display # Allows the use of display() for DataFrames


from xgboost import XGBClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
# Import supplementary visualization code visuals.py
from sklearn import tree
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB

# Initialize the six models
```

```
model_list = []

model1 = tree.DecisionTreeClassifier(random_state=0)
model_list.append(model1)

model2 = LogisticRegression(random_state=0)
model_list.append(model2)

model3 = GaussianNB()
model_list.append(model3)

model4 = XGBClassifier(random_state=0)
model_list.append(model4)

model5 = GradientBoostingClassifier(random_state=0)
model_list.append(model5)

model6 = RandomForestClassifier(n_estimators=30,random_state=0)
model_list.append(model6)

# Collect results on the learners)
results = {}
for model in model_list:
    model_name = model.__class__.__name__
    print ("Getting results for model_name=" + str(model_name))
    results[model_name] = {}
    results[model_name] = \
        calculate_metrics(model, train_data_x, train_data_y, test_data_x ,test_data_
```

```
Getting results for model_name=DecisionTreeClassifier
Doing learner.fit
Done learner.fit
training time=0.5924162864685059
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=0.050864219665527344
Calculating training accuracy_score
accuracy_score training data=0.9999692884125181
Calculating precision, recal, fscore, support
Calculating accuracy_score
accuracy_score test data=0.7632209323751612
Getting results for model_name=LogisticRegression
Doing learner.fit
```

```
E:\anaconda\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMetricWarning: 
  'precision', 'predicted', average, warn_for)
```

```
Done learner.fit
training time=0.5655145645141602
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=0.09472274780273438
Calculating training accuracy_score
accuracy_score training data=0.846657043702589
Calculating precision, recal, fscore, support
Calculating accuracy_score
accuracy_score test data=0.8059701492537313
Getting results for model_name=GaussianNB
Doing learner.fit
Done learner.fit
training time=0.09774017333984375
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=0.16757655143737793
Calculating training accuracy_score
accuracy_score training data=0.5996437455852093
Calculating precision, recal, fscore, support
Calculating accuracy_score
accuracy_score test data=0.3994840611756035
Getting results for model_name=XGBClassifier
Doing learner.fit
Done learner.fit
training time=7.931801795959473
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
```

```
E:\anaconda\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarning: The trutl
  if diff:
E:\anaconda\lib\site-packages\sklearn\preprocessing\label.py:151: DeprecationWarning: The trutl
  if diff:
E:\anaconda\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMetricWarning: I
  'precision', 'predicted', average, warn_for)
```

```
prediction time=0.26628541946411133
Calculating training accuracy_score
accuracy_score training data=0.8664353060409693
Calculating precision, recal, fscore, support
Calculating accuracy_score
accuracy_score test data=0.8255021190344574
Getting results for model_name=GradientBoostingClassifier
Doing learner.fit
Done learner.fit
training time=7.803111791610718
```

```
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=0.1635885238647461
Calculating training accuracy_score
accuracy_score training data=0.8690457909769356
Calculating precision, recal, fscore, support
Calculating accuracy_score
accuracy_score test data=0.8215097352742461
Getting results for model_name=RandomForestClassifier
Doing learner.fit
Done learner.fit
training time=1.3224265575408936
Doing learner.predict X_test
Doing learner.predict X_train 300 samples
prediction time=0.38300275802612305
Calculating training accuracy_score
accuracy_score training data=0.9982494395135284
Calculating precision, recal, fscore, support
Calculating accuracy_score
accuracy_score test data=0.8046188809041214
```

```
E:\anaconda\lib\site-packages\sklearn\metrics\classification.py:1135: UndefinedMetricWarning: I
  'precision', 'predicted', average, warn_for)
```

```
In [218]: print(results)
```

```
{'DecisionTreeClassifier': {'train_time': 0.5924162864685059, 'prediction_time': 0.05086421966
```

1.22 Comparison of Metrics to different models

Accuracy Score, F score on test data and 300 training samples alongwith Training Time and Prediction time are listed below -

Metric	Decision Tree	Logistics	GaussianNB	XGBBoost	GradientBoost	RandomForest
Test Accuracy	0.7632209	0.8059701	0.39948406	0.82550211	0.8215097	0.8046188
Test F-score	0.660729	0.7193783	0.228066	0.746593	0.7410097	0.717505009
Test Recall	0.763221	0.8059701	0.399484	0.82550211	0.920480	0.80461888
Test Precision	0.582506	0.649587	0.1595875	0.681453	0.8215097	0.64741154
Training Time	0.592416	0.565514	0.097740	7.931801	7.80311	0.3830027
Prediction Time	0.050864	0.094722	0.1675765	0.26628	0.1635885	0.99824943
Training Acuracy	0.999969	0.846657	0.5996437	0.866435	0.8690457	28.703508
Training F-score	0.999969	0.576792	0.576792	0.8734665	0.8753095	0.9982509
Training Recall	1.	0.599644	0.5996437	0.8664353	0.869045	0.9982494
Training Precision	0.999969	0.7583802	0.7583802	0.8878870	0.887937	0.9982563

1.23 Interpretation:

By comparing all the model, found that performance metrics for XGBoost and GradientBoost have almost equally best Test Accuracy and Test F-score and Training Time. But I have chosen GradientBoost has better Test Recall and Test Precision.

1.24 Further Refinement Using GridSearchCV

The model can be further refined using grid search (GridSearchCV) where parameters with different values are provided and grid search finds the best .

In [222]:
```
#  Import 'GridSearchCV', 'make_scorer', and any other necessary libraries
from sklearn.grid_search import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.metrics import f1_score


#  Initialize the classifier
grad_boost_classifier = GradientBoostingClassifier(random_state=0)


parameters= {'n_estimators': [100, 150], 'learning_rate':[ 0.5, 0.2], 'max_depth': [3


#  Perform grid search on the classifier using 'scorer' as the scoring method using G
grid_obj = GridSearchCV(grad_boost_classifier, param_grid=parameters,  scoring='accur

#  Fit the grid search object to the training data and find the optimal parameters us
print("Calling fit on grid_obj")
grid_fit = grid_obj.fit(train_data_x, train_data_y)

# Get the estimator
best_clf = grid_fit.best_estimator_

# Make predictions using the unoptimized and model
print("Calling predict")
predictions = (grad_boost_classifier.fit(train_data_x, train_data_y)).predict(test_da
best_predictions = best_clf.predict(test_data_x)

best_parameters = grid_obj.best_estimator_.get_params()
print("Best parameters are:")
for param_name in sorted(parameters.keys()):
    print('\t%s: %r' % (param_name, best_parameters[param_name]))

# Report the before-and-afterscores
print("Unoptimized model\n------")
print("Accuracy score on testing data: {:.4f}".format(accuracy_score(test_data_y, pre
print("\nOptimized Model\n------")
print("Final accuracy score on the testing data: {:.4f}".format(accuracy_score(test_d
```

```
Calling fit on grid_obj
Calling predict
Best parameters are:
        learning_rate: 0.2
        max_depth: 3
        n_estimators: 150
Unoptimized model
------
Accuracy score on testing data: 0.8215

Optimized Model
------
Final accuracy score on the testing data: 0.8117
```

1.25 Extracting Feature Importance

Choose a scikit-learn supervised learning algorithm ExtraTreesClassifier that has a feature_importance_ attribute availble for it. This attribute is a function that ranks the importance of each feature when making predictions based on the chosen algorithm.

In the code cell below, I have implemented the following:

```
Import a supervised learning model ExtraTreesClassifier from sklearn
Train the supervised model on the entire training set.
Extract the feature importances using '.feature_importances_'.
```

```
In [226]: from sklearn.ensemble import ExtraTreesClassifier


          #  Train the supervised model on the training set using .fit(train_data_x, train_data
          print("defining ExtraTreesClassifier")
          model = ExtraTreesClassifier(random_state = 0)
          print("fit using ExtraTreesClassifier")
          model.fit(train_data_x, train_data_y)
          print("fit done")

          #  Extract the feature importances using .feature_importances_
          importances = model.feature_importances_
          #print("important features=" + str(importances))

          import pandas as pd
          feature_importances = pd.DataFrame(importances, index = train_data_x.columns,
                                             columns=['importance']).sort_values('importance',

          feature_importances
```

```
defining ExtraTreesClassifier
fit using ExtraTreesClassifier
```

fit done

Out[226]:

	importance
FNLWGT	0.168785
AGE	0.149096
HOURS_PER_WEEK	0.091448
MARITAL_STATUS_ Married-civ-spouse	0.091281
CAPITAL_GAIN	0.059426
EDUCATION_NUM	0.049366
RELATIONSHIP_ Husband	0.037491
MARITAL_STATUS_ Never-married	0.032884
CAPITAL_LOSS	0.021454
OCCUPATION_ Exec-managerial	0.020789
OCCUPATION_ Prof-specialty	0.020355
EDUCATION_ Bachelors	0.015557
SEX_ Female	0.011110
EDUCATION_ HS-grad	0.010945
WORKCLASS_ Private	0.010221
WORKCLASS_ Self-emp-not-inc	0.008745
EDUCATION_ Some-college	0.007862
RELATIONSHIP_ Not-in-family	0.007718
RELATIONSHIP_ Wife	0.007573
WORKCLASS_ Self-emp-inc	0.007460
MARITAL_STATUS_ Divorced	0.006910
RACE_ White	0.006909
OCCUPATION_ Other-service	0.006777
EDUCATION_ Masters	0.006760
RELATIONSHIP_ Own-child	0.006713
NATIVE_COUNTRY_ United-States	0.006345
OCCUPATION_ Sales	0.006243
SEX_ Male	0.006118
WORKCLASS_ Federal-gov	0.005619
OCCUPATION_ Craft-repair	0.005550
WORKCLASS_ Local-gov	0.005526
OCCUPATION_ Tech-support	0.004886
OCCUPATION_ Adm-clerical	0.004799
EDUCATION_ Prof-school	0.004784
RACE_ Black	0.004758
WORKCLASS_ State-gov	0.004527
EDUCATION_ Doctorate	0.004124
OCCUPATION_ Handlers-cleaners	0.003995
OCCUPATION_ Transport-moving	0.003922
OCCUPATION_ Farming-fishing	0.003918
OCCUPATION_ Machine-op-inspct	0.003868
EDUCATION_ Assoc-voc	0.003111
NATIVE_COUNTRY_ ?	0.003104
RACE_ Asian-Pac-Islander	0.002956

EDUCATION_ 7th-8th	0.002872
RELATIONSHIP_ Unmarried	0.002591
EDUCATION_ Assoc-acdm	0.002506
OCCUPATION_ Protective-serv	0.002438
EDUCATION_ 10th	0.002110
RELATIONSHIP_ Other-relative	0.002001
NATIVE_COUNTRY_ Mexico	0.001897
OCCUPATION_ ?	0.001745
EDUCATION_ 11th	0.001595
MARITAL_STATUS_ Separated	0.001495
WORKCLASS_ ?	0.001483
EDUCATION_ 9th	0.001474
RACE_ Amer-Indian-Eskimo	0.001468
NATIVE_COUNTRY_ Canada	0.001357
MARITAL_STATUS_ Widowed	0.001338
NATIVE_COUNTRY_ Germany	0.001189
NATIVE_COUNTRY_ Philippines	0.001119
NATIVE_COUNTRY_ England	0.001114
RACE_ Other	0.001085
MARITAL_STATUS_ Married-spouse-absent	0.001064
EDUCATION_ 12th	0.001049
NATIVE_COUNTRY_ Italy	0.000978
NATIVE_COUNTRY_ India	0.000862
NATIVE_COUNTRY_ Cuba	0.000826
EDUCATION_ 5th-6th	0.000682
NATIVE_COUNTRY_ Japan	0.000626
NATIVE_COUNTRY_ South	0.000623
NATIVE_COUNTRY_ Puerto-Rico	0.000595
NATIVE_COUNTRY_ China	0.000550
NATIVE_COUNTRY_ Poland	0.000540
NATIVE_COUNTRY_ Greece	0.000473
NATIVE_COUNTRY_ Jamaica	0.000471
NATIVE_COUNTRY_ Iran	0.000468
NATIVE_COUNTRY_ El-Salvador	0.000419
NATIVE_COUNTRY_ Vietnam	0.000360
EDUCATION_ 1st-4th	0.000339
NATIVE_COUNTRY_ Cambodia	0.000336
NATIVE_COUNTRY_ France	0.000317
NATIVE_COUNTRY_ Columbia	0.000301
NATIVE_COUNTRY_ Yugoslavia	0.000297
MARITAL_STATUS_ Married-AF-spouse	0.000286
NATIVE_COUNTRY_ Ireland	0.000243
NATIVE_COUNTRY_ Taiwan	0.000241
NATIVE_COUNTRY_ Ecuador	0.000239
NATIVE_COUNTRY_ Portugal	0.000210
NATIVE_COUNTRY_ Haiti	0.000208
NATIVE_COUNTRY_ Dominican-Republic	0.000194
NATIVE_COUNTRY_ Hungary	0.000186

```
OCCUPATION_ Priv-house-serv                  0.000185
NATIVE_COUNTRY_ Peru                         0.000149
NATIVE_COUNTRY_ Laos                         0.000137
NATIVE_COUNTRY_ Guatemala                    0.000126
NATIVE_COUNTRY_ Hong                         0.000126
WORKCLASS_ Without-pay                       0.000114
NATIVE_COUNTRY_ Scotland                     0.000105
NATIVE_COUNTRY_ Nicaragua                    0.000103
NATIVE_COUNTRY_ Thailand                     0.000092
NATIVE_COUNTRY_ Trinadad&Tobago              0.000086
EDUCATION_ Preschool                         0.000061
NATIVE_COUNTRY_ Outlying-US(Guam-USVI-etc)   0.000042
OCCUPATION_ Armed-Forces                     0.000016
NATIVE_COUNTRY_ Honduras                     0.000005
WORKCLASS_ Never-worked                      0.000003
```

1.26 Further Refinement: Neural Network

For refinement to the model I tried to build Deep Neural Nework with 256 x 256 hidden layer, MinibatchGradient with the following:

```
AdamOptimizer
batch size = 256
epochs = 25
learning rate =0.01
keep_probability = 0.8
```

In [199]:
```
def one_hot_encode(x):
    """
    One hot encode a list of sample labels. Return a one-hot encoded vector for each
    : x: List of sample Labels
    : return: Numpy array of one-hot encoded labels
    """

    max_value = 2
    if not "encode_map" in globals():
        global encode_map
        encode_map = {}

    value = encode_map.get(str(x))
    if value is not None:
        return value

    hot_encode = np.eye(max_value)[x]
    encode_map[str(x)] = hot_encode
    return hot_encode
```

In [200]:
```
train_data_y_encode = one_hot_encode(train_data_y)
test_data_y_encode = one_hot_encode(test_data_y)
```

```
print("Y_train_encode.shape:" + str(train_data_y_encode.shape))
print("Y_test_encode.shape:" + str(test_data_y_encode.shape))
```

```
Y_train_encode.shape:(32561, 2)
Y_test_encode.shape:(16281, 2)
```

In [201]:
```
import tensorflow as tf
def deep_neural_network(x, weights, biases,  keep_prob):
    hidden_layer1 = tf.add(tf.matmul(x, weights["w1"]), biases["b1"])
    hidden_layer1 = tf.nn.relu(hidden_layer1)
    hidden_layer1 = tf.nn.dropout(hidden_layer1, keep_prob)
    hidden_layer2 = tf.matmul(hidden_layer1, weights["w2"]) + biases["b2"]
    hidden_layer2 = tf.nn.relu(hidden_layer2)
    hidden_layer2 = tf.nn.dropout(hidden_layer2, keep_prob)
    output_layer = tf.matmul(hidden_layer2, weights["out"]) + biases["out"]
    return output_layer
```

In [212]:
```
hidden_layer1_length = 256
hidden_layer2_length = 256
input_feature_size = train_data_x.shape[1]
output_classes = 2
learning_rate = 0.001
epochs = 50
batch_size = 256
keep_prob_value = 0.8

weights = {
    'w1': tf.Variable(tf.random_normal([input_feature_size, hidden_layer1_length])),
    'w2': tf.Variable(tf.random_normal([ hidden_layer1_length, hidden_layer2_length]
    'out': tf.Variable(tf.random_normal([hidden_layer2_length, output_classes]))
}

biases = {
    'b1': tf.Variable(tf.random_normal([hidden_layer1_length])),
    'b2': tf.Variable(tf.random_normal([hidden_layer2_length])),
    'out': tf.Variable(tf.random_normal([output_classes]))
}

keep_prob = tf.placeholder("float")


x = tf.placeholder("float", [None, input_feature_size])
y = tf.placeholder("float", [None, output_classes])

logits = deep_neural_network(x, weights, biases, keep_prob)

# Loss and Optimizer
```

```
        cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=y
        optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

        # Accuracy
        correct_pred = tf.equal(tf.argmax(logits, 1), tf.argmax(y, 1))
        accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32), name='accuracy')
```

```
In [213]: def print_stats(session, valid_features, valid_labels, cost, accuracy):
              loss = sess.run(cost,
                              feed_dict={
                                  x: valid_features,
                                  y: valid_labels,
                                  keep_prob: 1.
                              })
              validation_accuracy = sess.run(accuracy,
                                   feed_dict={
                                       x: valid_features,
                                       y: valid_labels,
                                       keep_prob: 1.
                                   })

              print('Loss: {:>10.4f} Training Validation Accuracy: {:.6f}'.format(loss, valida
```

```
In [215]: print("dimension of X_train=" + str(train_data_x.shape))
          print("dimension of Y_train=" + str(train_data_y_encode.shape))
          print("dimension of X_test=" + str(test_data_x.shape))
          print("dimension of Y_test=" + str(train_data_y_encode.shape))

          import sklearn
          predictions_proba = None
          y_p = None
          y_pred = None
          y_true = None
          with tf.Session() as sess:
              sess.run(tf.global_variables_initializer())

              for epoch in range(epochs):
                  avg_cost = 0.0
                  total_batch = int(len(train_data_x) / batch_size)
                  x_batches = np.array_split(train_data_x, total_batch)
                  y_batches = np.array_split(train_data_y_encode, total_batch)
                  for i in range(total_batch):
                      batch_x, batch_y = x_batches[i], y_batches[i]
                      _, c = sess.run([optimizer, cost],
                                      feed_dict={
                                          x: batch_x,
                                          y: batch_y,
                                          keep_prob: keep_prob_value
```

```
                                })

                    print('Epoch {:>2}:  '.format(epoch + 1), end='')
                    print_stats(sess, batch_x, batch_y, cost, accuracy)


                print("Neutral Network training done!")

                print("Training Accuracy:", accuracy.eval({x: train_data_x, y: train_data_y_encod
                print("Testing Accuracy:", accuracy.eval({x:test_data_x, y: test_data_y_encode, k
```

```
dimension of X_train=(32561, 107)
dimension of Y_train=(32561, 2)
dimension of X_test=(16281, 107)
dimension of Y_test=(32561, 2)
Epoch  1:  Loss:   297.1618 Training Validation Accuracy: 0.785156
Epoch  2:  Loss:   241.8817 Training Validation Accuracy: 0.792969
Epoch  3:  Loss:   155.9109 Training Validation Accuracy: 0.800781
Epoch  4:  Loss:    89.7025 Training Validation Accuracy: 0.835938
Epoch  5:  Loss:    51.6288 Training Validation Accuracy: 0.839844
Epoch  6:  Loss:    39.4984 Training Validation Accuracy: 0.835938
Epoch  7:  Loss:    48.5257 Training Validation Accuracy: 0.820312
Epoch  8:  Loss:    50.8458 Training Validation Accuracy: 0.800781
Epoch  9:  Loss:    29.1506 Training Validation Accuracy: 0.789062
Epoch 10:  Loss:    15.4280 Training Validation Accuracy: 0.804688
Epoch 11:  Loss:     6.7027 Training Validation Accuracy: 0.824219
Epoch 12:  Loss:     6.0752 Training Validation Accuracy: 0.808594
Epoch 13:  Loss:     3.0493 Training Validation Accuracy: 0.816406
Epoch 14:  Loss:     1.3972 Training Validation Accuracy: 0.828125
Epoch 15:  Loss:     1.1382 Training Validation Accuracy: 0.808594
Epoch 16:  Loss:     0.4909 Training Validation Accuracy: 0.835938
Epoch 17:  Loss:     0.8516 Training Validation Accuracy: 0.789062
Epoch 18:  Loss:     0.4367 Training Validation Accuracy: 0.789062
Epoch 19:  Loss:     0.4011 Training Validation Accuracy: 0.835938
Epoch 20:  Loss:     0.4315 Training Validation Accuracy: 0.789062
Epoch 21:  Loss:     0.4476 Training Validation Accuracy: 0.789062
Epoch 22:  Loss:     0.4485 Training Validation Accuracy: 0.789062
Epoch 23:  Loss:     0.4637 Training Validation Accuracy: 0.789062
Epoch 24:  Loss:     0.4353 Training Validation Accuracy: 0.789062
Epoch 25:  Loss:     0.4404 Training Validation Accuracy: 0.789062
Epoch 26:  Loss:     0.4542 Training Validation Accuracy: 0.789062
Epoch 27:  Loss:     0.4369 Training Validation Accuracy: 0.789062
Epoch 28:  Loss:     0.4586 Training Validation Accuracy: 0.789062
Epoch 29:  Loss:     0.4390 Training Validation Accuracy: 0.789062
Epoch 30:  Loss:     0.4323 Training Validation Accuracy: 0.789062
Epoch 31:  Loss:     0.4332 Training Validation Accuracy: 0.789062
Epoch 32:  Loss:     0.4347 Training Validation Accuracy: 0.789062
```

```
Epoch 33:  Loss:     0.4337 Training Validation Accuracy: 0.789062
Epoch 34:  Loss:     0.4645 Training Validation Accuracy: 0.789062
Epoch 35:  Loss:     0.4422 Training Validation Accuracy: 0.789062
Epoch 36:  Loss:     0.4400 Training Validation Accuracy: 0.789062
Epoch 37:  Loss:     0.4616 Training Validation Accuracy: 0.789062
Epoch 38:  Loss:     0.4664 Training Validation Accuracy: 0.789062
Epoch 39:  Loss:     0.4439 Training Validation Accuracy: 0.789062
Epoch 40:  Loss:     0.4425 Training Validation Accuracy: 0.789062
Epoch 41:  Loss:     0.4489 Training Validation Accuracy: 0.789062
Epoch 42:  Loss:     0.4272 Training Validation Accuracy: 0.789062
Epoch 43:  Loss:     0.4469 Training Validation Accuracy: 0.789062
Epoch 44:  Loss:     0.4413 Training Validation Accuracy: 0.789062
Epoch 45:  Loss:     0.4276 Training Validation Accuracy: 0.789062
Epoch 46:  Loss:     0.4377 Training Validation Accuracy: 0.789062
Epoch 47:  Loss:     0.4099 Training Validation Accuracy: 0.789062
Epoch 48:  Loss:     0.4353 Training Validation Accuracy: 0.789062
Epoch 49:  Loss:     0.4247 Training Validation Accuracy: 0.789062
Epoch 50:  Loss:     0.4547 Training Validation Accuracy: 0.789062
Neutral Network training done!
Training Accuracy: 0.7592519
Testing Accuracy: 0.99987715
```

1.27 Conclusion:

I found that Neural Network is giving the best test accuracy score of 99.98%, which is far better than six models that i tried. We can use the Neural Network if the business requirement does not need explinable model. Otherwise, we can use GradientBoost model