```
## Doing necessary imports
import numpy as np
import pandas as pd
from imblearn.over_sampling import RandomOverSampler
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_style("whitegrid")

## to display all columns of the data set
pd.pandas.set_option('display.max_columns',None)
```

```
## reading the data file

df=pd.read_csv("/content/drive/MyDrive/incomeData.csv")
```

```
df.head()
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	ra
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	Wl
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	Wl
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	Wl
3	53	Private	234721	11th	7	Married-civ-	Handlers-	Husband	Bl

```
df.tail()
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct
32558	58	Private	151910	HS-grad	9	Widowed	Adm-

```
df.shape
```

```
(32561, 15)
```

```
df.columns
```

```
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')
```

```
## making sure that data set doesn't contain unnecessay space
df= df.apply(lambda x: x.str.strip() if x.dtype =="object" else x )  # for object only
```

```
df.isna().sum()
```

```
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
```

```
df[df['workclass'] == '?']
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupatior
27	54	?	180211	Some-college	10	Married-civ-spouse	‘
61	32	?	293936	7th-8th	4	Married-spouse-absent	‘
69	25	?	200681	Some-college	10	Never-married	‘
77	67	?	212759	10th	6	Married-civ-spouse	‘
106	17	?	304873	10th	6	Never-married	‘
...	...	...	...	...	...	...	..
32530	35	?	320084	Bachelors	13	Married-civ-spouse	‘
32531	30	?	33811	Bachelors	13	Never-	‘

```
## In this dataset missing values have been denoted by '?'
## we are replacing ? with NaN for them to be imputed down the line.
df.replace('?',np.NaN,inplace=True)
```

```
## Here we will check the percentage of nan values present in each feature

features_with_na = [features for features in df.columns if df[features].isnull().sum()>1 ]

## printing the feature name and the percentage of missing values

for feature in features_with_na:
   print(feature, np.round(df[feature].isnull().mean(), 4),   " %   missing values")
```

```
workclass 0.0564  %   missing values
occupation 0.0566  %   missing values
native-country 0.0179  %   missing values
```

```
df.info()
```

```
<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       30725 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      30718 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  31978 non-null  object
 14  Income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

```
## As the columns which have missing values, they are only categorical, we'll use the categorical imputer
# Importing the categorical imputer
from sklearn_pandas import CategoricalImputer  # mode
imputer = CategoricalImputer()
```

```
## imputing the missing values from the column

df['workclass']=imputer.fit_transform(df['workclass'])
df['occupation']=imputer.fit_transform(df['occupation'])
df['native-country']=imputer.fit_transform(df['native-country'])
```

```
df.isna().sum()
```

```
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
```

```
df.head()
```

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	re
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	

```
df[['education','education-num']].head(10)
```

	education	education-num

```
## The education column has a corresponding education-num column which has numerical values
df.drop(columns=['education'] ,inplace=True) # 1 education is needed
```

2	HS-grad	9

```
## Extracting the categorical columns
cat_df = df.select_dtypes(include=['object']).copy()
```

4	Bachelors	13

```
cat_df.columns
```

```
Index(['workclass', 'marital-status', 'occupation', 'relationship', 'race',
       'sex', 'native-country', 'Income'],
      dtype='object')
```

```
cat_df.head()
```

	workclass	marital-status	occupation	relationship	race	sex	native-country
0	State-gov	Never-married	Adm-clerical	Not-in-family	White	Male	United-States
1	Self-emp-not-inc	Married-civ-spouse	Exec-managerial	Husband	White	Male	United-States

```
cat_df['Income'].unique()
```

```
array(['<=50K', '>50K'], dtype=object)
```

```
cat_df['Income'] = cat_df['Income'].map({'<=50K' : 0, '>50K' : 1}) # encoding target column into 2 things
```

```
cat_df.head()
```

	workclass	marital-status	occupation	relationship	race	sex	native-country
0	State-gov	Never-married	Adm-clerical	Not-in-family	White	Male	United-States
1	Self-emp-not-inc	Married-civ-spouse	Exec-managerial	Husband	White	Male	United-States

```
## Using the dummy encoding to encode the categorical columns to numericsl ones
for col in cat_df.drop('Income',axis=1).columns:
    x=cat_df[col].head(1)
    cat_df= pd.get_dummies(cat_df, columns=[col], prefix = [col], drop_first=True)
```

```
cat_df.head()
```

	Income	workclass_Local-gov	workclass_Never-worked	workclass_Private	workclas
0	0	0	0	0	
1	0	0	0	0	
2	0	0	0	1	
3	0	0	0	1	
4	0	0	0	1	

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 14 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-num   32561 non-null  int64
 4   marital-status  32561 non-null  object
 5   occupation      32561 non-null  object
 6   relationship    32561 non-null  object
 7   race            32561 non-null  object
 8   sex             32561 non-null  object
 9   capital-gain    32561 non-null  int64
 10  capital-loss    32561 non-null  int64
 11  hours-per-week  32561 non-null  int64
 12  native-country  32561 non-null  object
 13  Income          32561 non-null  object
dtypes: int64(6), object(8)
memory usage: 3.5+ MB
```

```
## extracting the numerical columns
num_df = df.select_dtypes(include=['int64']).copy()
```

```
num_df.head()
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
0	39	77516	13	2174	0	40
1	50	83311	13	0	0	13
2	38	215646	9	0	0	40
3	53	234721	7	0	0	40

```
## Noramlizing the numerical columns
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaled_data=scaler.fit_transform(num_df)
scaled_data
```

```
array([[ 0.03067056, -1.06361075,  1.13473876,  0.1484529 , -0.21665953,
        -0.03542945],
       [ 0.83710898, -1.008707  ,  1.13473876, -0.14592048, -0.21665953,
        -2.22215312],
       [-0.04264203,  0.2450785 , -0.42005962, -0.14592048, -0.21665953,
        -0.03542945],
       ...,
       [ 1.42360965, -0.35877741, -0.42005962, -0.14592048, -0.21665953,
        -0.03542945],
       [-1.21564337,  0.11095988, -0.42005962, -0.14592048, -0.21665953,
        -1.65522476],
       [ 0.98373415,  0.92989258, -0.42005962,  1.88842434, -0.21665953,
        -0.03542945]])
```

```
scaled_num_df= pd.DataFrame(data=scaled_data, columns=num_df.columns)
scaled_num_df.head()
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week
0	0.030671	-1.063611	1.134739	0.148453	-0.21666	-0.035429
1	0.837109	-1.008707	1.134739	-0.145920	-0.21666	-2.222153
2	-0.042642	0.245079	-0.420060	-0.145920	-0.21666	-0.035429
3	1.057047	0.425801	-1.197459	-0.145920	-0.21666	-0.035429

```
## combining the Numerical and categorical dataframes to get the final dataset
final_df=pd.concat([scaled_num_df,cat_df], axis=1)
final_df.head()
```

	age	fnlwgt	education-num	capital-gain	capital-loss	hours-per-week	Income
0	0.030671	-1.063611	1.134739	0.148453	-0.21666	-0.035429	0
1	0.837109	-1.008707	1.134739	-0.145920	-0.21666	-2.222153	0
2	-0.042642	0.245079	-0.420060	-0.145920	-0.21666	-0.035429	0
3	1.057047	0.425801	-1.197459	-0.145920	-0.21666	-0.035429	0
4	-0.775768	1.408176	1.134739	-0.145920	-0.21666	-0.035429	0

```
df.shape
```

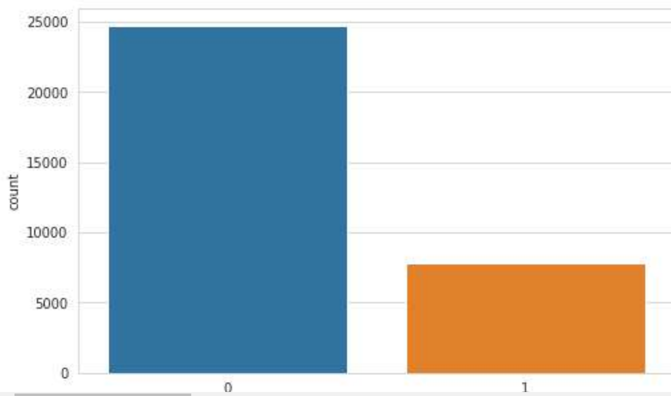
```
(32561, 14)
```

```
## separating the feature and target columns
x=final_df.drop('Income',axis=1)
y=final_df['Income']
```

If we plot the distribution of the target feature, we'd find that the peole with less than 50K annual income are more in number than the people with an annual income greaterthan 50K

```
plt.figure(figsize=(8,5))
sns.countplot(y);
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWa
  warnings.warn(
```

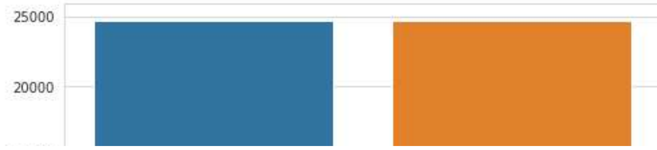


Hence, the dataset is imbalanced. we need to introduce some random sampling to make it balanced.

```
rdsmple = RandomOverSampler()
x_sampled,y_sampled  = rdsmple.fit_resample(x,y)
```

```
## again plotting the target feature
plt.figure(figsize=(8,5))
sns.countplot(y_sampled);
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWa
  warnings.warn(
```



As shown above, now the data looks to be balanced

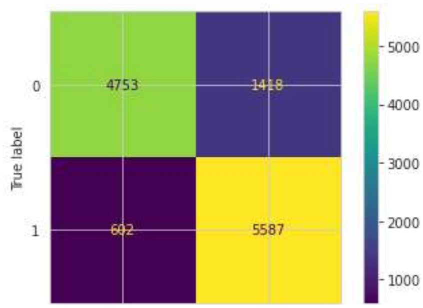
```
# splitting the data into training and test set
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x_sampled,y_sampled, random_state=42 )
```

```
## Model creating
from sklearn.neighbors import KNeighborsClassifier
model=KNeighborsClassifier(n_neighbors=5)
model.fit(x_train,y_train)
y_pred=model.predict(x_test)
y_pred
```

```
array([0, 0, 0, ..., 1, 0, 0])
```

```
from sklearn.metrics import classification_report,accuracy_score,confusion_matrix,ConfusionMatrixDisplay
cm=confusion_matrix(y_test,y_pred)
label= [0,1]
cmd = ConfusionMatrixDisplay(cm, display_labels=label)
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f5a35b8ee50>
```




```
## accuracy of the model is,
accuracy = accuracy_score(y_test,y_pred)
accuracy
```

```
0.8365695792880259
```

```
## generating classification report
report=classification_report(y_test,y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.89      0.77      0.82      6171
           1       0.80      0.90      0.85      6189

    accuracy                           0.84     12360
   macro avg       0.84      0.84      0.84     12360
weighted avg       0.84      0.84      0.84     12360
```

```
## applying GridSearchCV to find best parameter to improve the accuracy
from sklearn.model_selection import GridSearchCV

cls=KNeighborsClassifier()

params={'n_neighbors':[3,5,7,8],'weights':['uniform','distance']}
```

```
clf=GridSearchCV(cls,params,cv=10,scoring='accuracy')  # cv ===> cross validation
clf.fit(x_train,y_train)
```

```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [3, 5, 7, 8],
                         'weights': ['uniform', 'distance']},
             scoring='accuracy')
```

```
print(clf.best_params_)
```

```
{'n_neighbors': 8, 'weights': 'distance'}
```