In [1]:

```
import numpy as np
import pandas as pd
```

In [5]:

```
dataset = pd.read_csv("C:/Users/Sampath/Downloads/adult.csv")
dataset.head()
dataset.head()
```

Out[5]:

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

In [6]:

```
colnames= ['Age', 'Workclass', 'Fnlwgt', 'Education','education_num','marital_status','occu
dataset = pd.read_csv("C:/Users/Sampath/Downloads/adult.csv",names=colnames, header=None)
dataset.head()
```

Out[6]:

	Age	Workclass	Fnlwgt	Education	education_num	marital_status	occupation	relationship
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife

In [7]:

```
print("Original features:\n", list(dataset.columns), "\n")
data_dummies = pd.get_dummies(dataset)
print("Features after get_dummies:\n", list(data_dummies.columns))
data_dummies.head()
```

```
Original features:
 ['Age', 'Workclass', 'Fnlwgt', 'Education', 'education_num', 'marital_statu
s', 'occupation', 'relationship', 'race', 'sex', 'capital_gain', 'capital_lo
ss', 'hours_per_week', 'native_country', 'income']

Features after get_dummies:
 ['Age', 'Fnlwgt', 'education_num', 'capital_gain', 'capital_loss', 'hours_p
er_week', 'Workclass_ ?', 'Workclass_ Federal-gov', 'Workclass_ Local-gov',
'Workclass_ Never-worked', 'Workclass_ Private', 'Workclass_ Self-emp-inc',
'Workclass_ Self-emp-not-inc', 'Workclass_ State-gov', 'Workclass_ Without-p
ay', 'Education_ 10th', 'Education_ 11th', 'Education_ 12th', 'Education_ 1s
t-4th', 'Education_ 5th-6th', 'Education_ 7th-8th', 'Education_ 9th', 'Educa
tion_ Assoc-acdm', 'Education_ Assoc-voc', 'Education_ Bachelors', 'Educatio
n_ Doctorate', 'Education_ HS-grad', 'Education_ Masters', 'Education_ Presc
hool', 'Education_ Prof-school', 'Education_ Some-college', 'marital_status_
Divorced', 'marital_status_ Married-AF-spouse', 'marital_status_ Married-civ
-spouse', 'marital_status_ Married-spouse-absent', 'marital_status_ Never-ma
rried', 'marital_status_ Separated', 'marital_status_ Widowed', 'occupation_
?', 'occupation_ Adm-clerical', 'occupation_ Armed-Forces', 'occupation_ Cra
ft-repair', 'occupation_ Exec-managerial', 'occupation_ Farming-fishing', 'o
ccupation_ Handlers-cleaners', 'occupation_ Machine-op-inspct', 'occupation_
Other-service', 'occupation_ Priv-house-serv', 'occupation_ Prof-specialty',
'occupation_ Protective-serv', 'occupation_ Sales', 'occupation_ Tech-suppor
t', 'occupation_ Transport-moving', 'relationship_ Husband', 'relationship_
Not-in-family', 'relationship_ Other-relative', 'relationship_ Own-child',
'relationship_ Unmarried', 'relationship_ Wife', 'race_ Amer-Indian-Eskimo',
'race_ Asian-Pac-Islander', 'race_ Black', 'race_ Other', 'race_ White', 'se
x_ Female', 'sex_ Male', 'native_country_ ?', 'native_country_ Cambodia', 'n
ative_country_ Canada', 'native_country_ China', 'native_country_ Columbia',
'native_country_ Cuba', 'native_country_ Dominican-Republic', 'native_countr
y_ Ecuador', 'native_country_ El-Salvador', 'native_country_ England', 'nati
ve_country_ France', 'native_country_ Germany', 'native_country_ Greece', 'n
ative_country_ Guatemala', 'native_country_ Haiti', 'native_country_ Holand-
Netherlands', 'native_country_ Honduras', 'native_country_ Hong', 'native_co
untry_ Hungary', 'native_country_ India', 'native_country_ Iran', 'native_co
untry_ Ireland', 'native_country_ Italy', 'native_country_ Jamaica', 'native
_country_ Japan', 'native_country_ Laos', 'native_country_ Mexico', 'native_
country_ Nicaragua', 'native_country_ Outlying-US(Guam-USVI-etc)', 'native_c
ountry_ Peru', 'native_country_ Philippines', 'native_country_ Poland', 'nat
ive_country_ Portugal', 'native_country_ Puerto-Rico', 'native_country_ Scot
land', 'native_country_ South', 'native_country_ Taiwan', 'native_country_ T
hailand', 'native_country_ Trinadad&Tobago', 'native_country_ United-State
s', 'native_country_ Vietnam', 'native_country_ Yugoslavia', 'income_ <=50
K', 'income_ >50K']
```

Out[7]:

	Age	Fnlwgt	education_num	capital_gain	capital_loss	hours_per_week	Workclass_ ?	W
0	39	77516	13	2174	0	40	0	
1	50	83311	13	0	0	13	0	

	Age	Fnlwgt	education_num	capital_gain	capital_loss	hours_per_week	Workclass_?	W
2	38	215646	9	0	0	40	0	
3	53	234721	7	0	0	40	0	
4	28	338409	13	0	0	40	0	

5 rows × 110 columns

In [8]:

```
dataset.info('income')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Age             32561 non-null  int64
 1   Workclass       32561 non-null  object
 2   Fnlwgt          32561 non-null  int64
 3   Education       32561 non-null  object
 4   education_num   32561 non-null  int64
 5   marital_status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital_gain    32561 non-null  int64
 11  capital_loss    32561 non-null  int64
 12  hours_per_week  32561 non-null  int64
 13  native_country  32561 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

In [9]:

```
X = data_dummies.iloc[:, :-2].values
y = data_dummies.iloc[:, -1].values
data_dummies.head()
```

Out[9]:

	Age	Fnlwgt	education_num	capital_gain	capital_loss	hours_per_week	Workclass_?	Workc Fe
0	39	77516	13	2174	0	40	0	
1	50	83311	13	0	0	13	0	
2	38	215646	9	0	0	40	0	
3	53	234721	7	0	0	40	0	
4	28	338409	13	0	0	40	0	

5 rows × 110 columns

In [11]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state =
```

In [12]:

```
print(y_train)
```

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

In [13]:

```
print(y_train)
```

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

In [14]:

```
print(X_test)
```

```
[[    27 177119     10 ...      1      0      0]
 [    27 216481     13 ...      1      0      0]
 [    25 256263     12 ...      1      0      0]
 ...
 [    38 229236      9 ...      0      0      0]
 [    63 213095     13 ...      1      0      0]
 [    47 186009     10 ...      0      0      0]]
```

In [17]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [18]:

```
print(X_train)
```

```
[[ 1.49894077 -1.2079589  -0.80338299 ...  0.33936755 -0.04296689
  -0.02307887]
 [ 2.38045121 -0.08366135  1.53114709 ...  0.33936755 -0.04296689
  -0.02307887]
 [ 0.2501343  -0.77396423  1.53114709 ...  0.33936755 -0.04296689
  -0.02307887]
 ...
 [-1.14559057 -1.36469155 -0.0252063  ...  0.33936755 -0.04296689
  -0.02307887]
 [ 0.47051192  0.24865765  2.30932378 ...  0.33936755 -0.04296689
  -0.02307887]
 [-0.99867216 -0.02508527 -0.0252063  ...  0.33936755 -0.04296689
  -0.02307887]]
```

In [19]:

```
print(X_test)
```

```
[[-0.85175375 -0.11784965 -0.0252063  ...  0.33936755 -0.04296689
  -0.02307887]
 [-0.85175375  0.25451337  1.14205874 ...  0.33936755 -0.04296689
  -0.02307887]
 [-0.99867216  0.63084957  0.7529704  ...  0.33936755 -0.04296689
  -0.02307887]
 ...
 [-0.04370251  0.37517518 -0.41429464 ... -2.94665767 -0.04296689
  -0.02307887]
 [ 1.79277758  0.22248194  1.14205874 ...  0.33936755 -0.04296689
  -0.02307887]
 [ 0.61743032 -0.03375059 -0.0252063  ... -2.94665767 -0.04296689
  -0.02307887]]
```

DECISION TREE

In [20]:

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

Out[20]:

```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In [21]:

```
print(classifier.predict(sc.transform(X_train)))
```

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

In [22]:

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

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

In [23]:

```
from sklearn.metrics import confusion_matrix, accuracy_score,classification_report
cm = confusion_matrix(y_test, y_pred)
print(cm)
DT_accu=accuracy_score(y_test, y_pred)
print(classification_report(y_test, y_pred))
```

```
[[5412  747]
 [ 756 1226]]
              precision    recall  f1-score   support

           0       0.88      0.88      0.88      6159
           1       0.62      0.62      0.62      1982

    accuracy                           0.82      8141
   macro avg       0.75      0.75      0.75      8141
weighted avg       0.82      0.82      0.82      8141
```

RANDOM FOREST CLASSIFIER

In [25]:

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state
classifier.fit(X_train, y_train)
```

Out[25]:

```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

In [26]:

```
print(classifier.predict(sc.transform(X_train)))
```

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

In [27]:

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

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

In [28]:

```
from sklearn.metrics import confusion_matrix, accuracy_score,classification_report
cm = confusion_matrix(y_test, y_pred)
print(cm)
RF_accu=accuracy_score(y_test, y_pred)
print(classification_report(y_test, y_pred))
```

```
[[5741  418]
 [ 852 1130]]
              precision    recall  f1-score   support

           0       0.87      0.93      0.90      6159
           1       0.73      0.57      0.64      1982

    accuracy                           0.84      8141
   macro avg       0.80      0.75      0.77      8141
weighted avg       0.84      0.84      0.84      8141
```

LOGISTIC REGRESSION

In [29]:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

Out[29]:

```
LogisticRegression(random_state=0)
```

In [30]:

```
print(classifier.predict(sc.transform(X_train)))
```

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

In [31]:

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

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

In [32]:

```
from sklearn.metrics import confusion_matrix, accuracy_score,classification_report
cm = confusion_matrix(y_test, y_pred)
print(cm)
LR_accu=accuracy_score(y_test, y_pred)
print(classification_report(y_test, y_pred))
```

```
[[5719  440]
 [ 779 1203]]
              precision    recall  f1-score   support

           0       0.88      0.93      0.90      6159
           1       0.73      0.61      0.66      1982

    accuracy                           0.85      8141
   macro avg       0.81      0.77      0.78      8141
weighted avg       0.84      0.85      0.85      8141
```

KNN CLASSIFIER

In [33]:

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

Out[33]:

```
KNeighborsClassifier()
```

In [36]:

```
print(classifier.predict(sc.transform(X_train)))
```

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

In [38]:

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

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

In [39]:

```
from sklearn.metrics import confusion_matrix, accuracy_score,classification_report
cm = confusion_matrix(y_test, y_pred)
print(cm)
KNN_accu = accuracy_score(y_test, y_pred)
print(classification_report(y_test, y_pred))
```

```
[[5562  597]
 [ 854 1128]]
              precision    recall  f1-score   support

           0       0.87      0.90      0.88      6159
           1       0.65      0.57      0.61      1982

    accuracy                           0.82      8141
   macro avg       0.76      0.74      0.75      8141
weighted avg       0.82      0.82      0.82      8141
```

SVC CLASSIFIER(with linear kernel)

In [40]:

```
# Importing SVM from sklearn
from sklearn import svm
```

In [41]:

```
# Creating the SVM model and training the model with X_train and y_train
SVM_m = svm.SVC()
SVM_m.fit(X_train, y_train)
```

Out[41]:

```
SVC()
```

In [42]:

```
# Using the X_test to predict the values
y_pred_SVM = SVM_m.predict(X_test)
```

In [43]:

```
# Calculating the Confusion Matrix and Accuracy_Score of the model
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
cm = confusion_matrix(y_test, y_pred_SVM)
print("Confusion Matrix = ", cm)
print("*"*80)
# Classification Report
print(classification_report(y_test, y_pred_SVM))
print("*"*80)
SVM_accu = accuracy_score(y_test, y_pred_SVM)
print("SVM Classifier Accuracy = ", SVM_accu)
```

```
Confusion Matrix =  [[5789  370]
 [ 867 1115]]
********************************************************************************
****
              precision    recall  f1-score   support

           0       0.87      0.94      0.90      6159
           1       0.75      0.56      0.64      1982

    accuracy                           0.85      8141
   macro avg       0.81      0.75      0.77      8141
weighted avg       0.84      0.85      0.84      8141

********************************************************************************
****
SVM Classifier Accuracy =  0.8480530647340622
```

In [44]:

```
print("Decision Tree Classifier Accuracy = ", DT_accu)
print("Random Forest Classifier Accuracy = ", RF_accu)
print("KNN Classifier Accuracy = ", KNN_accu)
print("Logistic Regression Accuracy = ", LR_accu)
print("SVM Classifier Accuracy = ", SVM_accu)
```

```
Decision Tree Classifier Accuracy =  0.8153789460754207
Random Forest Classifier Accuracy =  0.8439995086598698
KNN Classifier Accuracy =  0.8217663677680874
Logistic Regression Accuracy =  0.8502640953199853
SVM Classifier Accuracy =  0.8480530647340622
```

Logistic Regression has the best accuracy

In [ ]:

```

```