Income Prediction from Census Data: A Machine Learning Analysis

Welcome to this Jupyter Notebook, where we will dive into the fascinating world of data analysis and machine learning. In this notebook, we'll be working with the "Census Income" dataset, also known as the "Census Income" dataset, to predict whether an individual's income exceeds $50,000 per year based on various census attributes.

Import Python libraries

In [1]:

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
df=pd.read_csv(r"D:\Data Science Course\Machine Learning\Classification\1.Logistic Regression\1.LOGISTIC REGRESSION CODE
```

In [3]:

```
df.head()
```

Out[3]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.lo
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female	0	43
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	43
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0	43
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	39
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	39

In [4]:

```
df.shape
```

Out[4]:

```
(32561, 15)
```

View summary of dataframe

In [5]:

```
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       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
```

Summary of the dataset shows that there are no missing values. But the preview shows that the dataset contains values coded as `?` . So, I will encode `?` as NaN values.

Missing value Analysis

In [6]:

```
df[df=='?']=np.nan
```

In [7]:

```
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
```

Now, the summary shows that the variables - workclass, occupation and native.country contain missing values. All of these variables are categorical data type. So, I will impute the missing values with the most frequent value- the mode.

Impute missing values with mode

In [8]:

```
for col in ['workclass', 'occupation', 'native.country']:
    df[col].fillna(df[col].mode()[0], inplace=True)
```

Check again for missing values

In [9]:

```
df.isnull().sum()
```

Out[9]:

```
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
```

In [10]:

```
#doing private job and salary more than 50K
new_df=df[df['workclass']=='Private']
print(len(new_df[new_df['income']== '>50K']))
new_df[new_df['income']== '>50K'].head(2)
```

```
5154
```

Out[10]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.l
9	41	Private	70037	Some-college	10	Never-married	Craft-repair	Unmarried	White	Male	0	3
10	45	Private	172274	Doctorate	16	Divorced	Prof-specialty	Unmarried	Black	Female	0	3

Insights

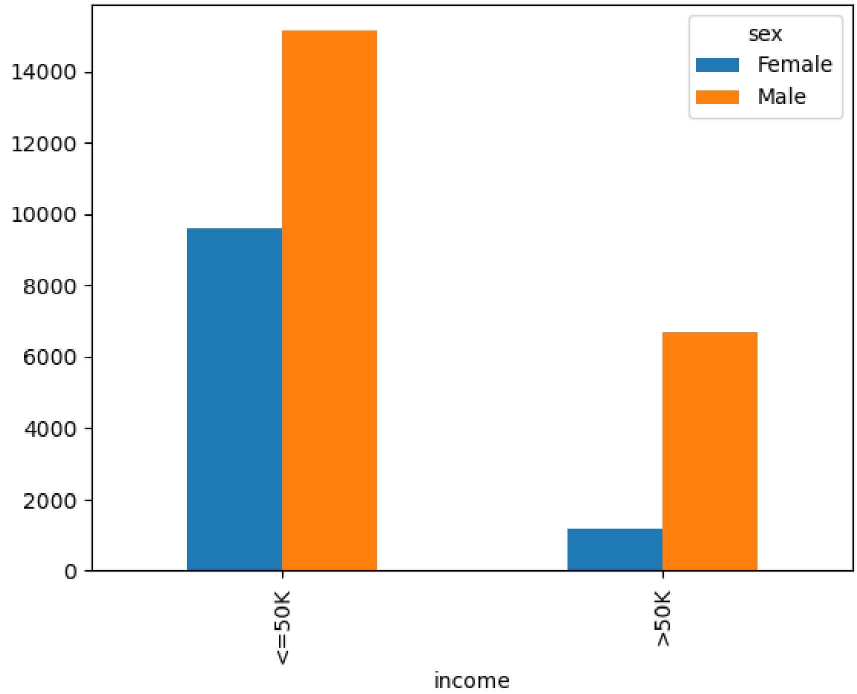
- There is 4876 people are doing job in private class and earn more than 50K

In [11]:

```
#cheacking male and female ration according to their salary
class_sex = pd.crosstab(df['income'],df['sex'])
class_sex.plot(kind='bar')
```

Out[11]:

```
<Axes: xlabel='income'>
```



Insights

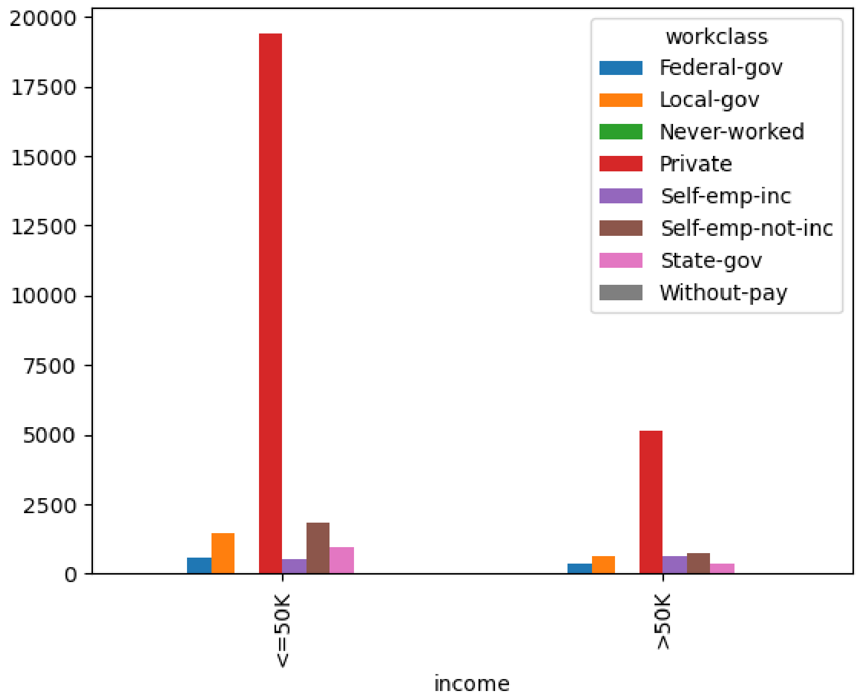
- Income Distribution: The majority of individuals fall into the income category <=50k, indicating that a significant portion of the population earns a lower income.
- Gender Disparity: Within both income categories, males tend to have a higher income compared to females. This gender-based income disparity is a noteworthy observation

In [12]:

```
#cheacking distribution of people in workclass according to salary
class_salary = pd.crosstab(df['income'],df['workclass'])
class_salary.plot(kind='bar')
```

Out[12]:

```
<Axes: xlabel='income'>
```



Insights

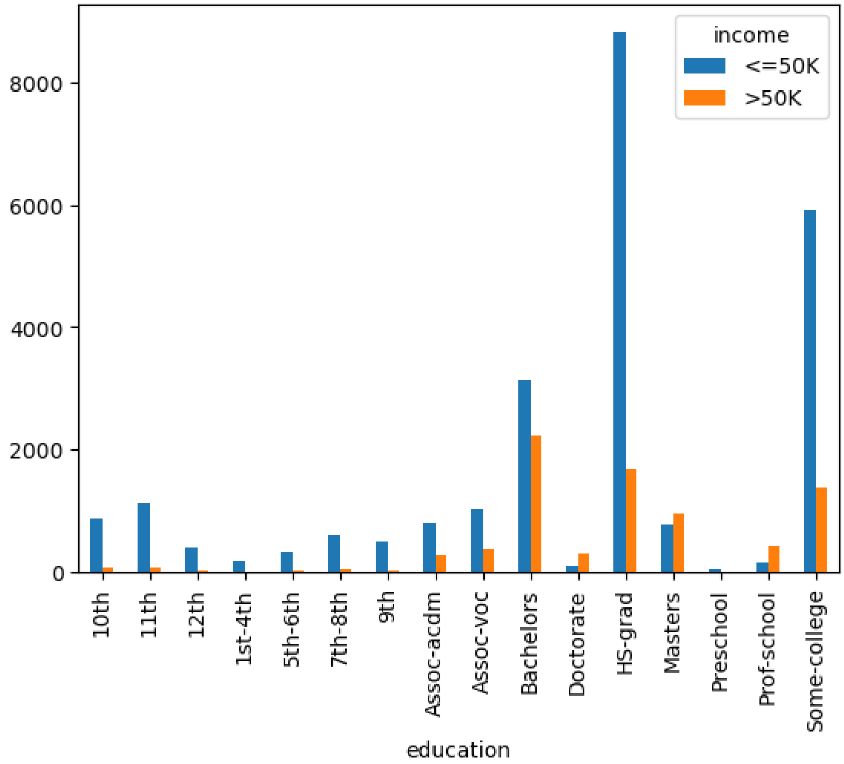
- we can clearly see that private class people is more than other classes in both type of category of salary

In [13]:

```
#cheacking distribution of people according to salary
salary_education = pd.crosstab(df['education'],df['income'])
salary_education.plot(kind='bar')
```

Out[13]:

```
<Axes: xlabel='education'>
```



In [14]:

```
num_col=df[['age','education']]
```

Target And Independent Variable

In [15]:

```
x=df.drop(['income'],axis=1)
y=df['income']
```

In [16]:

```
x.head()
```

Out[16]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.lo
0	90	Private	77053	HS-grad	9	Widowed	Prof-specialty	Not-in-family	White	Female	0	43
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	43
2	66	Private	186061	Some-college	10	Widowed	Prof-specialty	Unmarried	Black	Female	0	43
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	39
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	39

In [17]:

```
y.head()
```

Out[17]:

```
0    <=50K
1    <=50K
2    <=50K
3    <=50K
4    <=50K
Name: income, dtype: object
```

Split data into separate training and test set

In [18]:

```
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 = 0)
```

Encode categorical variables

In [19]:

```
from sklearn import preprocessing

categorical = ['workclass', 'education', 'marital.status', 'occupation', 'relationship', 'race', 'sex', 'native.country
for feature in categorical:
        le = preprocessing.LabelEncoder()
        x_train[feature] = le.fit_transform(x_train[feature])
        x_test[feature] = le.transform(x_test[feature])
```

Feature Scaling

In [20]:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

x_train = pd.DataFrame(scaler.fit_transform(x_train), columns = x.columns)

x_test = pd.DataFrame(scaler.transform(x_test), columns = x.columns)
```

Logistic Regression model with all features

In [21]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

logreg = LogisticRegression()
logreg.fit(x_train, y_train)
y_pred = logreg.predict(x_test)

print('Logistic Regression accuracy score with all the features: {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Logistic Regression accuracy score with all the features: 0.8217
```

PCA implementation

In [22]:

```
from sklearn.decomposition import PCA
pca = PCA()
x_train = pca.fit_transform(x_train)
```

In [23]:

```
pca.explained_variance_ratio_
```

Out[23]:

```
array([0.14757168, 0.10182915, 0.08147199, 0.07880174, 0.07463545,
       0.07274281, 0.07009602, 0.06750902, 0.0647268 , 0.06131155,
       0.06084207, 0.04839584, 0.04265038, 0.02741548])
```

Comment

- We can see that approximately 97.25% of variance is explained by the first 13 variables.
- Only 2.75% of variance is explained by the last variable. So, we can assume that it carries little information.
- So, I will drop it, train the model again and calculate the accuracy.

Logistic Regression with first 13 features

In [24]:

```
x = df.drop(['income','native.country'], axis=1)
y = df['income']


x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 0)


categorical = ['workclass', 'education', 'marital.status', 'occupation', 'relationship', 'race', 'sex']
for feature in categorical:
        le = preprocessing.LabelEncoder()
        x_train[feature] = le.fit_transform(x_train[feature])
        x_test[feature] = le.transform(x_test[feature])


x_train = pd.DataFrame(scaler.fit_transform(x_train), columns = x.columns)

x_test = pd.DataFrame(scaler.transform(x_test), columns = x.columns)

logreg = LogisticRegression()
logreg.fit(x_train, y_train)
y_pred = logreg.predict(x_test)

print('Logistic Regression accuracy score with the first 13 features: {0:0.4f}'. format(accuracy_score(y_test, y_pred))
```

```
Logistic Regression accuracy score with the first 13 features: 0.8213
```

Comment

- We can see that accuracy has been decreased from 0.8218 to 0.8213 after dropping the last feature.
- Now, if I take the last two features combined, then we can see that approximately 7% of variance is explained by them.
- I will drop them, train the model again and calculate the accuracy.

Logistic Regression with first 12 features

In [25]:

```
X = df.drop(['income','native.country', 'hours.per.week'], axis=1)
y = df['income']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)


categorical = ['workclass', 'education', 'marital.status', 'occupation', 'relationship', 'race', 'sex']
for feature in categorical:
        le = preprocessing.LabelEncoder()
        X_train[feature] = le.fit_transform(X_train[feature])
        X_test[feature] = le.transform(X_test[feature])


X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = X.columns)

X_test = pd.DataFrame(scaler.transform(X_test), columns = X.columns)

logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)

print('Logistic Regression accuracy score with the first 12 features: {0:0.4f}'. format(accuracy_score(y_test, y_pred))
```

```
Logistic Regression accuracy score with the first 12 features: 0.8227
```

In [26]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

In [27]:

```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print('Logistic Regression accuracy score with the first 12 features: {0:0.4f}'. format(accuracy_score(y_test, y_pred))
```

```
Logistic Regression accuracy score with the first 12 features: 0.8281
```

In [28]:

```
svc = SVC()
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)
print('Logistic Regression accuracy score with the first 12 features: {0:0.4f}'. format(accuracy_score(y_test, y_pred))
```

```
Logistic Regression accuracy score with the first 12 features: 0.8423
```

Comment

- Now, it can be seen that the highest accuracy is now 0.8423, if the model is trained with 12 features using SVC algorithm.

Conclusion

- I have demonstrated PCA implementation with Logistic Regression,KNN & SVC on the adult dataset.
- I found the maximum accuracy with the first 12 features and it is found to be 0.8423 with SVC.

Thank You