ADULT INCOME CENSUS PREDICTION

PROJECT NAME-ADULT INCOME CENSUS PREDICTION
NAME-ARUN SHANKAR OVHAL
IT VEDANT THANE

Description:

- In this notebook, we are going to predict whether a person's income is above 50k or below 50k using various features like age, education, and occupation.
- The dataset we are going to use is the Adult census income dataset from Kaggle which contains about 32,561 rows and 13 features.
- The dataset contains the labels which we have to predict and the labels are discrete and binary. So the problem we have is a Supervised Classification type.

Motivation:

Building such predictive models can help us better understand the population of a country as well as the various factors affecting the economy. Governments can understand such factors and improve upon them leading to the growth of the country.

Features Description:

Column ID	Column Name	Description
1	Age	Age of person
2	Workclass	Work class of person
3	Fnlwgt	Final weight
4	Education	Education Degree of person
5	Education.num	Number of years of education
6	Marital.status	Marital status of person
7	Occupation	Occupation of person
8	Sex	Sex of person
9	Capital.gain	Capital gain of person
10	Capital.loss	Capital loss of person
11	Hours.per.week	Number of hours per week
12	Native.country	Native country of person
13	Income	Income category of person

Step 0: Load libraries and dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
import warnings
warnings.filterwarnings("ignore")
```

Step 1:Importing Dataset

```
df=pd.read_csv(r"D:\ARUN DATA\DATA SCIENCE PRACTISE\Machine Learning\Project\adult (1).csv")
```

```
df
```

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	sex	capital.gain	capital.loss	hours.per.week	native.country	income
0	90	?	77053	HS-grad	9	Widowed	?	Female	0	4356	40	United-States	<=50K
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Female	0	4356	18	United-States	<=50K
2	66	?	186061	Some-college	10	Widowed	?	Female	0	4356	40	United-States	<=50K
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Female	0	3900	40	United-States	<=50K
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Female	0	3900	40	United-States	<=50K
...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	22	Private	310152	Some-college	10	Never-married	Protective-serv	Male	0	0	40	United-States	<=50K
32557	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Female	0	0	38	United-States	<=50K
32558	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Male	0	0	40	United-States	>50K
32559	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Female	0	0	40	United-States	<=50K
32560	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Male	0	0	20	United-States	<=50K

32561 rows × 13 columns

```
df.describe()
```

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 13 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   sex             32561 non-null  object
 8   capital.gain    32561 non-null  int64
 9   capital.loss    32561 non-null  int64
 10  hours.per.week  32561 non-null  int64
 11  native.country  32561 non-null  object
 12  income          32561 non-null  object
```

Step 2: Exploratory Data Analysis

```
df["workclass"].value_counts()
```

```
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
```

Observations:

The dataset doesn't have any null values, but it contains missing values in the form of '?' which needs to be preprocessed.

```
df["occupation"].mode()
```

```
0    Prof-specialty
Name: occupation, dtype: object
```

```
NL_mode=df["occupation"].mode()
NL_mode
```

```
0    Prof-specialty
Name: occupation, dtype: object
```

```
df["occupation"].replace("?","Prof-specialty",inplace=True)
df
```

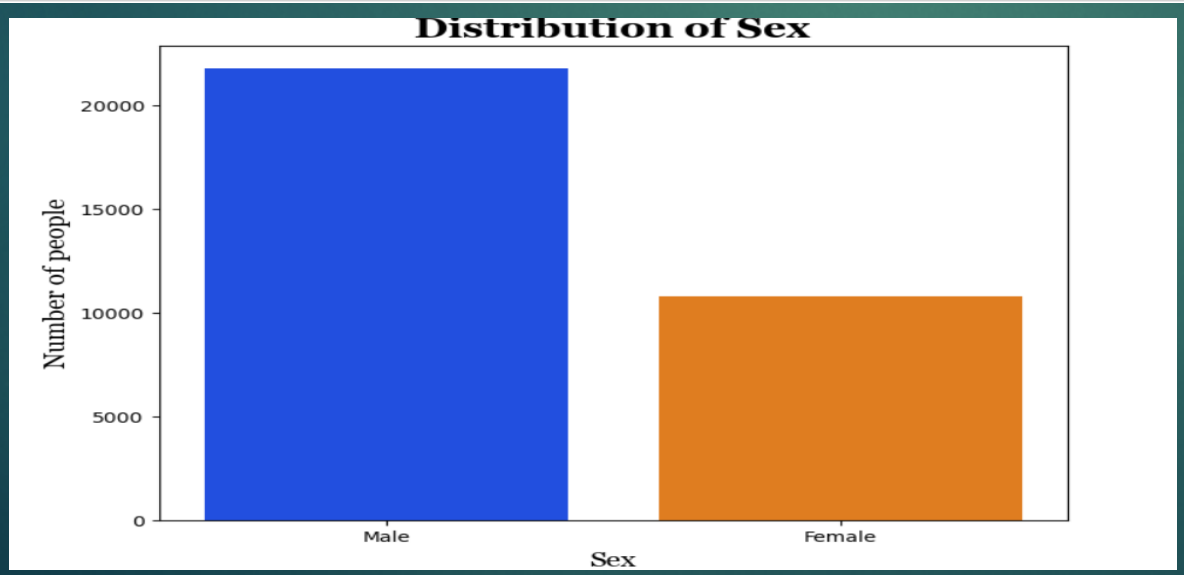
```
df["native.country"].mode()
```

```
0    United-States
Name: native.country, dtype: object
```

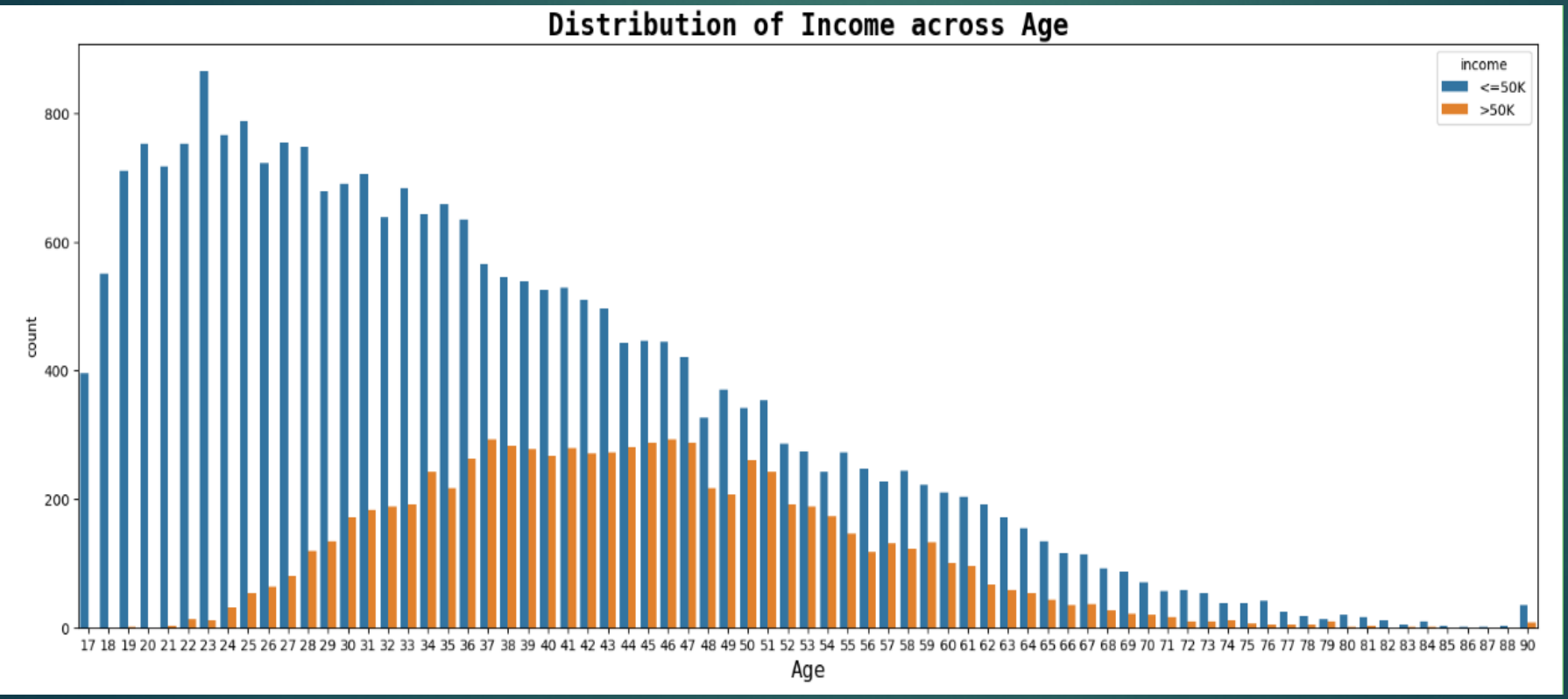
```
df["native.country"].replace("?","United-States",inplace=True)
df
```

Step 2.1:Visualization

```
sex = df['sex'].value_counts()
plt.style.use('default')
plt.figure(figsize=(8, 6))
sns.barplot(x=sex.index, y=sex.values, palette='bright')
plt.title('Distribution of Sex', fontdict={'fontname': 'Georgia', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Sex', fontdict={'fontname': 'Georgia', 'fontsize': 15})
plt.ylabel('Number of people', fontdict={'fontname': 'Georgia', 'fontsize': 15})
plt.tick_params(labelsize=10)
plt.show()
```



```
plt.style.use('default')
plt.figure(figsize=(20, 7))
sns.countplot(x='age', hue='income', data=df)
plt.title('Distribution of Income across Age', fontdict={
          'fontname': 'Monospace', 'fontsize': 20, 'fontweight': 'bold'})
plt.xlabel('Age', fontdict={'fontname': 'Monospace', 'fontsize': 15})
plt.show()
```



Step 2.2: Separting Numeric and category Data

```
df_num=df.select_dtypes(['int','float'])
df_num
```

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week
0	90	77053	9	0	4356	40
1	82	132870	9	0	4356	18
2	66	186061	10	0	4356	40
3	54	140359	4	0	3900	40
4	41	264663	10	0	3900	40
...	...	...	...	...	...	...
32556	22	310152	10	0	0	40
32557	27	257302	12	0	0	38
32558	40	154374	9	0	0	40
32559	58	151910	9	0	0	40
32560	22	201490	9	0	0	20

32561 rows × 6 columns

```
df_cat=df.select_dtypes(['object'])
df_cat
```

	workclass	education	marital.status	occupation	sex	native.country	income
0	Private	HS-grad	Widowed	Prof-specialty	Female	United-States	<=50K
1	Private	HS-grad	Widowed	Exec-managerial	Female	United-States	<=50K
2	Private	Some-college	Widowed	Prof-specialty	Female	United-States	<=50K
3	Private	7th-8th	Divorced	Machine-op-inspct	Female	United-States	<=50K
4	Private	Some-college	Separated	Prof-specialty	Female	United-States	<=50K
...	...	...	...	...	...	...	...
32556	Private	Some-college	Never-married	Protective-serv	Male	United-States	<=50K
32557	Private	Assoc-acdm	Married-civ-spouse	Tech-support	Female	United-States	<=50K
32558	Private	HS-grad	Married-civ-spouse	Machine-op-inspct	Male	United-States	>50K
32559	Private	HS-grad	Widowed	Adm-clerical	Female	United-States	<=50K
32560	Private	HS-grad	Never-married	Adm-clerical	Male	United-States	<=50K

32561 rows × 7 columns

step 2.3: Using Label Encoder for converting category data into numeric

```
from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

for i in df_cat:
    df_cat[i]=le.fit_transform(df_cat[i])
df_cat
```

	workclass	education	marital.status	occupation	sex	native.country	income
0	3	11	6	9	0	38	0
1	3	11	6	3	0	38	0
2	3	15	6	9	0	38	0
3	3	5	0	6	0	38	0
4	3	15	5	9	0	38	0
...	...	...	...	...	...	...	...
32556	3	15	4	10	1	38	0
32557	3	7	2	12	0	38	0
32558	3	11	2	6	1	38	1
32559	3	11	6	0	0	38	0
32560	3	11	4	0	1	38	0

32561 rows × 7 columns

Step 2.4: Features Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
sc=StandardScaler()
```

```
x_scaled=sc.fit_transform(df_num)
x_scaled
```

```
array([[ 3.76961234, -1.06799736, -0.42005962, -0.14592048, 10.59350656,
        -0.03542945],
       [ 3.18311167, -0.53916866, -0.42005962, -0.14592048, 10.59350656,
        -1.81720429],
       [ 2.01011032, -0.03521956, -0.03136003, -0.14592048, 10.59350656,
        -0.03542945],
       ...,
       [ 0.10398314, -0.33543266, -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]])
```

```
df_scaled = pd.DataFrame(data=x_scaled, columns=df_num.columns)
print(df_scaled)
```

```
             age    fnlwgt  education.num  capital.gain  capital.loss  \
0       3.769612 -1.067997       -0.420060       -0.14592     10.593507
1       3.183112 -0.539169       -0.420060       -0.14592     10.593507
2       2.010110 -0.035220       -0.031360       -0.14592     10.593507
3       1.130359 -0.468215       -2.363558       -0.14592      9.461864
4       0.177296  0.709482       -0.031360       -0.14592      9.461864
...          ...       ...             ...            ...           ...
32556 -1.215643  1.140460       -0.031360       -0.14592     -0.216660
32557 -0.849080  0.639741        0.746039       -0.14592     -0.216660
32558  0.103983 -0.335433       -0.420060       -0.14592     -0.216660
32559  1.423610 -0.358777       -0.420060       -0.14592     -0.216660
32560 -1.215643  0.110960       -0.420060       -0.14592     -0.216660

       hours.per.week
0           -0.035429
1           -1.817204
2           -0.035429
3           -0.035429
4           -0.035429
...               ...
32556       -0.035429
32557       -0.197409
32558       -0.035429
32559       -0.035429
32560       -1.655225

[32561 rows x 6 columns]
```

step 2.5: Concatenate Numerical and Categorical Columns

```
df_new=pd.concat([df_scaled,df_cat],axis=1)
df_new
```

	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week	workclass	education	marital.status	occupation	sex	native.country	ir
0	3.769612	-1.067997	-0.420060	-0.14592	10.593507	-0.035429	3	11	6	9	0	38	
1	3.183112	-0.539169	-0.420060	-0.14592	10.593507	-1.817204	3	11	6	3	0	38	
2	2.010110	-0.035220	-0.031360	-0.14592	10.593507	-0.035429	3	15	6	9	0	38	
3	1.130359	-0.468215	-2.363558	-0.14592	9.461864	-0.035429	3	5	0	6	0	38	
4	0.177296	0.709482	-0.031360	-0.14592	9.461864	-0.035429	3	15	5	9	0	38	
...	...	...	...	...	...	...	...	...	...	...	...	...	
32556	-1.215643	1.140460	-0.031360	-0.14592	-0.216660	-0.035429	3	15	4	10	1	38	
32557	-0.849080	0.639741	0.746039	-0.14592	-0.216660	-0.197409	3	7	2	12	0	38	
32558	0.103983	-0.335433	-0.420060	-0.14592	-0.216660	-0.035429	3	11	2	6	1	38	
32559	1.423610	-0.358777	-0.420060	-0.14592	-0.216660	-0.035429	3	11	6	0	0	38	
32560	-1.215643	0.110960	-0.420060	-0.14592	-0.216660	-1.655225	3	11	4	0	1	38	

step 2.6:Splitting Data into X and Y

```
y=df_new.iloc[:,12:13]
y
```

	income
0	0
1	0
2	0
3	0
4	0
...	...
32556	0
32557	0
32558	1
32559	0
32560	0

32561 rows × 1 columns

```
x=df_new.drop('income',axis=1)
x
```

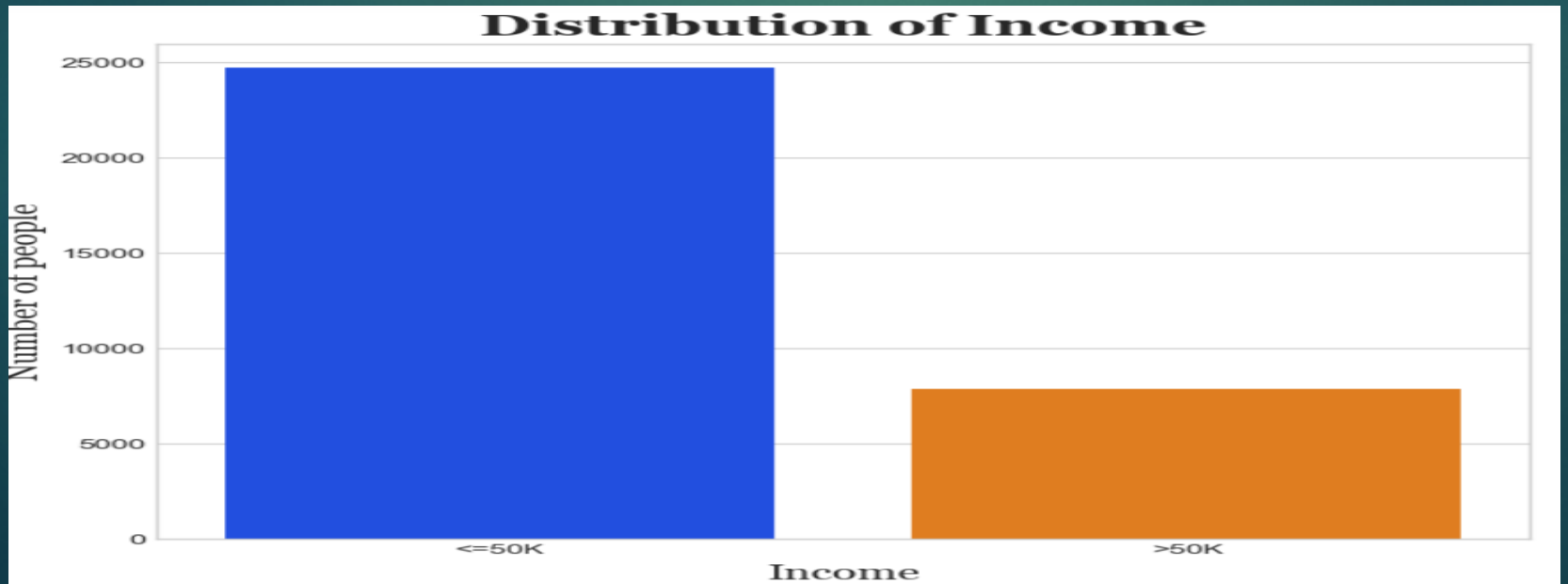
	age	fnlwgt	education.num	capital.gain	capital.loss	hours.per.week	workclass	education	marital.status	occupation	sex	native.country
0	3.769612	-1.067997	-0.420060	-0.14592	10.593507	-0.035429	3	11	6	9	0	38
1	3.183112	-0.539169	-0.420060	-0.14592	10.593507	-1.817204	3	11	6	3	0	38
2	2.010110	-0.035220	-0.031360	-0.14592	10.593507	-0.035429	3	15	6	9	0	38
3	1.130359	-0.468215	-2.363558	-0.14592	9.461864	-0.035429	3	5	0	6	0	38
4	0.177296	0.709482	-0.031360	-0.14592	9.461864	-0.035429	3	15	5	9	0	38
...	...	...	...	...	...	...	...	...	...	...	...	...
32556	-1.215643	1.140460	-0.031360	-0.14592	-0.216660	-0.035429	3	15	4	10	1	38
32557	-0.849080	0.639741	0.746039	-0.14592	-0.216660	-0.197409	3	7	2	12	0	38
32558	0.103983	-0.335433	-0.420060	-0.14592	-0.216660	-0.035429	3	11	2	6	1	38
32559	1.423610	-0.358777	-0.420060	-0.14592	-0.216660	-0.035429	3	11	6	0	0	38
32560	-1.215643	0.110960	-0.420060	-0.14592	-0.216660	-1.655225	3	11	4	0	1	38

32561 rows × 12 columns

Observation:
The dataset is unbalanced, as the dependent feature 'income' contains 75.92% values have income less than 50k and 24.08% values have income more than 50k.

```
income = df['income'].value_counts(normalize=True)
round(income * 100, 2).astype('str') + ' %'
```

```
<=50K    75.92 %
>50K     24.08 %
Name: income, dtype: object
```



*Since the Data is imbalanced , need to do sampling (either Oversampling or Undersampling)

```
import imblearn
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=42)
x_ros, y_ros = ros.fit_resample(x, y)
```

```
x=x_ros
y=y_ros
```

step 3: Model Evaluation

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=42)
```

```
logreg = LogisticRegression()
knn = KNeighborsClassifier()
dt = DecisionTreeClassifier()
rf = RandomForestClassifier(random_state=42)
svm = SVC()
```

```
def mymodel(model):
    model.fit(x_train,y_train)
    y_pred = model.predict(x_test)
    print(classification_report(y_test,y_pred))
    return model
```

mymodel(logreg)

	precision	recall	f1-score	support
0	0.77	0.77	0.77	7393
1	0.77	0.77	0.77	7439
accuracy			0.77	14832
macro avg	0.77	0.77	0.77	14832
weighted avg	0.77	0.77	0.77	14832

LogisticRegression

LogisticRegression()

mymodel(knn)

	precision	recall	f1-score	support
0	0.87	0.77	0.82	7393
1	0.79	0.89	0.84	7439
accuracy			0.83	14832
macro avg	0.83	0.83	0.83	14832
weighted avg	0.83	0.83	0.83	14832

KNeighborsClassifier

KNeighborsClassifier()

mymodel(dt)

	precision	recall	f1-score	support
0	0.95	0.85	0.90	7393
1	0.87	0.96	0.91	7439
accuracy			0.91	14832
macro avg	0.91	0.91	0.91	14832
weighted avg	0.91	0.91	0.91	14832

DecisionTreeClassifier

DecisionTreeClassifier()

mymodel(rf)

	precision	recall	f1-score	support
0	0.97	0.87	0.92	7393
1	0.89	0.97	0.93	7439
accuracy			0.92	14832
macro avg	0.93	0.92	0.92	14832
weighted avg	0.93	0.92	0.92	14832

RandomForestClassifier

RandomForestClassifier(random_state=42)

Step 4: Hyperparameter Tuning using GridSearchCV

```
from sklearn.model_selection import GridSearchCV
```

4.1.1: GridSearchCV For Logistic Regression

```
parameters={'solver':['poly','liblinear','lbfgs','saga','sag','newton-cg'],'penalty':['none','l1','l2','elasticnet'],'C':[100,10,
gs=GridSearchCV(logreg,parameters,verbose=4)
```

```
gs.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 120 candidates, totalling 600 fits
[CV 1/5] END ....C=100, penalty=none, solver=poly;, score=nan total time=   0.0s
[CV 2/5] END ....C=100, penalty=none, solver=poly;, score=nan total time=   0.0s
[CV 3/5] END ....C=100, penalty=none, solver=poly;, score=nan total time=   0.0s
[CV 4/5] END ....C=100, penalty=none, solver=poly;, score=nan total time=   0.0s
[CV 5/5] END ....C=100, penalty=none, solver=poly;, score=nan total time=   0.0s
[CV 1/5] END C=100, penalty=none, solver=liblinear;, score=nan total time=   0.0s
[CV 2/5] END C=100, penalty=none, solver=liblinear;, score=nan total time=   0.0s
[CV 3/5] END C=100, penalty=none, solver=liblinear;, score=nan total time=   0.0s
[CV 4/5] END C=100, penalty=none, solver=liblinear;, score=nan total time=   0.0s
[CV 5/5] END C=100, penalty=none, solver=liblinear;, score=nan total time=   0.0s
[CV 1/5] END ...C=100, penalty=none, solver=lbfgs;, score=nan total time=   0.0s
[CV 2/5] END ...C=100, penalty=none, solver=lbfgs;, score=nan total time=   0.0s
[CV 3/5] END ...C=100, penalty=none, solver=lbfgs;, score=nan total time=   0.0s
[CV 4/5] END ...C=100, penalty=none, solver=lbfgs;, score=nan total time=   0.0s
[CV 5/5] END ...C=100, penalty=none, solver=lbfgs;, score=nan total time=   0.0s
[CV 1/5] END ....C=100, penalty=none, solver=saga;, score=nan total time=   0.0s
[CV 2/5] END ....C=100, penalty=none, solver=saga;, score=nan total time=   0.0s
[CV 3/5] END ....C=100, penalty=none, solver=saga;, score=nan total time=   0.0s
```

```
gs.best_params_
```

```
{'C': 0.1, 'penalty': 'l1', 'solver': 'liblinear'}
```

4.1.3 Getting best accuracy for Logistic Regression as per Hyper tuned best Parameters

```
logreg=LogisticRegression(C=0.1,penalty='l2',solver='newton-cg')

logreg.fit(x_train,y_train)
y_pred = logreg.predict(x_test)
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.77      0.76      0.77      7393
           1       0.77      0.77      0.77      7439

    accuracy                           0.77     14832
   macro avg       0.77      0.77      0.77     14832
weighted avg       0.77      0.77      0.77     14832
```

4.2: GridSearchCV For Decision Tree

```
parameters={'max_depth':['none',10,20,30],'min_samples_split':[2,5,10],'min_samples_leaf':[1,2,4]}
gs=GridSearchCV(dt,parameters,verbose=4)
```

```
gs.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[CV 1/5] END max_depth=none, min_samples_leaf=1, min_samples_split=2;, score=nan total time=   0.0s
[CV 2/5] END max_depth=none, min_samples_leaf=1, min_samples_split=2;, score=nan total time=   0.0s
[CV 3/5] END max_depth=none, min_samples_leaf=1, min_samples_split=2;, score=nan total time=   0.0s
[CV 4/5] END max_depth=none, min_samples_leaf=1, min_samples_split=2;, score=nan total time=   0.0s
[CV 5/5] END max_depth=none, min_samples_leaf=1, min_samples_split=2;, score=nan total time=   0.0s
[CV 1/5] END max_depth=none, min_samples_leaf=1, min_samples_split=5;, score=nan total time=   0.0s
[CV 2/5] END max_depth=none, min_samples_leaf=1, min_samples_split=5;, score=nan total time=   0.0s
[CV 3/5] END max_depth=none, min_samples_leaf=1, min_samples_split=5;, score=nan total time=   0.0s
[CV 4/5] END max_depth=none, min_samples_leaf=1, min_samples_split=5;, score=nan total time=   0.0s
[CV 5/5] END max_depth=none, min_samples_leaf=1, min_samples_split=5;, score=nan total time=   0.0s
[CV 1/5] END max_depth=none, min_samples_leaf=1, min_samples_split=10;, score=nan total time=   0.0s
[CV 2/5] END max_depth=none, min_samples_leaf=1, min_samples_split=10;, score=nan total time=   0.0s
[CV 3/5] END max_depth=none, min_samples_leaf=1, min_samples_split=10;, score=nan total time=   0.0s
[CV 4/5] END max_depth=none, min_samples_leaf=1, min_samples_split=10;, score=nan total time=   0.0s
[CV 5/5] END max_depth=none, min_samples_leaf=1, min_samples_split=10;, score=nan total time=   0.0s
[CV 1/5] END max_depth=none, min_samples_leaf=2, min_samples_split=2;, score=nan total time=   0.0s
[CV 2/5] END max_depth=none, min_samples_leaf=2, min_samples_split=2;, score=nan total time=   0.0s
[CV 3/5] END max_depth=none, min_samples_leaf=2, min_samples_split=2;, score=nan total time=   0.0s
```

```
gs.best_params_
```

```
{'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

```
dt=DecisionTreeClassifier(max_depth=30,min_samples_leaf=1,min_samples_split=2)

dt.fit(x_train,y_train)
y_pred = dt.predict(x_test)
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.85      0.90      7393
           1       0.87      0.96      0.91      7439

    accuracy                           0.91     14832
   macro avg       0.91      0.91      0.90     14832
weighted avg       0.91      0.91      0.90     14832
```

Conclusion:

In this project, we build various models like logistic regression, knn classifier,decision tree classifier, random forest classifier and Hyper Tuning Parameter Using GridSerachCV for logistic regression & decision tree classifier.

A Random Forest Classifier gives the highest accuracy score of 92.00 percent among all the ML Models.

Thus, for the Adult Census Income Prediction Dataset, the best accuracy can be achieved using Random Forest Model.

THANK YOU!