In [69]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [70]:
```python
df = pd.read_csv('adult.csv')
df.head()
```

Out[70]:

	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United-States	<=50K

In [71]:
```python
df.shape
```

Out[71]:
```
(32560, 15)
```

Renaming the Columns.

In [72]:
```python
df.columns = ['Age', 'Workclass', 'Fnlwgt', 'Education', 'education_num', 'marital_status', 'occupation', 'relationship', 'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week', 'native_country', 'income' ]
df.head()
```

Out[72]:

	Age	Workclass	Fnlwgt	Education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week	native_country	income
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	United-States	<=50K

In [73]:
```python
df.columns
```

Out[73]:
```
Index(['Age', 'Workclass', 'Fnlwgt', 'Education', 'education_num',
       'marital_status', 'occupation', 'relationship', 'race', 'sex',
       'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
       'income'],
      dtype='object')
```

In [74]:
```python
len(df.columns)
```

Out[74]:
```
15
```

In [75]:
```python
df.dtypes
```

Out[75]:
```
Age                int64
Workclass         object
Fnlwgt             int64
Education         object
education_num      int64
marital_status    object
occupation        object
relationship      object
race              object
sex               object
capital_gain       int64
capital_loss       int64
hours_per_week     int64
native_country    object
income            object
dtype: object
```

Checking whether the dataset has Null Values.

In [76]:
```python
df.isnull().sum()
```

Out[76]:
```
Age               0
Workclass         0
Fnlwgt            0
Education         0
education_num     0
marital_status    0
occupation        0
relationship      0
race              0
sex               0
capital_gain      0
capital_loss      0
hours_per_week    0
native_country    0
income            0
dtype: int64
```

Note :

Since there is no null values so that we can move to other steps.

Converting all 'object' parameters into 'integer' values.

In [124]:
```python
from sklearn.preprocessing import LabelEncoder
```

In [78]:
```python
lb = LabelEncoder()
df['Workclass'] = lb.fit_transform(df['Workclass'])
df['Education'] = lb.fit_transform(df['Education'])
df['marital_status'] = lb.fit_transform(df['marital_status'])
df['relationship'] = lb.fit_transform(df['relationship'])
df['race'] = lb.fit_transform(df['race'])
df['sex'] = lb.fit_transform(df['sex'])
df['native_country'] = lb.fit_transform(df['native_country'])
df['occupation'] = lb.fit_transform(df['occupation'])
df['income'] = lb.fit_transform(df['income'])
```

In [79]:
```python
print(df['Workclass'].value_counts())
print(df['Education'].value_counts())
print(df['marital_status'].value_counts())
print(df['relationship'].value_counts())
print(df['race'].value_counts())
print(df['sex'].value_counts())
print(df['native_country'].value_counts())
print(df['occupation'].value_counts())
print(df['income'].value_counts())
```

```
4    22696
6     2541
2     2093
0     1836
7     1297
5     1116
1      960
8       14
3        7
Name: Workclass, dtype: int64
11    10501
15     7291
9      5354
12     1723
8      1382
1      1175
7      1067
0       933
5       646
14      576
6       514
2       433
10      413
4       333
3       168
13       51
Name: Education, dtype: int64
2    14976
4    10682
0     4443
5     1025
6      993
3      418
1       23
Name: marital_status, dtype: int64
0    13193
1     8304
3     5068
4     3446
5     1568
2      981
Name: relationship, dtype: int64
4    27815
2     3124
1     1039
0      311
3      271
Name: race, dtype: int64
1    21789
0    10771
Name: sex, dtype: int64
39    29169
26      643
0       583
30      198
11      137
2       121
33      114
8       106
19      100
5        95
9        90
23       81
35       80
3        75
22       73
6        70
40       67
13       64
24       62
31       60
4        59
36       51
14       44
20       43
32       37
27       34
29       31
10       29
12       29
7        28
21       24
17       20
1        19
38       19
25       18
37       18
41       16
28       14
16       13
18       13
34       12
15        1
Name: native_country, dtype: int64
10    4140
3     4099
4     4066
1     3769
12    3650
8     3295
7     2002
0     1843
14    1597
6     1370
5      994
13     928
11     649
9      149
2        9
Name: occupation, dtype: int64
0    24719
1     7841
Name: income, dtype: int64
```

In [80]:
```python
df.dtypes
```

Out[80]:
```
Age               int64
Workclass         int32
Fnlwgt            int64
Education         int32
education_num     int64
marital_status    int32
occupation        int32
relationship      int32
race              int32
sex               int32
capital_gain      int64
capital_loss      int64
hours_per_week    int64
native_country    int32
income            int32
dtype: object
```

Splitting the dataset into Training Data and Test Data.

In [81]:
```python
x = df.iloc[:,:-1]
y = df.iloc[:,-1]
print(type(x))
print(type(y))
print(x.shape)
print(y.shape)
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
(32560, 14)
(32560,)
```

In [82]:
```python
from sklearn.model_selection import train_test_split
```

In [83]:
```python
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.25)
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(24420, 14)
(8140, 14)
(24420,)
(8140,)
```

In [84]:
```python
print(x.shape)
print(y.shape)
```

```
(32560, 14)
(32560,)
```

In [85]:
```python
x_train.head()
```

Out[85]:

	Age	Workclass	Fnlwgt	Education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week	native_country
20863	44	7	185832	9	13	2	1	0	4	1	0	0	46	39
8776	47	4	121124	6	5	2	7	0	4	1	0	0	40	39
29067	24	4	113515	11	9	4	4	1	4	0	0	0	40	39
18608	18	4	173255	11	9	4	14	3	4	1	1055	0	25	39
1063	49	4	175958	15	10	2	12	0	4	1	0	0	80	39

In [86]:
```python
y_train.head()
```

Out[86]:
```
20863    1
8776     0
29067    0
18608    0
1063     1
Name: income, dtype: int32
```

Different Training Models.

a. Decision Tree Classifier.

In [87]:
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix,classification_report
```

In [88]:
```python
s1 = DecisionTreeClassifier(criterion = 'gini', max_depth = 15)
s1.fit(x_train,y_train)
```

Out[88]:
```
DecisionTreeClassifier(max_depth=15)
```

In [89]:
```python
print('Training Score', s1.score(x_train,y_train))
print('Testing Score', s1.score(x_test,y_test))
```

```
Training Score 0.915028665028665
Testing Score 0.8339066339066339
```

Income Prediction for Test Data.

In [90]:
```python
ypredict_s1 = s1.predict(x_test)
print(ypredict_s1)
```

```
[0 0 0 ... 0 0 0]
```

Confusion Matrix and Classification Report.

In [127]:
```python
conf_m1 = confusion_matrix(y_test,ypredict_s1)
print('Confusion Matrix \n')
print(conf_m1)
print('\n')
print('Classification Report \n')
print(classification_report(y_test,ypredict_s1))
```

```
Confusion Matrix

[[5563  591]
 [ 761 1225]]


Classification Report

              precision    recall  f1-score   support

           0       0.88      0.90      0.89      6154
           1       0.67      0.62      0.64      1986

    accuracy                           0.83      8140
   macro avg       0.78      0.76      0.77      8140
weighted avg       0.83      0.83      0.83      8140
```

Validating the Result for Precision, Recall, F1-Score and Accuracy.

Precision = TP/(TP+FP), TN/(TN+FN)

Recall = TP/(TP+FN), TN/(TN+FP)

F1-Score = (2*Precision* Recall)/(Precision + Recall)

Accuracy = (TP+TN)/(TP+FN+FP+TN)

In [118]:
```python
pre0 = 5563/(5563+761)
pre1 = 1255/(1255+591)
rec0 = 5563/(5563+591)
rec1 = 1255/(1255+761)
print('Precision 0 :', pre0)
print('\n')
print('Precision 1 :', pre1)
print('\n')
print('Recall 0    :', rec0)
print('\n')
print('Recall 1    :', rec1)
print('\n')
f1s0 = 2*pre0*rec0/(pre0 + rec0)
f1s1 = 2*pre1*rec1/(pre1 + rec1)
print('F1-score 0  :', f1s0)
print('\n')
print('F1-score 1  :', f1s1)
print('\n')
acc= (5563+1255)/(5563+591+1255+761)
print('Accuracy    :', acc)
```

```
Precision 0 : 0.8796647691334598

Precision 1 : 0.6798483206933911

Recall 0    : 0.903964900877478

Recall 1    : 0.6225198412698413

F1-score 0  : 0.8916493027728802

F1-score 1  : 0.6499223200414294

Accuracy    : 0.8345165238878091
```

Percentage of Misclassification.

Percentage of Misclassification = (FP+TN)/(TP+TN+FP+FN)

In [119]:
```python
per_mis = (761+591)/(5563+591+1255+761)
print('Percentage of Misclassification :', per_mis*100)
```

```
Percentage of Misclassification : 16.548347613219093
```

b. Random Forest Classifier.

In [94]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix,classification_report
```

In [95]:
```python
s2 = RandomForestClassifier(n_estimators = 80, criterion = 'gini', max_depth = 15)
s2.fit(x_train,y_train)
```

Out[95]:
```
RandomForestClassifier(max_depth=15, n_estimators=80)
```

In [96]:
```python
print('Training Score', s2.score(x_train,y_train))
print('Testing Score', s2.score(x_test,y_test))
```

```
Training Score 0.9124078624078624
Testing Score 0.8635135135135135
```

Income Prediction for Test Data.

In [97]:
```python
ypredict_s2 = s2.predict(x_test)
print(ypredict_s2)
```

```
[0 0 0 ... 0 0 0]
```

Confusion Matrix and Classification Report.

In [98]:
```python
conf_m2 = confusion_matrix(y_test,ypredict_s2)
print('Confusion Matrix :\n')
print(conf_m2)
print('\n')
print('Classification Report :\n')
print(classification_report(y_test,ypredict_s2))
```

```
Confusion Matrix :

[[5805  349]
 [ 762 1224]]


Classification Report :

              precision    recall  f1-score   support

           0       0.88      0.94      0.91      6154
           1       0.78      0.62      0.69      1986

    accuracy                           0.86      8140
   macro avg       0.83      0.78      0.80      8140
weighted avg       0.86      0.86      0.86      8140
```

Validating the Result for Precision, Recall, F1-Score and Accuracy.

In [117]:
```python
pre0 = 5805/(5805+762)
pre1 = 1224/(1224+349)
rec0 = 5805/(5805+349)
rec1 = 1224/(1224+762)
print('Precision 0 :', pre0)
print('\n')
print('Precision 1 :', pre1)
print('\n')
print('Recall 0    :', rec0)
print('\n')
print('Recall 1    :', rec1)
print('\n')
f1s0 = 2*pre0*rec0/(pre0 + rec0)
f1s1 = 2*pre1*rec1/(pre1 + rec1)
print('F1-score 0  :', f1s0)
print('\n')
print('F1-score 1  :', f1s1)
print('\n')
acc= (5805+1224)/(5805+349+1224+762)
print('Accuracy    :', acc)
```

```
Precision 0 : 0.8839652809502055

Precision 1 : 0.7781309599491417

Recall 0    : 0.9432889177770556

Recall 1    : 0.6163141993957704

F1-score 0  : 0.9126640987343762

F1-score 1  : 0.6878336611407698

Accuracy    : 0.8635135135135135
```

Percentage of Misclassification.

In [120]:
```python
per_mis = (762+349)/(5805+349+1224+762)
print('Percentage of Misclassification :', per_mis*100)
```

```
Percentage of Misclassification : 13.648648648648647
```

c. KNN Classifier.

In [101]:
```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix,classification_report
```

In [102]:
```python
s3 = KNeighborsClassifier(n_neighbors = 18)
s3.fit(x_train,y_train)
```

Out[102]:
```
KNeighborsClassifier(n_neighbors=18)
```

In [103]:
```python
print('Training Score', s3.score(x_train,y_train))
print('Testing Score', s3.score(x_test,y_test))
```

```
Training Score 0.8054463554463555
Testing Score 0.7963144963144964
```

Income Prediction for Test Data.

In [104]:
```python
ypredict_s3 = s3.predict(x_test)
print(ypredict_s3)
```

```
[0 0 0 ... 0 0 0]
```

Confusion Matrix and Classification Report.

In [105]:
```python
conf_m3 = confusion_matrix(y_test,ypredict_s3)
print('Confusion Matrix :\n')
print(conf_m3)
print('\n')
print('Classification Report :\n')
print(classification_report(y_test,ypredict_s3))
```

```
Confusion Matrix :

[[6051  103]
 [1555  431]]


Classification Report :

              precision    recall  f1-score   support

           0       0.80      0.98      0.88      6154
           1       0.81      0.22      0.34      1986

    accuracy                           0.80      8140
   macro avg       0.80      0.60      0.61      8140
weighted avg       0.80      0.80      0.75      8140
```

Validating the Result for Precision, Recall, F1-Score and Accuracy.

In [116]:
```python
pre0 = 6051/(6051+1555)
pre1 = 431/(431+103)
rec0 = 6051/(6051+103)
rec1 = 431/(431+1555)
print('Precision 0 :', pre0)
print('\n')
print('Precision 1 :', pre1)
print('\n')
print('Recall 0    :', rec0)
print('\n')
print('Recall 1    :', rec1)
print('\n')
f1s0 = 2*pre0*rec0/(pre0 + rec0)
f1s1 = 2*pre1*rec1/(pre1 + rec1)
print('F1-score 0  :', f1s0)
print('\n')
print('F1-score 1  :', f1s1)
print('\n')
acc= (6051+431)/(6051+103+431+1555)
print('Accuracy    :', acc)
```

```
Precision 0 : 0.7955561398895609

Precision 1 : 0.8071161048689138

Recall 0    : 0.9832629184270393

Recall 1    : 0.21701913393756295

F1-score 0  : 0.8795058139534884

F1-score 1  : 0.34206349206349207

Accuracy    : 0.7963144963144964
```

Percentage of Misclassification.

In [121]:
```python
per_mis = (1555+103)/(6051+103+1555+431)
print('Percentage of Misclassification :', per_mis*100)
```

```
Percentage of Misclassification : 20.36855036855037
```

d. Logistic Regression.

In [108]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,classification_report
```

In [109]:
```python
s4 = LogisticRegression(solver = 'liblinear')
s4.fit(x_train,y_train)
```

Out[109]:
```
LogisticRegression(solver='liblinear')
```

In [110]:
```python
print('Training Score', s3.score(x_train,y_train))
print('Testing Score', s3.score(x_test,y_test))
```

```
Training Score 0.8054463554463555
Testing Score 0.7963144963144964
```

Income Prediction for Test Data.

In [111]:
```python
ypredict_s4 = s4.predict(x_test)
print(ypredict_s4)
```

```
[0 0 0 ... 0 0 0]
```

Confusion Matrix and Classification Report.

In [112]:
```python
conf_m4 = confusion_matrix(y_test,ypredict_s4)
print('Confusion Matrix :\n')
print(conf_m4)
print('\n')
print('Classification Report : \n')
print(classification_report(y_test,ypredict_s4))
```

```
Confusion Matrix :

[[5869  285]
 [1416  570]]


Classification Report :

              precision    recall  f1-score   support

           0       0.81      0.95      0.87      6154
           1       0.67      0.29      0.40      1986

    accuracy                           0.79      8140
   macro avg       0.74      0.62      0.64      8140
weighted avg       0.77      0.79      0.76      8140
```

Validating the Result for Precision, Recall, F1-Score and Accuracy.

In [115]:
```python
pre0 = 5869/(5869+1416)
pre1 = 570/(570+285)
rec0 = 5869/(5869+285)
rec1 = 570/(570+1416)
print('Precision 0 :', pre0)
print('\n')
print('Precision 1 :', pre1)
print('\n')
print('Recall 0    : ', rec0)
print('\n')
print('Recall 1    :', rec1)
print('\n')
f1s0 = 2*pre0*rec0/(pre0 + rec0)
f1s1 = 2*pre1*rec1/(pre1 + rec1)
print('F1-score 0  :', f1s0)
print('\n')
print('F1-score 1  :', f1s1)
print('\n')
acc= (5869+570)/(5869+285+570+1416)
print('Accuracy    :', acc)
```

```
Precision 0 : 0.8056280027453672

Precision 1 : 0.6666666666666666

Recall 0    :  0.9536886577835554

Recall 1    : 0.28700906344410876

F1-score 0  : 0.8734280824466105

F1-score 1  : 0.4012671594508976

Accuracy    : 0.7910319410319411
```

Percentage of Misclassification.

In [122]:
```python
per_mis = (1416+285)/(5869+285+570+1416)
print('Percentage of Misclassification :', per_mis*100)
```

```
Percentage of Misclassification : 20.896805896805898
```

e. SVM Classifier.

In [47]:
```python
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix,classification_report
```

In [48]:
```python
s5 = SVC(kernel = 'linear', C=1)
s5.fit(x_train,y_train)
```

Out[48]:
```
SVC(C=1, kernel='linear')
```

In [49]:
```python
print('Training Score', s5.score(x_train,y_train))
print('Testing Score', s5.score(x_test,y_test))
```

```
Training Score 0.7938165438165438
Testing Score 0.7976658476658477
```

Income Prediction for Test Data.

In [50]:
```python
ypredict_s5 = s5.predict(x_test)
print(ypredict_s5)
```

```
[0 1 0 ... 0 0 0]
```

Confusion Matrix and Classification Report.

In [57]:
```python
conf_m4 = confusion_matrix(y_test,ypredict_s4)
print('Confusion Matrix :\n')
print(conf_m4)
print('\n')
print('Classification Report : \n')
print(classification_report(y_test,ypredict_s4))
```

```
Confusion Matrix :

[[5923  274]
 [1391  552]]


Classification Report :

              precision    recall  f1-score   support

           0       0.81      0.96      0.88      6197
           1       0.67      0.28      0.40      1943

    accuracy                           0.80      8140
   macro avg       0.74      0.62      0.64      8140
weighted avg       0.78      0.80      0.76      8140
```

Validating the Result for Precision, Recall, F1-Score and Accuracy.

In [58]:
```python
pre0 = 5923/(5923+1391)
pre1 = 552/(552+274)
rec0 = 5923/(5923+274)
rec1 = 552/(552+1391)
print('Precision 0 :', pre0)
print('\n')
print('Precision 1 :', pre1)
print('\n')
print('Recall 0    : ', rec0)
print('\n')
print('Recall 1    :', rec1)
print('\n')
f1s0 = 2*pre0*rec0/(pre0 + rec0)
f1s1 = 2*pre1*rec1/(pre1 + rec1)
print('F1-score 0  :', f1s0)
print('\n')
print('F1-score 1  :', f1s1)
print('\n')
acc= (5923+552)/(5923+274+1391+552)
print('Accuracy    :', acc)
```

```
Precision 0 : 0.8098167897183484

Precision 1 : 0.6682808716707022

Recall 0    :  0.9557850572857834

Recall 1    : 0.28409675759135355

F1-score 0  : 0.876767078766337

F1-score 1  : 0.39869989165763814

Accuracy    : 0.7954545454545454
```

Percentage of Misclassification.

In [123]:
```python
per_mis = (1391+274)/(5923+274+1391+552)
print('Percentage of Misclassification :', per_mis*100)
```

```
Percentage of Misclassification : 20.454545454545457
```

Model With Best Accuracy.

From the above Machine Learning Models,

Random Forest Classifier Model has an Accuracy of '0.86' and Misclassification of approximately '14%'.