Abstract :

The main aim is to Create a classification model to predict whether a person makes over $50k a year

Data Understanding :

In [1]:
```python
import pandas as pd
a=['Age', 'Workclass', 'Fnlwgt', 'Education', 'education_num' ,'marital_status', 'occupation', 'relationship', 'race', 'sex', 'capital_gain','capital_loss', '
data=pd.read_csv(r"C:\Users\zeusm\Downloads\adult.csv",names=a)
data
```

Out[1]:

	Age	Workclass	Fnlwgt	Education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week	native_country	income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	United-States	<=50K
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	United-States	>50K
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	United-States	<=50K
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	White	Female	15024	0	40	United-States	>50K

32561 rows × 15 columns

In [2]:
```python
data.isnull().sum() #Checking if there are any null values in data set
```

Out[2]:
```
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
```

Data Cleaning :

In [3]:
```python
# Checking if there are any unwanted values in data set

(data['Age']==' ?').sum()
(data['Workclass']==' ?').sum()
(data['Education']==' ?').sum()
(data['education_num']==' ?').sum()
(data['marital_status']==' ?').sum()
(data['occupation']==' ?').sum()
(data['relationship']==' ?').sum()
(data['race']==' ?').sum()
(data['sex']==' ?').sum()
(data['capital_gain']==' ?').sum()
(data['capital_loss']==' ?').sum()
(data['hours_per_week']==' ?').sum()
(data['native_country']==' ?').sum()
(data['income']==' ?').sum()
```

Out[3]:
```
0
```

Now checking count of values in each row to replace unwanted value with this :

In [4]:
```python
data['Workclass'].value_counts()
```

Out[4]:
```
 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 [5]:
```python
data['occupation'].value_counts()
```

Out[5]:
```
 Prof-specialty       4140
 Craft-repair         4099
 Exec-managerial      4066
 Adm-clerical         3770
 Sales                3650
 Other-service        3295
 Machine-op-inspct    2002
 ?                    1843
 Transport-moving     1597
 Handlers-cleaners    1370
 Farming-fishing       994
 Tech-support          928
 Protective-serv       649
 Priv-house-serv       149
 Armed-Forces            9
Name: occupation, dtype: int64
```

In [6]:
```python
data['native_country'].value_counts()
```

Out[6]:
```
 United-States                 29170
 Mexico                          643
 ?                               583
 Philippines                     198
 Germany                         137
 Canada                          121
 Puerto-Rico                     114
 El-Salvador                     106
 India                           100
 Cuba                             95
 England                          90
 Jamaica                          81
 South                            80
 China                            75
 Italy                            73
 Dominican-Republic               70
 Vietnam                          67
 Guatemala                        64
 Japan                            62
 Poland                           60
 Columbia                         59
 Taiwan                           51
 Haiti                            44
 Iran                             43
 Portugal                         37
 Nicaragua                        34
 Peru                             31
 France                           29
 Greece                           29
 Ecuador                          28
 Ireland                          24
 Hong                             20
 Trinadad&Tobago                  19
 Cambodia                         19
 Thailand                         18
 Laos                             18
 Yugoslavia                       16
 Outlying-US(Guam-USVI-etc)       14
 Honduras                         13
 Hungary                          13
 Scotland                         12
 Holand-Netherlands                1
Name: native_country, dtype: int64
```

In [7]:
```python
#Replacing the unwanted values
data['Workclass']=data['Workclass'].replace(to_replace=' ?',value='Private')
data['occupation']=data['occupation'].replace(to_replace=' ?',value='Prof-specialty')
data['occupation']=data['occupation'].replace(to_replace='  ?',value='Prof-specialty')
data['native_country']=data['native_country'].replace(to_replace='  ?',value='United-States')
```

In [8]:
```python
data.columns
```

Out[8]:
```
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')
```

As all the values should be in integer datatype , string values has to be converted into integer values by using LabelEncoder

In [9]:
```python
from sklearn.preprocessing import LabelEncoder
l1=LabelEncoder()
data['Workclass']=l1.fit_transform(data['Workclass'])
data['Education']=l1.fit_transform(data['Education'])
data['marital_status']=l1.fit_transform(data['marital_status'])
data['occupation']=l1.fit_transform(data['occupation'])
data['relationship']=l1.fit_transform(data['relationship'])
data['race']=l1.fit_transform(data['race'])
data['sex']=l1.fit_transform(data['sex'])
data['race']=l1.fit_transform(data['race'])
data['native_country']=l1.fit_transform(data['native_country'])
```

In [10]:
```python
data.shape
```

Out[10]:
```
(32561, 15)
```

Spliting dataset into train and test data :

In [11]:
```python
x=data.iloc[:,:14].values
y=data.iloc[:,14].values
x.shape
```

Out[11]:
```
(32561, 14)
```

In [12]:
```python
y.shape
```

Out[12]:
```
(32561,)
```

Data Preprocessing :

In [13]:
```python
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.20,random_state=1)
```

In [14]:
```python
xtrain.shape
```

Out[14]:
```
(26048, 14)
```

In [15]:
```python
ytrain.shape
```

Out[15]:
```
(26048,)
```

Model Builiding :

In [16]:
```python
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.metrics import accuracy_score
```

In [17]:
```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

In [18]:
```python
#creating function for all models
def apply_model(xtrain,xtest,ytrain,ytest,model):
    model.fit(xtrain,ytrain)
    ypred=model.predict(xtest)
    print("prediction of income for test data :", ypred)
    print("Training score:",model.score(xtrain,ytrain))
    print("Testing score:",model.score(xtest,ytest))
    print('Accuracy score:',accuracy_score(ytest,ypred)*100)
    cm=confusion_matrix(ytest,ypred)
    print("confusion matrix: \n",cm)
    print("classification report : \n ",classification_report(ytest,ypred))
    pr=cm[0][0]/(cm[0][0]+cm[1][0])
    re=cm[0][0]/(cm[0][0]+cm[0][1])
    print("precision:",pr)
    print("recall:",re)
    print("F-1 score: ",2*pr*re/(pr+re))
    acc=(cm[0][0]+cm[1][1])/(cm[0][0]+cm[1][0]+cm[0][1]+cm[1][1])
    print("Accuracy: ",acc)
    print(" ")
    miss=(cm[1][0]+cm[0][1]/(cm[0][0]+cm[1][0]+cm[0][1]+cm[1][1]))
    print("percentage of missclassification :\n",miss)
```

Model 1 : Decision Tree Classifier

In [19]:
```python
Model1=DecisionTreeClassifier(criterion='gini')
apply_model(xtrain,xtest,ytrain,ytest,Model1)
```

```
prediction of income for test data : [' <=50K' ' <=50K' ' >50K' ... ' <=50K' ' <=50K' ' >50K']
Training score: 0.9999616093366094
Testing score: 0.8129894058037771
Accuracy score: 81.2989405803777
confusion matrix:
 [[4365  661]
 [ 557  930]]
classification report :
                precision    recall  f1-score   support

       <=50K       0.89      0.87      0.88      5026
        >50K       0.58      0.63      0.60      1487

    accuracy                           0.81      6513
   macro avg       0.74      0.75      0.74      6513
weighted avg       0.82      0.81      0.82      6513

precision: 0.886834620073141
recall: 0.868483883804218
F-1 score:  0.8775633293124246
Accuracy:  0.8129894058037771

percentage of missclassification :
 557.1014893290343
```

Model 2 : Random Forest Classifer

In [20]:
```python
Model2=RandomForestClassifier()
apply_model(xtrain,xtest,ytrain,ytest,Model2)
```

```
prediction of income for test data : [' <=50K' ' <=50K' ' >50K' ... ' <=50K' ' <=50K' ' >50K']
Training score: 0.9999616093366094
Testing score: 0.8604329801934593
Accuracy score: 86.04329801934593
confusion matrix:
 [[4647  379]
 [ 530  957]]
classification report :
                precision    recall  f1-score   support

       <=50K       0.90      0.92      0.91      5026
        >50K       0.72      0.64      0.68      1487

    accuracy                           0.86      6513
   macro avg       0.81      0.78      0.79      6513
weighted avg       0.86      0.86      0.86      6513

precision: 0.8976241066254588
recall: 0.924592120970951
F-1 score:  0.9109085563069685
Accuracy:  0.8604329801934593

percentage of missclassification :
 530.0581913096884
```

Model 3 : Logistic Regression

In [21]:
```python
Model3=LogisticRegression(solver='liblinear')
apply_model(xtrain,xtest,ytrain,ytest,Model3)
```

```
prediction of income for test data : [' <=50K' ' <=50K' ' >50K' ... ' <=50K' ' <=50K' ' <=50K']
Training score: 0.7916538697788698
Testing score: 0.8020881314294488
Accuracy score: 80.20881314294488
confusion matrix:
 [[4792  234]
 [1055  432]]
classification report :
                precision    recall  f1-score   support

       <=50K       0.82      0.95      0.88      5026
        >50K       0.65      0.29      0.40      1487

    accuracy                           0.80      6513
   macro avg       0.73      0.62      0.64      6513
weighted avg       0.78      0.80      0.77      6513

precision: 0.8195655891910382
recall: 0.9534421010744131
F-1 score:  0.8814494619700175
Accuracy:  0.8020881314294488

percentage of missclassification :
 1055.0359281437127
```

Model 4 : KNN Classifier

In [22]:
```python
Model4=KNeighborsClassifier(n_neighbors=3)
apply_model(xtrain,xtest,ytrain,ytest,Model4)
```

```
prediction of income for test data : [' <=50K' ' <=50K' ' >50K' ... ' <=50K' ' <=50K' ' >50K']
Training score: 0.8624462530712531
Testing score: 0.7729157070474436
Accuracy score: 77.29157070474436
confusion matrix:
 [[4440  586]
 [ 893  594]]
classification report :
                precision    recall  f1-score   support

       <=50K       0.83      0.88      0.86      5026
        >50K       0.50      0.40      0.45      1487

    accuracy                           0.77      6513
   macro avg       0.67      0.64      0.65      6513
weighted avg       0.76      0.77      0.76      6513

precision: 0.8325520345021564
recall: 0.8834062873060088
F-1 score:  0.8572256009267305
Accuracy:  0.7729157070474436

percentage of missclassification :
 893.0899738983571
```

Model 5 : SVC Classifier (with Linear Kernel)

In [25]:
```python
Model5=SVC(kernel='linear',C=1)
apply_model(xtrain,xtest,ytrain,ytest,Model5)
```

```
prediction of income for test data : [' <=50K' ' <=50K' ' >50K' ... ' <=50K' ' <=50K' ' <=50K']
Training score: 0.7929207616707616
Testing score: 0.8053124520190389
Accuracy score: 80.53124520190389
confusion matrix:
 [[5018    8]
 [1260  227]]
classification report :
                precision    recall  f1-score   support

       <=50K       0.80      1.00      0.89      5026
        >50K       0.97      0.15      0.26      1487

    accuracy                           0.81      6513
   macro avg       0.88      0.58      0.58      6513
weighted avg       0.84      0.81      0.75      6513

precision: 0.7992991398534565
recall: 0.9984082769598089
F-1 score:  0.8878273177636234
Accuracy:  0.8053124520190389

percentage of missclassification :
 1260.0012283126055
```

Conclusion:

Random Forest is the best model in terms of accuracy