ASBD PROJECT

Group Members

Name	Roll Number
HARINI .V	CED18I023
THOTA SAI KEERTHANA	CED18I053
UPPALAPATI PRANITA	CED18I062
S S VARSHIT	EDM18B050

DATASET : Census Income Data Set

Dataset link : https://archive.ics.uci.edu/ml/datasets/census+income

Attribute Information:
age : the age of an individual
workclass : a general term to represent the employment status of an individual
fnlwgt : final weight. In other words, this is the number of people the census believes the entry represents.
education : the highest level of education achieved by an individual.
education_num : the highest level of education achieved in numerical form.
marital_status : marital status of an individual.
occupation : the general type of occupation of an individual
relationship : represents what this individual is relative to others. For example an individual could be a Husband. Each entry only has one relationship attribute and is somewhat redundant with marital status.
race : Descriptions of an individual's race
sex : the biological sex of the individual
capital_gain : capital gains for an individual
capital_loss : capital loss for an individual
hours_per_week : the hours an individual has reported to work per week
native_country : country of origin for an individual
income : whether or not an individual makes more than $50,000 annually.

Data Cleaning:

Importing the required libraries

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import statistics
import seaborn as sns
import csv
```

Renaming the column names

```
col_names=['age','workclass','fnlwgt','education','education_num','marital_status','occupation','relationship','race','sex'
,'capital_gain','capital_loss','hours_per_week','native_country','income']


data=pd.read_csv("adult.data",names=col_names,header=None)
data.head(5)
```

Out[2]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40

In [3]:
```
data.shape
```

Out[3]: (32561, 15)

In [4]:
```
data.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
```

```
In [5]: data.describe()
```

Out[5]:

	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

Dropping duplicates

```
#dropping duplicate rows if any
data=data.drop_duplicates(keep='first', inplace=False)
data=data.reset_index(drop=True)
data.shape
```

```
Out[6]: (32537, 15)
```

The data had 24 duplicate records which have been removed now

Handling Missing Data

```
#displaying number of null values or NaN values in each column
data.isnull().sum()
```

```
Out[13]: 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
```

There are NO Null values in the dataset

Handling Categorical Attributes

- All the categorical attributes have been converted into numerical values using label encoder

```
from sklearn.preprocessing import LabelEncoder
label_enc = LabelEncoder()

data['sex_label'] = label_enc.fit_transform(data['sex'])
sex_labels=label_enc.classes_
data['race_label'] = label_enc.fit_transform(data['race'])
race_labels=label_enc.classes_
```

```
data['workclass_label'] = label_enc.fit_transform(data['workclass'])
workclass_labels=label_enc.classes_
data['marital_status_label'] = label_enc.fit_transform(data['marital_status'])
marital_status_labels=label_enc.classes_
data['occupation_label'] = label_enc.fit_transform(data['occupation'])
occupation_labels=label_enc.classes_
data['relationship_label'] = label_enc.fit_transform(data['relationship'])
relationship_labels=label_enc.classes_
data['native_country_label'] = label_enc.fit_transform(data['native_country'])
native_country_labels=label_enc.classes_
data['income_label'] = label_enc.fit_transform(data['income'])
```

```
In [32]: sex_labels
Out[32]: array([' Female', ' Male'], dtype=object)

In [33]: race_labels
Out[33]: array([' Amer-Indian-Eskimo', ' Asian-Pac-Islander', ' Black', ' Other',
                ' White'], dtype=object)

In [34]: workclass_labels
Out[34]: array([' ?', ' Federal-gov', ' Local-gov', ' Never-worked', ' Private',
                ' Self-emp-inc', ' Self-emp-not-inc', ' State-gov', ' Without-pay'],
               dtype=object)

In [35]: marital_status_labels
Out[35]: array([' Divorced', ' Married-AF-spouse', ' Married-civ-spouse',
                ' Married-spouse-absent', ' Never-married', ' Separated',
                ' Widowed'], dtype=object)

In [36]: occupation_labels
Out[36]: array([' ?', ' Adm-clerical', ' Armed-Forces', ' Craft-repair',
                ' Exec-managerial', ' Farming-fishing', ' Handlers-cleaners',
                ' Machine-op-inspct', ' Other-service', ' Priv-house-serv',
                ' Prof-specialty', ' Protective-serv', ' Sales', ' Tech-support',
                ' Transport-moving'], dtype=object)

In [37]: relationship_labels
Out[37]: array([' Husband', ' Not-in-family', ' Other-relative', ' Own-child',
                ' Unmarried', ' Wife'], dtype=object)

In [38]: native_country_labels
Out[38]: array([' ?', ' Cambodia', ' Canada', ' China', ' Columbia', ' Cuba',
                ' Dominican-Republic', ' Ecuador', ' El-Salvador', ' England',
                ' France', ' Germany', ' Greece', ' Guatemala', ' Haiti',
                ' Holand-Netherlands', ' Honduras', ' Hong', ' Hungary', ' India',
                ' Iran', ' Ireland', ' Italy', ' Jamaica', ' Japan', ' Laos',
                ' Mexico', ' Nicaragua', ' Outlying-US(Guam-USVI-etc)', ' Peru',
                ' Philippines', ' Poland', ' Portugal', ' Puerto-Rico',
                ' Scotland', ' South', ' Taiwan', ' Thailand', ' Trinadad&Tobago',
                ' United-States', ' Vietnam', ' Yugoslavia'], dtype=object)
```

INFERENCE:
We see that though there are no null values shown , here we find that there are '?' in workclass, native_country and occupation columns. They have all been labeled to 0 in label encoder. For now we are not doing anything to remove these ‘ ?’ . We will later decide what to do with these records based on the features we are going to select while doing Predictive Analysis.

Now we have saved this cleaned training dataset to “cleaned_train_data.csv” for further usage . Along with this we have also saved all the labellings of each column into “label_encodings.csv” . We will need these labellings because we need to give same labels to values test data also.

```
data.to_csv('cleaned_train_data.csv')
col_names = ['Column_name', 'Label_encoded']
```

```
label_encodings= pd.DataFrame(columns = col_names)
dict1 = {'Column_name':['sex','race','work_class','marital_status','occupation','relationship','native_country'],
'Label_encoded':[sex_labels,race_labels,workclass_labels,marital_status_labels,occupation_labels,relationship_labels,native_country_labels]
    }
label_encodings = pd.DataFrame(dict1)
label_encodings
```

Out[44]:

	Column_name	Label_encoded
0	sex	[Female, Male]
1	race	[Amer-Indian-Eskimo, Asian-Pac-Islander, Bl...
2	work_class	[?, Federal-gov, Local-gov, Never-worked, ...
3	marital_status	[Divorced, Married-AF-spouse, Married-civ-s...
4	occupation	[?, Adm-clerical, Armed-Forces, Craft-repa...
5	relationship	[Husband, Not-in-family, Other-relative, O...
6	native_country	[?, Cambodia, Canada, China, Columbia, C...

```
label_encodings.to_csv('label_encodings.csv')
```

EDA:

We now load the ‘cleaned_train_data.csv’

```
censusData = pd.read_csv('cleaned_train_data.csv')
```

```
# Age Statistics
print("Age Statistics")
print(censusData['age'].describe())
print("Median Age: ", censusData['age'].median())
```

```
Age Statistics
count    32537.000000
mean        38.585549
std         13.637984
min         17.000000
25%         28.000000
50%         37.000000
75%         48.000000
max         90.000000
Name: age, dtype: float64
Median Age:  37.0
```

Here, we use the describe() function to describe the basic statistics.
1. Count: Number of values of age in the data.
2. Mean: Mean of all the ages.
3. Std: Standard deviation of all the ages.
4. Min: Minimum age in data.
5. 25%: 25th percentile value in data.
6. 50%: 50th percentile value in data.
7. 75%: 75th percentile value in data.
8. Max: Maximum age in data.

Apart from these, we also calculate the median value of age. We are now going to calculate these values for all numerical attributes.

In [4]:
```
# Final weight Statistics
print("Final Weight Statistics")
print(censusData['fnlwgt'].describe())
print("Median Final Weight: ", censusData['fnlwgt'].median())
```

```
Final Weight Statistics
count    3.253700e+04
mean     1.897808e+05
std      1.055565e+05
min      1.228500e+04
25%      1.178270e+05
50%      1.783560e+05
75%      2.369930e+05
max      1.484705e+06
Name: fnlwgt, dtype: float64
Median Final Weight:  178356.0
```

In [5]:
```
# Education number Statistics
print("Education Number Statistics")
print(censusData['education_num'].describe())
print("Median Education Number: ", censusData['education_num'].median())
```

```
Education Number Statistics
count    32537.000000
mean        10.081815
std          2.571633
min          1.000000
25%          9.000000
50%         10.000000
75%         12.000000
max         16.000000
Name: education_num, dtype: float64
Median Education Number:  10.0
```

In [6]:
```
# Capital Gain Statistics
print("Capital Gain Statistics")
print(censusData['capital_gain'].describe())
print("Median Capital Gain: ", censusData['capital_gain'].median())
```

```
Capital Gain Statistics
count    32537.000000
mean      1078.443741
std       7387.957424
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max      99999.000000
Name: capital_gain, dtype: float64
Median Capital Gain:  0.0
```

In [7]:
```
#Hours per week Statistics
print("Hous per week Statistics")
print(censusData['hours_per_week'].describe())
print("Median Hours per week: ", censusData['hours_per_week'].median())
```

```
Hous per week Statistics
count    32537.000000
mean        40.440329
std         12.346889
min          1.000000
25%         40.000000
50%         40.000000
75%         45.000000
max         99.000000
Name: hours_per_week, dtype: float64
Median Hours per week:  40.0
```

Summary of Categorical Data

```
[14]: censusData.describe(include=object)
```

[14]:

	workclass	education	marital_status	occupation	relationship	race	sex	native_country	income
count	32537	32537	32537	32537	32537	32537	32537	32537	32537
unique	9	16	7	15	6	5	2	42	2
top	Private	HS-grad	Married-civ-spouse	Prof-specialty	Husband	White	Male	United-States	<=50K
freq	22673	10494	14970	4136	13187	27795	21775	29153	24698

Summary of Numerical Data

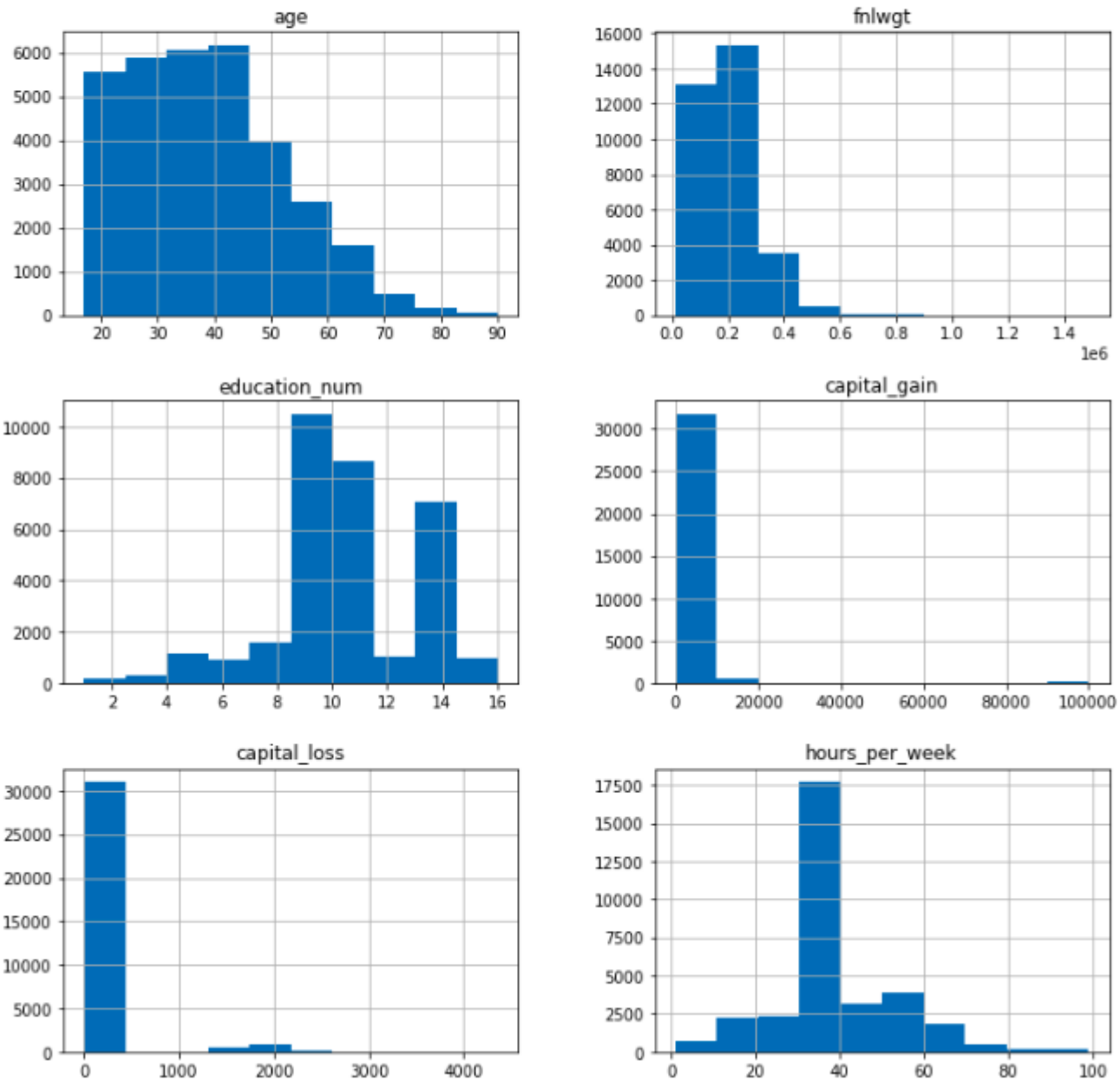
```
[15]: censusData.describe(include=['int64'])
```

[15]:

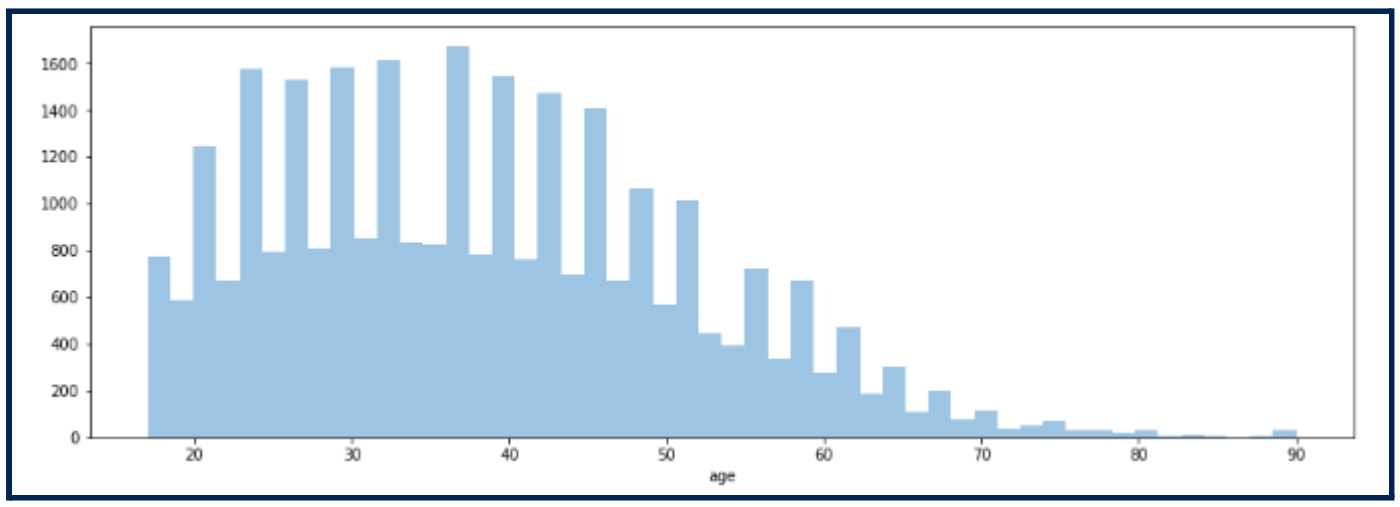
	age	fnlwgt	education_num	capital_gain	capital_loss	hours_per_week
count	32537.000000	3.253700e+04	32537.000000	32537.000000	32537.000000	32537.000000
mean	38.585549	1.897808e+05	10.081815	1078.443741	87.368227	40.440329
std	13.637984	1.055565e+05	2.571633	7387.957424	403.101833	12.346889
min	17.000000	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

Data visualization of Numerical Attributes

```
Out[16]: array([[<AxesSubplot:title={'center':'age'}>,
                 <AxesSubplot:title={'center':'fnlwgt'}>],
                [<AxesSubplot:title={'center':'education_num'}>,
                 <AxesSubplot:title={'center':'capital_gain'}>],
                [<AxesSubplot:title={'center':'capital_loss'}>,
                 <AxesSubplot:title={'center':'hours_per_week'}>]], dtype=object)
```



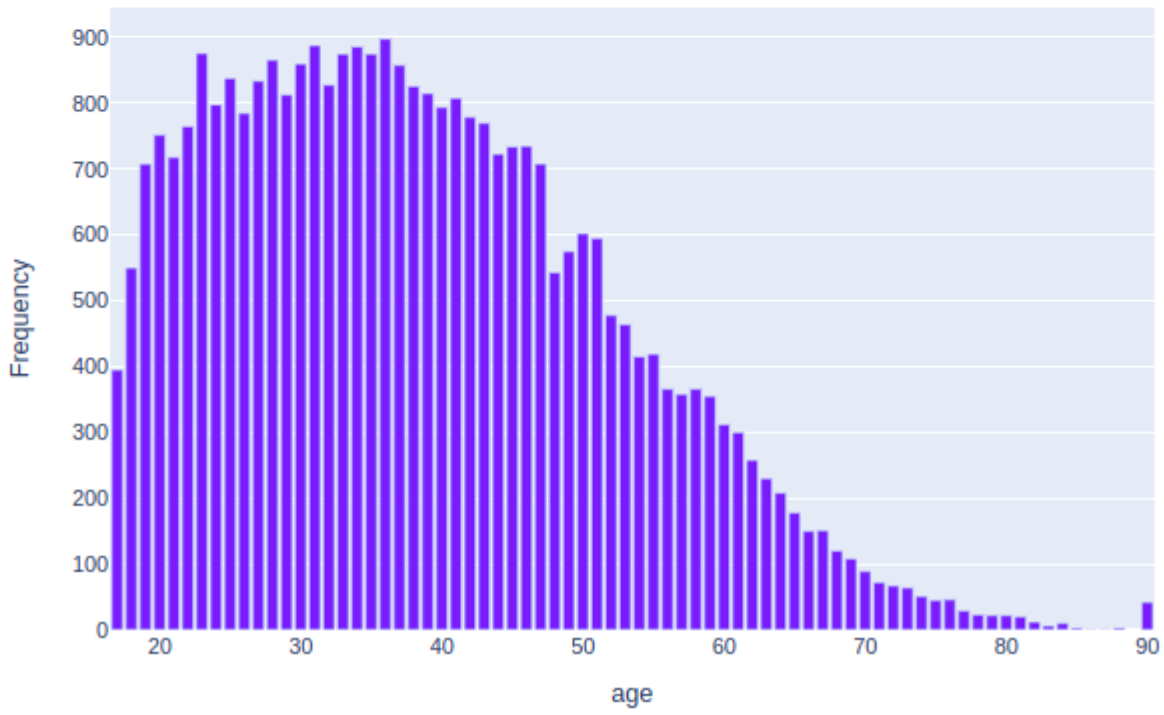

```
plt.figure(figsize=(15,5))
sns.distplot(censusData['age'],kde=False)
plt.show()
```



```
In [18]: import plotly.express as px
         import plotly.offline as pyo
```

```
In [19]: age_counts = censusData["age"].value_counts()
```

```
In [20]: fig = px.bar(age_counts, title="Age of respondents")
         fig.update_layout(
             xaxis_title = "age",
             yaxis_title = "Frequency",
             title_x = 0.5,
             showlegend = False
         )
         fig.show()
```

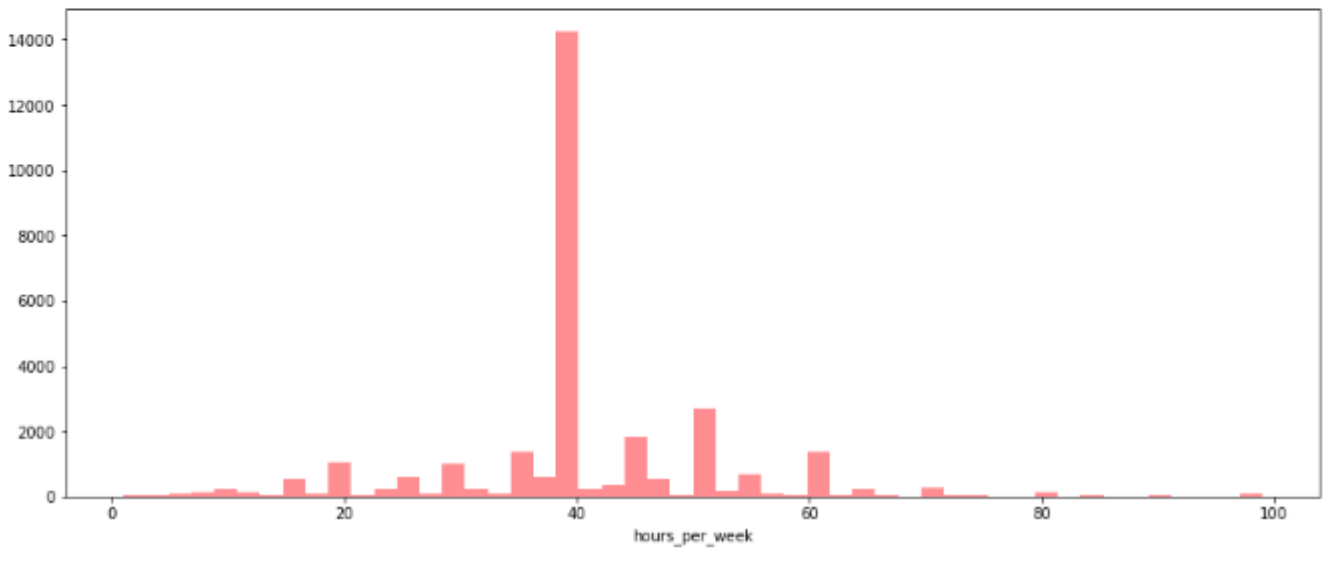


Majority population are aged around 40.

```
In [93]: (censusData.age>70).sum()       # right skewed but data > 70 is less person
Out[93]: 379
```

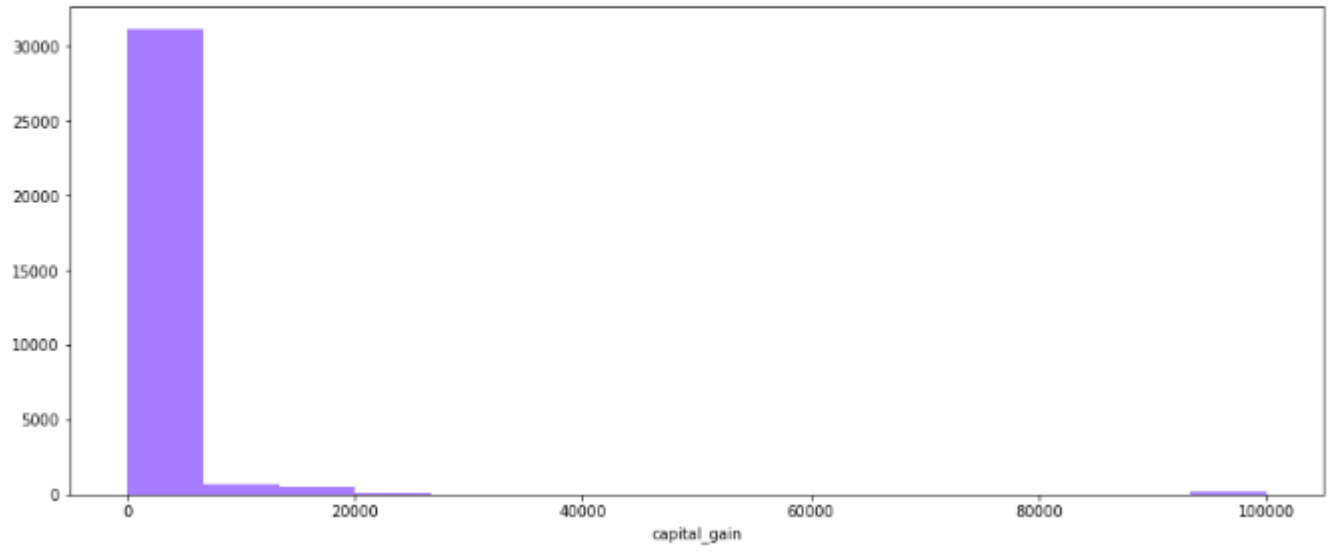
```
In [94]: plt.figure(figsize=(15,6))
         sns.distplot(censusData['hours_per_week'],kde=False,color='r')
         plt.show()
```

```
/home/ubuntu/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
```



3rd quartile is 45 i.e 75% spend less than 45 hr/week.
Majority population spend b/w 40-45 hr/wk

In [23]:
```
plt.figure(figsize=(15,6))
sns.distplot(censusData['capital_gain'],kde=False,color='b',bins=15)
plt.show()
```

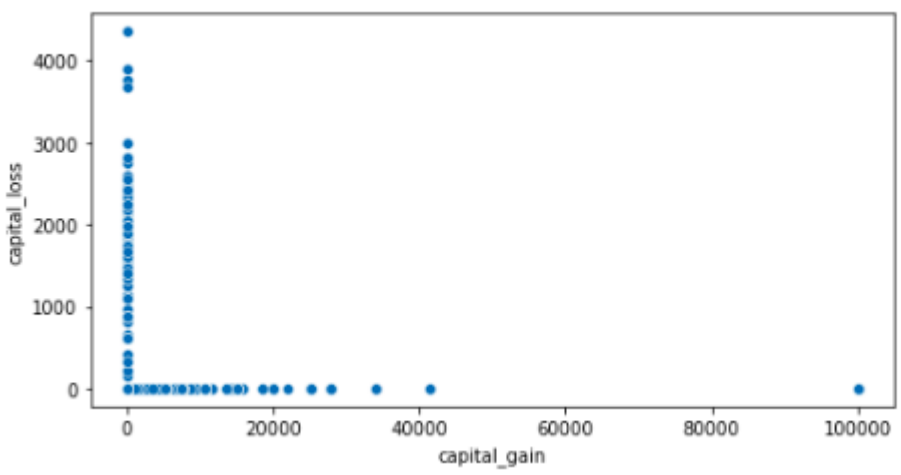


It shows either no gain or high gain due to high deviation.

In [24]:
```
print(censusData[['capital_gain','capital_loss']].corr())

plt.figure(figsize=(8,4))
sns.scatterplot(censusData['capital_gain'],censusData['capital_loss'])
plt.show()
```

```
              capital_gain  capital_loss
capital_gain      1.000000     -0.031639
capital_loss     -0.031639      1.000000
```

```
/home/ubuntu/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWar
ning:

Pass the following variables as keyword args: x, y. From version 0.12, the only valid p
ositional argument will be `data`, and passing other arguments without an explicit keyw
ord will result in an error or misinterpretation.
```



Capital gain and loss have a negative weak relationship.
If one is 0 ,another is high.

INFERENCES:

1. Most of the individuals have an age < 50 while the maximum age is around 90.

2. In general, people do not have investments other than their regular income. However, there are very few people who invest, and there are also a small number of outliers who earn more than 90000 via capital gains. However, among the people who had a capital loss the average loss looks to be around 2000.

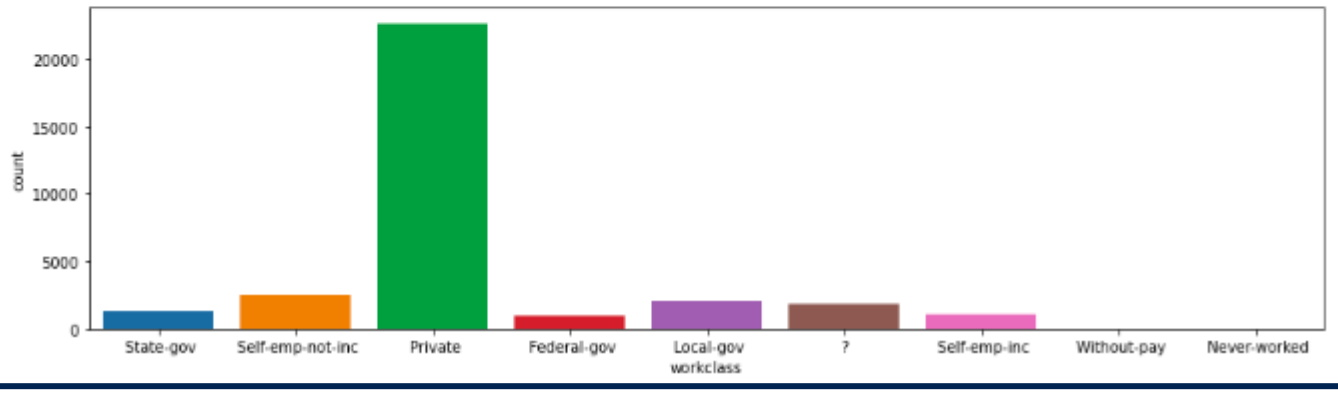
3. On average, most of the people have studied till education number 9 or 10 in the areas where the census was taken.

4. Most of the people work around 40 hrs per week. However there are a few who don’t work and a few who work for almost 100 hours a week.

Categorical Data

In [25]:
```
#plotting count plot for categorical values
categorical_attributes = censusData.select_dtypes(include=['object'])

plt.figure(figsize=(15,4))
sns.countplot(data=categorical_attributes,x='workclass')
```

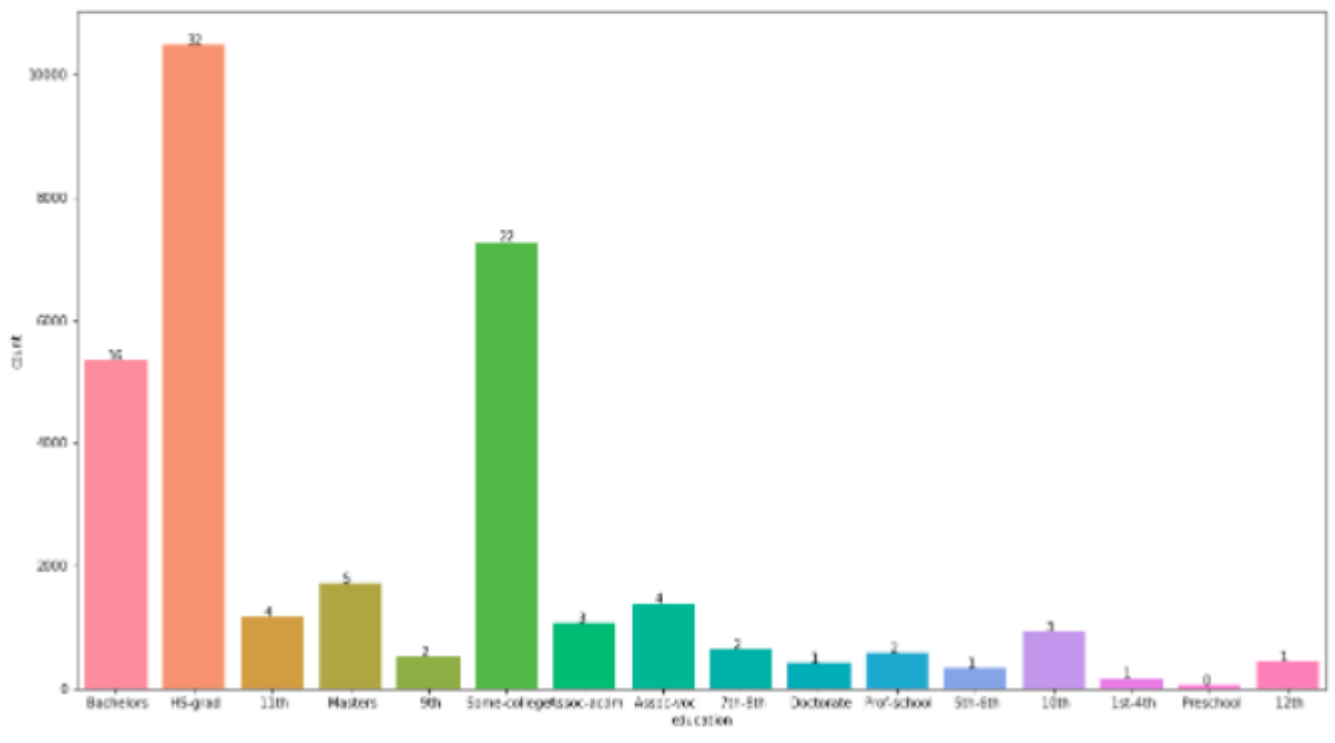
Out[25]: <AxesSubplot:xlabel='workclass', ylabel='count'>



Most of the people work in private sectors, and the rest are evenly distributed among state-gov, federal-gov, local-gov, self-emp-inc and self-emp-not-inc.

In [26]:
```
plt.figure(figsize=(18,10))

ax = sns.countplot(censusData['education'])

for p in ax.patches:

    x = p.get_x()+p.get_width()/2 - 0.1
    y = p.get_height() + 3
    t = round((p.get_height()/censusData.shape[0])*100)

    ax.text(x,y,t)

plt.show()
```

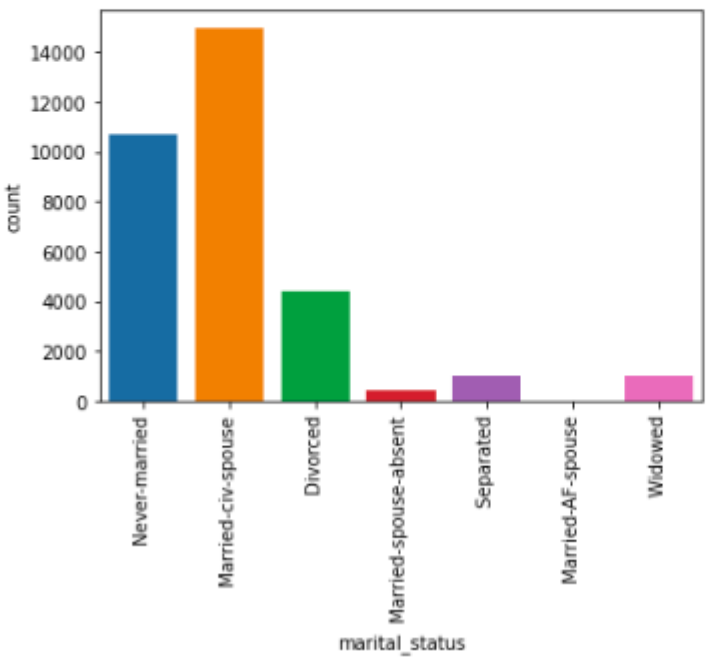
```
/home/ubuntu/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWar
ning:

Pass the following variable as a keyword arg: x. From version 0.12, the only valid posi
tional argument will be `data`, and passing other arguments without an explicit keyword
will result in an error or misinterpretation.
```



Most of the people are high-school grads or have studied in some college. This is the same as the education_num attribute, just that each of these values have been assigned a number there. We can use one of the two columns for the model, and ignore the other.

In [27]:
```
sns.countplot(censusData['marital_status'])
plt.xticks(rotation = 90)
plt.show()
```

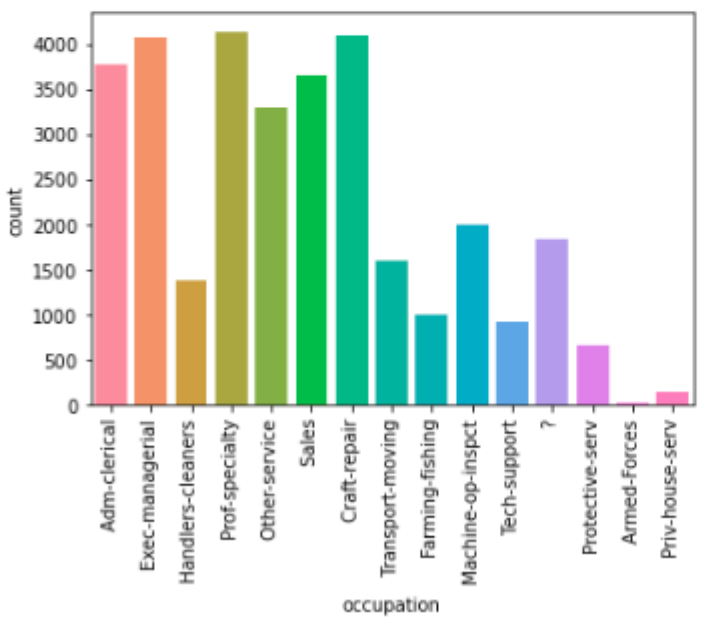
```
/home/ubuntu/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWar
ning:

Pass the following variable as a keyword arg: x. From version 0.12, the only valid posi
tional argument will be `data`, and passing other arguments without an explicit keyword
will result in an error or misinterpretation.
```



Most of the people are either married civilians or are never married.

In [28]:
```
sns.countplot(censusData['occupation'])
plt.xticks(rotation = 90)
plt.show()
```

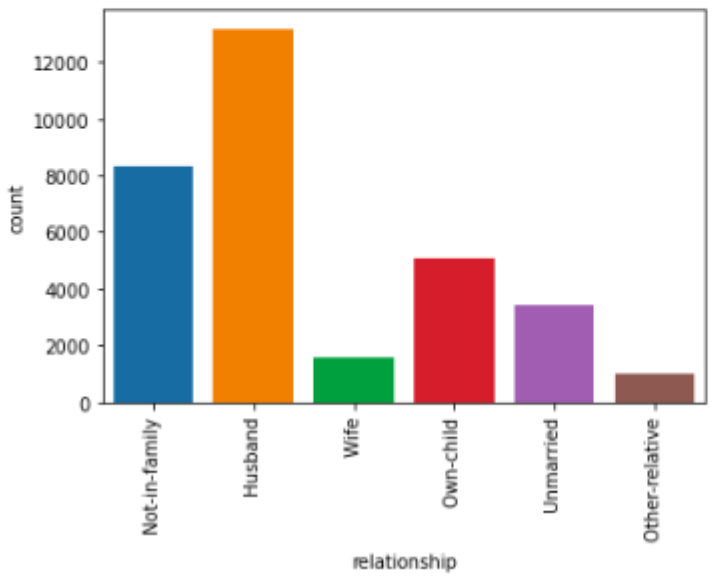
```
/home/ubuntu/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWar
ning:

Pass the following variable as a keyword arg: x. From version 0.12, the only valid posi
tional argument will be `data`, and passing other arguments without an explicit keyword
will result in an error or misinterpretation.
```



The jobs/occupation of the individuals are evenly distributed among a lot of values, like clerical administrator, executive managerial posts, professional speciality, sales, craft-repair and other-services.

In [29]:
```
sns.countplot(censusData['relationship'])
plt.xticks(rotation = 90)
plt.show()
```

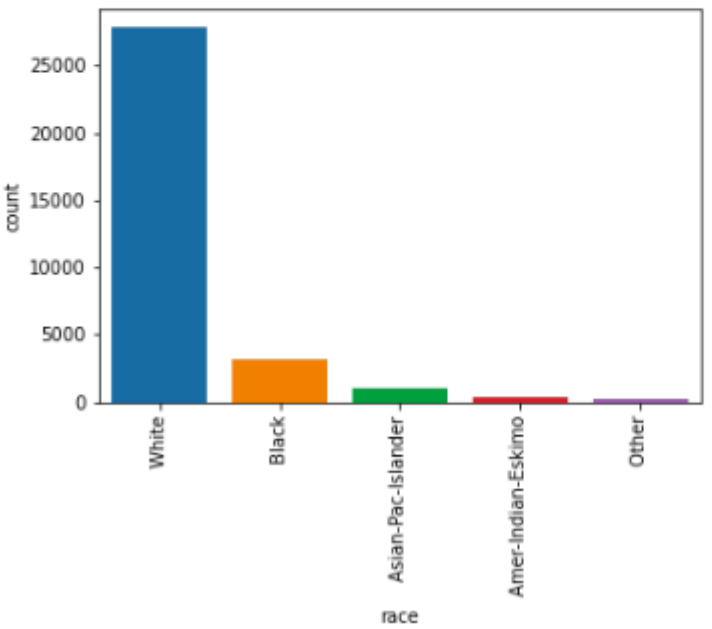
```
/home/ubuntu/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWar
ning:

Pass the following variable as a keyword arg: x. From version 0.12, the only valid posi
tional argument will be `data`, and passing other arguments without an explicit keyword
will result in an error or misinterpretation.
```



Most of the people in the survey are husbands, while a few are people who are not in family.

In [30]:
```
sns.countplot(censusData['race'])
plt.xticks(rotation = 90)
plt.show()
```

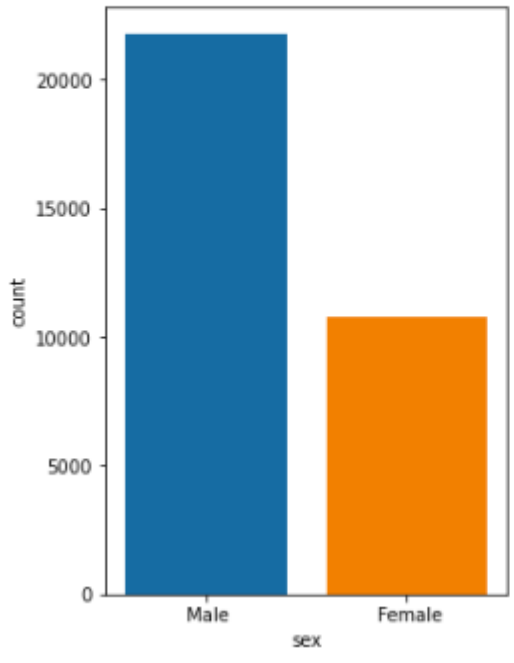
```
/home/ubuntu/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWar
ning:

Pass the following variable as a keyword arg: x. From version 0.12, the only valid posi
tional argument will be `data`, and passing other arguments without an explicit keyword
will result in an error or misinterpretation.
```



There's a huge number of white people in the data, while there are a considerable amount of black people. The others are in very few numbers.

```
In [31]: plt.figure(figsize=(4,6))
         sns.countplot(data=categorical_attributes, x='sex')
```

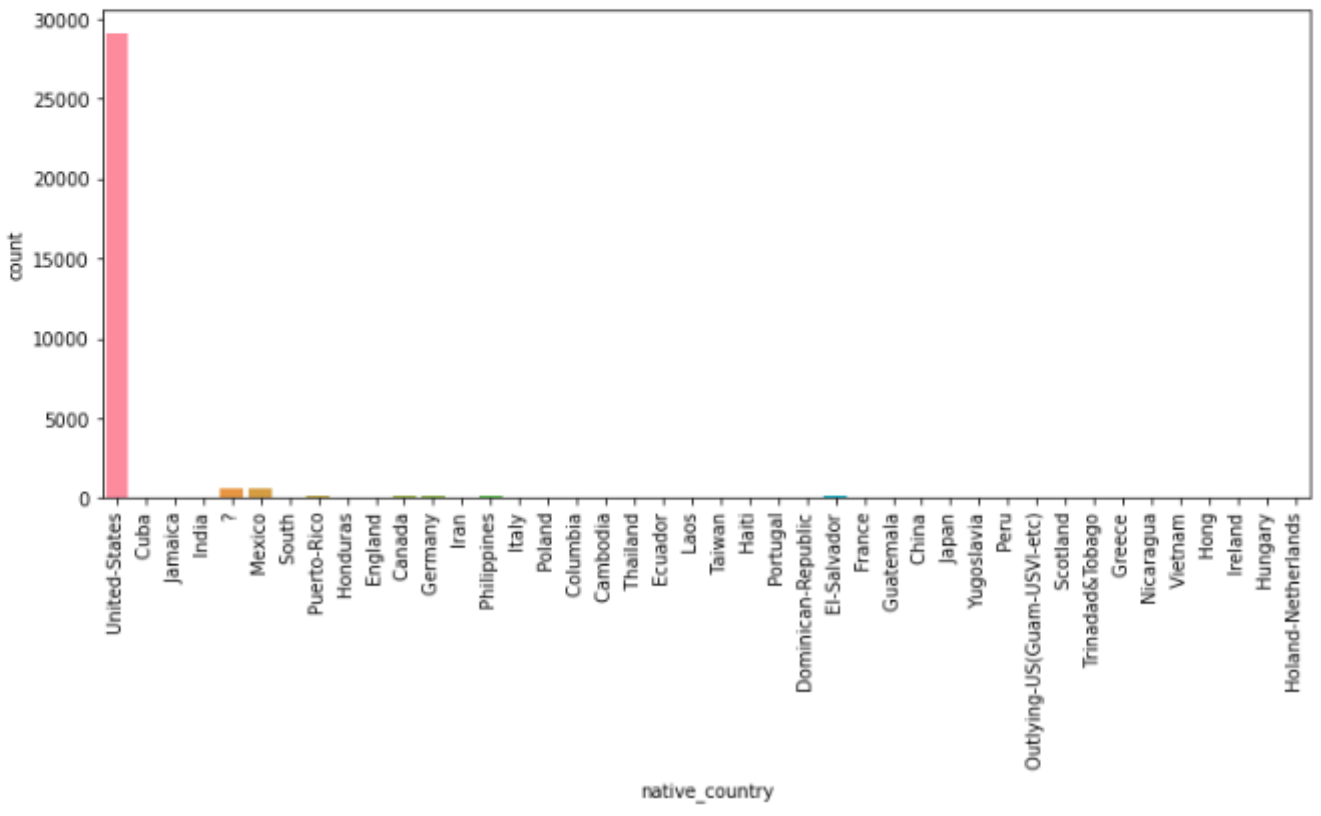
```
Out[31]: <AxesSubplot:xlabel='sex', ylabel='count'>
```



Majority of the population in the data are husbands, which is true according to the column relationship also, as the majority were husbands.

```
In [32]: plt.figure(figsize=(12,5))
         sns.countplot(censusData['native_country'])
         plt.xticks(rotation = 90)
         plt.show()
```

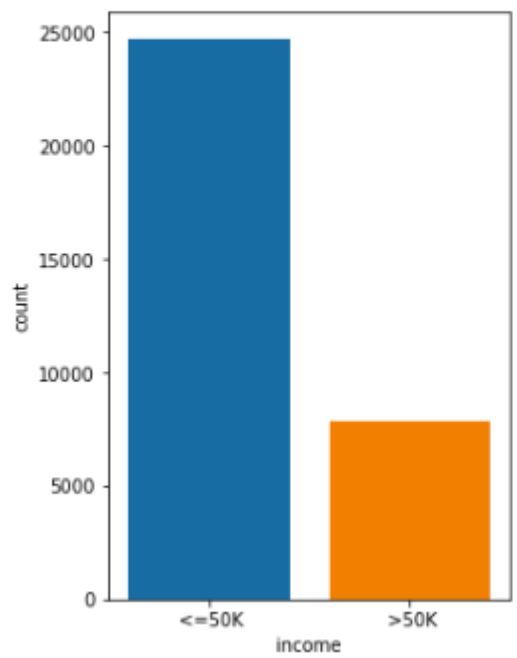
```
/home/ubuntu/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWar
ning:

Pass the following variable as a keyword arg: x. From version 0.12, the only valid posi
tional argument will be `data`, and passing other arguments without an explicit keyword
will result in an error or misinterpretation.
```



Majority of the people are from United States, however there are a few from Mexico, and very few from India, Jamaica, Cuba, Germany, etc.

In [33]:
```
plt.figure(figsize=(4,6))
sns.countplot(data=categorical_attributes, x='income')
```

Out[33]: <AxesSubplot:xlabel='income', ylabel='count'>



Majority of the people have an income of less than 50k according to the data given, indicating that the data is somewhat skewed.

INFERENCES:

There are 32537 entries in the dataframe. Only 3 attributes, namely workclass, occupation, native_country have some missing/null values as there is no data regarding them or people refused to give that specific information.
There are a lot of ways in which we can handle missing data. As all the three are categorical data, we can deal with it in the following ways:

1. Drop these rows, so that we don't have any missing values in our data.
2. Choose the median of values once we have transformed categorical values into corresponding numerical representations, as median is not affected by the range/spread of data unlike mean.
3. We could also use a classifier with non-missing values to predict the missing values, and then use them to build our final model.
4. There are certain classifiers like XGBoost which automatically handle missing data

We could drop these values as there are not in a huge number here(~10% of the data).

Incase of native_country, we could consider USA as the native country as more than 90% of the people in the dataset are from USA.

Missing values:

```
In [34]: censusData.workclass.unique()

Out[34]: array([' State-gov', ' Self-emp-not-inc', ' Private', ' Federal-gov',
                ' Local-gov', ' ?', ' Self-emp-inc', ' Without-pay',
                ' Never-worked'], dtype=object)

In [35]: count = 0
         for i in range(len(categorical_attributes['native_country'])):
             if(categorical_attributes['native_country'][i]==' ?'):
                 count+=1
         print("Missing values in native_country:", count)

         Missing values in native_country: 582

In [36]: censusData.occupation.unique()

Out[36]: array([' Adm-clerical', ' Exec-managerial', ' Handlers-cleaners',
                ' Prof-specialty', ' Other-service', ' Sales', ' Craft-repair',
                ' Transport-moving', ' Farming-fishing', ' Machine-op-inspct',
                ' Tech-support', ' ?', ' Protective-serv', ' Armed-Forces',
                ' Priv-house-serv'], dtype=object)

In [37]: count = 0
         for i in range(len(categorical_attributes['occupation'])):
             if(categorical_attributes['occupation'][i]==' ?'):
                 count+=1
         print("Missing values in occupation:", count)

         Missing values in occupation: 1843

In [38]: censusData.workclass.unique()

Out[38]: array([' State-gov', ' Self-emp-not-inc', ' Private', ' Federal-gov',
                ' Local-gov', ' ?', ' Self-emp-inc', ' Without-pay',
                ' Never-worked'], dtype=object)

In [39]: count = 0
         for i in range(len(categorical_attributes['workclass'])):
             if(categorical_attributes['workclass'][i]==' ?'):
                 count+=1
         print("Missing values in workclass:", count)

         Missing values in workclass: 1836

In [40]: print("Total records: ", len(censusData))

         Total records:  32537
```

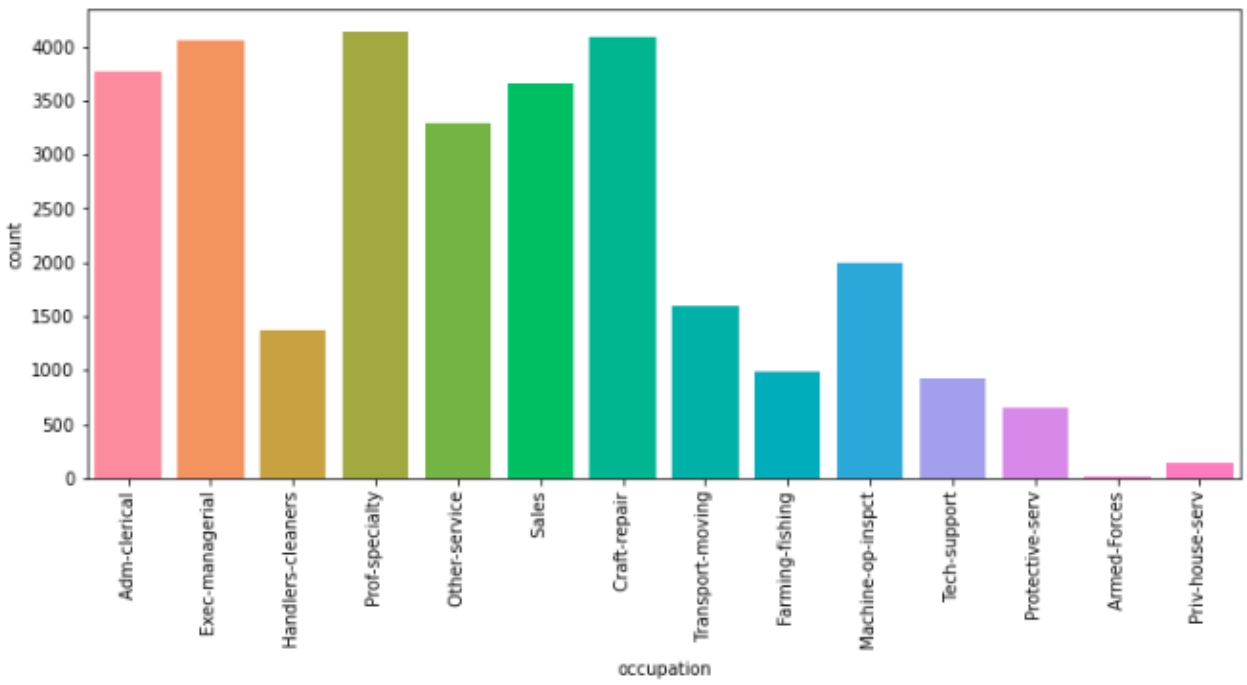
As we can see, the the missing values (native_country, occupation and work-class) form a small percent of the entire dataset (1.7%, 5.6% and 5.6% respectively)

So we can eliminate the records having missing values.

In [42]:
```
censusData = censusData.drop(censusData[censusData['occupation']==' ?'].index)
```

In [43]:
```
plt.figure(figsize=(12,5))
sns.countplot(censusData['occupation'])
plt.xticks(rotation = 90)
plt.show()
```

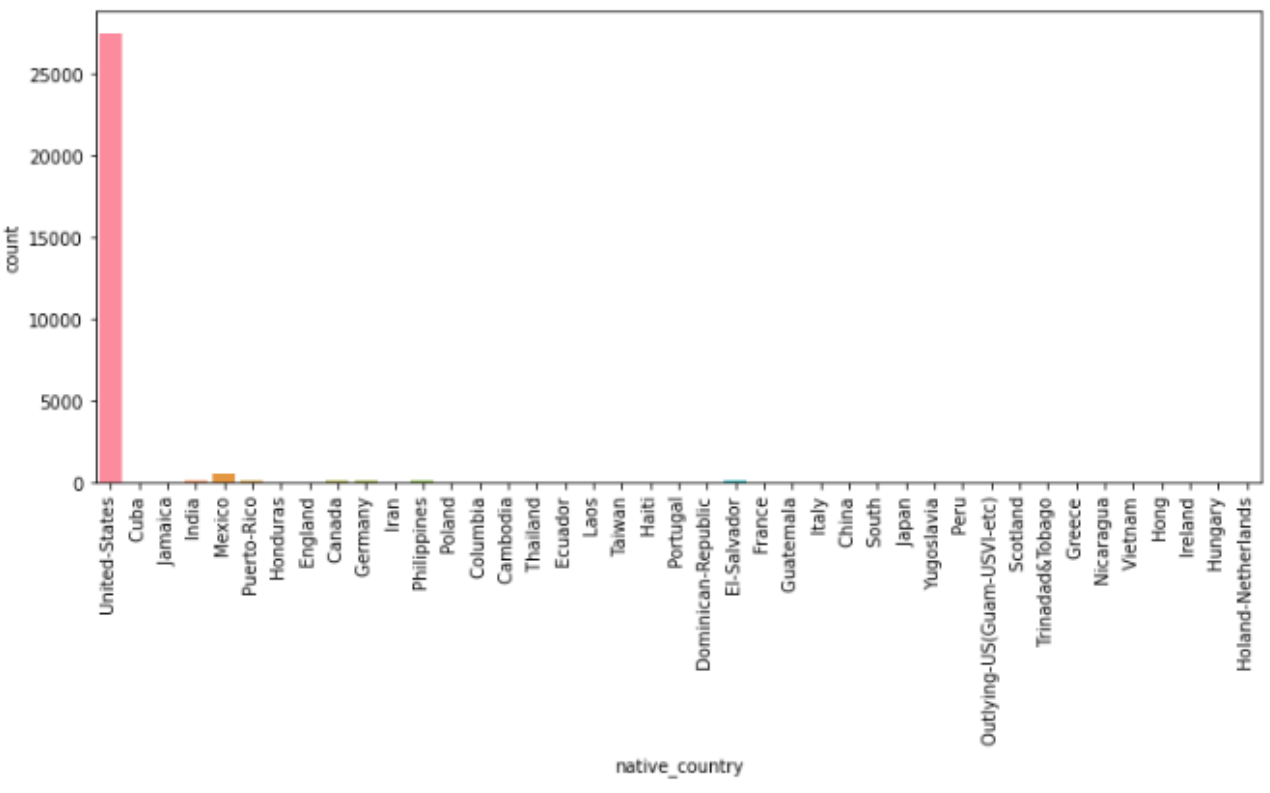
```
/home/ubuntu/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
```



In [44]:
```
censusData = censusData.drop(censusData[censusData['native_country']==' ?'].index)
```

In [45]:
```
plt.figure(figsize=(12,5))
sns.countplot(censusData['native_country'])
plt.xticks(rotation = 90)
plt.show()
```

```
/home/ubuntu/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
```



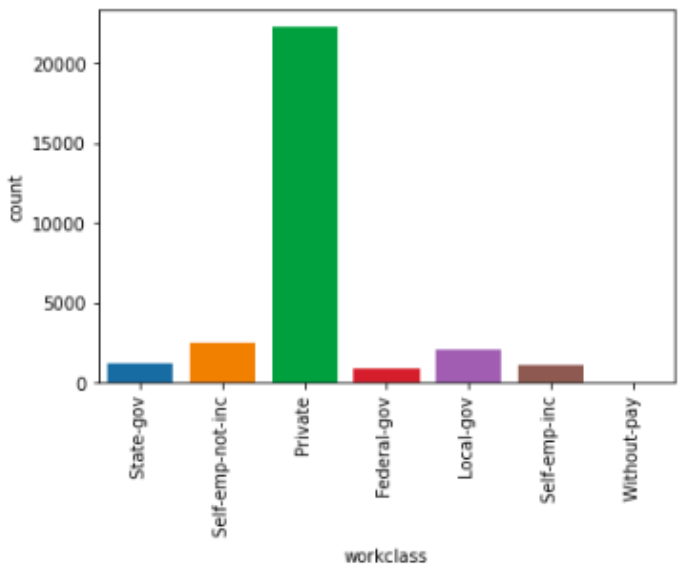
In [46]:
```
censusData = censusData.drop(censusData[censusData['workclass']==' ?'].index)
```

In [47]:
```
censusData.workclass.unique()
```

Out[47]:
```
array([' State-gov', ' Self-emp-not-inc', ' Private', ' Federal-gov',
       ' Local-gov', ' Self-emp-inc', ' Without-pay'], dtype=object)
```

In [48]:
```
sns.countplot(censusData['workclass'])
plt.xticks(rotation = 90)
plt.show()
```

```
/home/ubuntu/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning:

Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
```



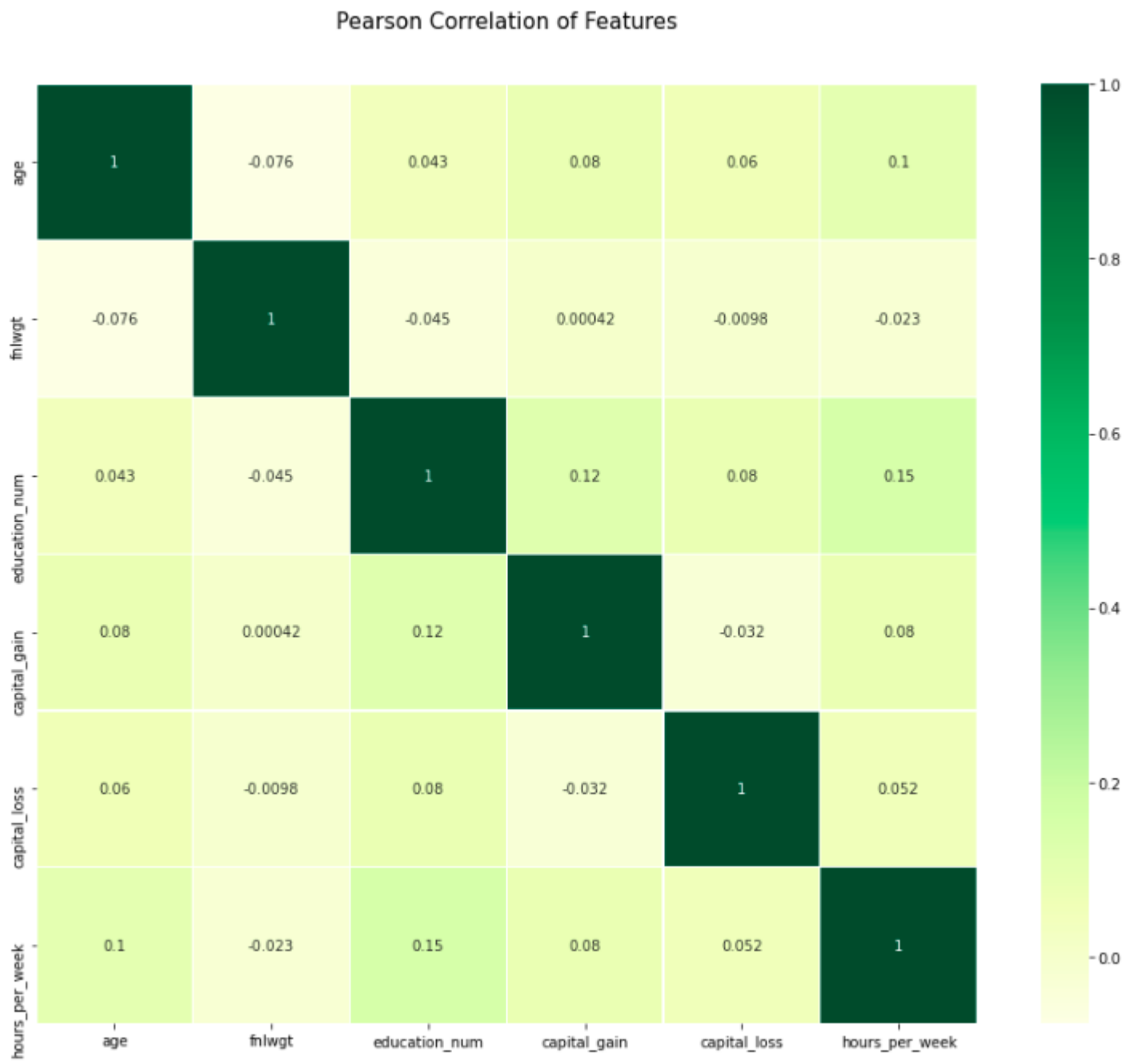
Correlation

```
sns.pairplot(censusData, height=3, diag_kind = 'kde', hue='income')
```

```
Out[49]: <seaborn.axisgrid.PairGrid at 0x7fa56618a220>
```


```
# Compute the correlation matrix
corr = censusData.corr()

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(16, 12))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap
_ = sns.heatmap(corr, cmap="YlGn", square=True, ax = ax, annot=True, linewidth = 0.1)

plt.title('Pearson Correlation of Features', y=1.05, size=15)
```

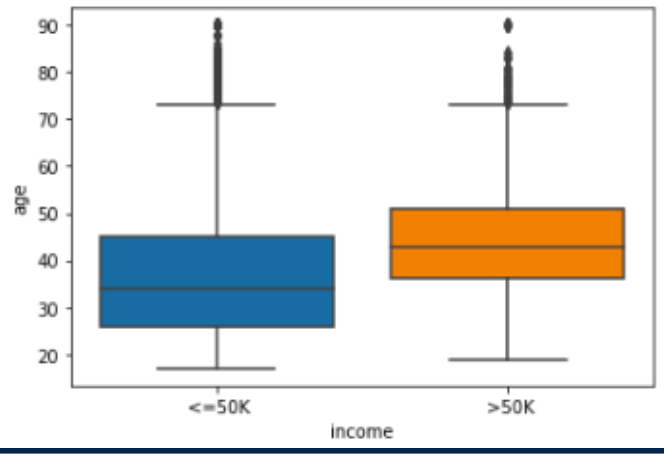
Out[50]: Text(0.5, 1.05, 'Pearson Correlation of Features')



There isn't a great correlation between the numerical features.

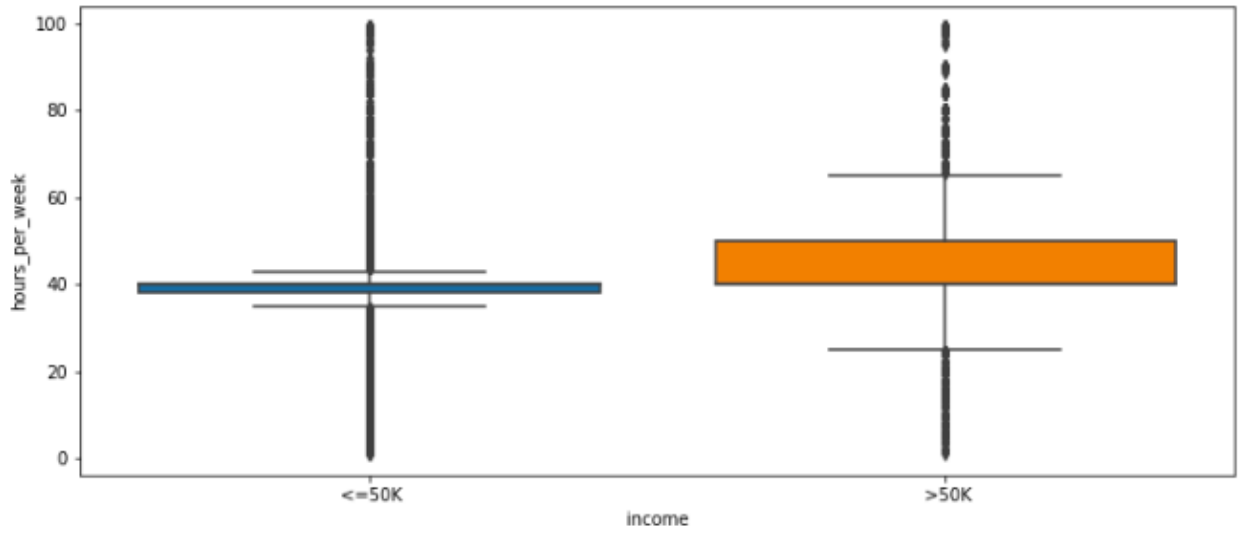
INFERENCES:
1. The fnlwgt feature looks useless.
2. There is some correlation between age, hours per week and education-num according to both pair plot and correlation heatmap.
3. Capital gain and Capital loss don't anticorrelate a lot, which says people can invest only if they have money.

In [51]:
```
sns.boxplot(x='income',y='age',data=censusData)
plt.show()
```



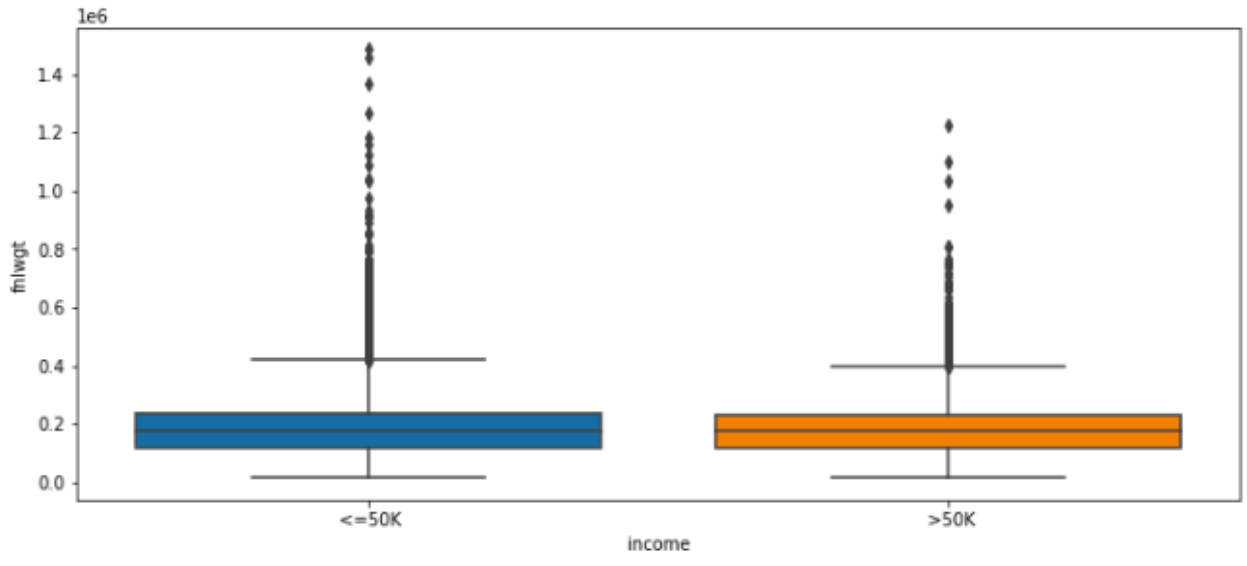
Avg people's age is 44 (approx) for income > 50k which is more compared to income <= 50k.

In [52]:
```
plt.figure(figsize=(12,5))
sns.boxplot(x='income',y='hours_per_week',data=censusData)
plt.show()
```



Income >50k have longer avg working hrs than <50k .
Working hr range is also higher for income >50k.

In [53]:
```
plt.figure(figsize=(12,5))
sns.boxplot(x='income',y='fnlwgt',data=censusData)
plt.show()
```



Weight has no significance on income.

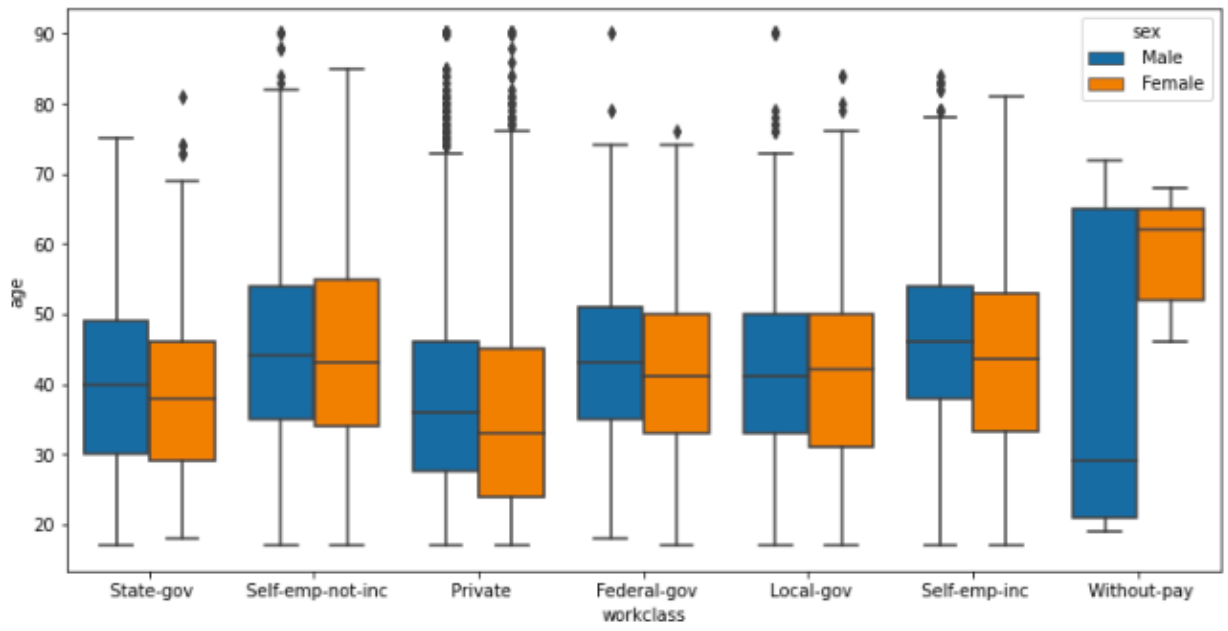
In [54]:
```
# Cross tabulation between work-class and sex

pd.crosstab(censusData['workclass'],censusData['sex'], margins=True)
```

Out[54]:

sex workclass	Female	Male	All
Federal-gov	309	634	943
Local-gov	824	1243	2067
Private	7633	14631	22264
Self-emp-inc	126	948	1074
Self-emp-not-inc	392	2106	2498
State-gov	484	795	1279
Without-pay	5	9	14
All	9773	20366	30139

In [55]:
```
# Box plot between work-class and age for different sex

plt.figure(figsize=(12,6))
sns.boxplot(x="workclass",y="age", hue="sex", data=censusData)
```

Out[55]: <AxesSubplot:xlabel='workclass', ylabel='age'>



The cross tabulation values show that more number of men are self-employed rather than women. The box plot gives a few interesting observations:

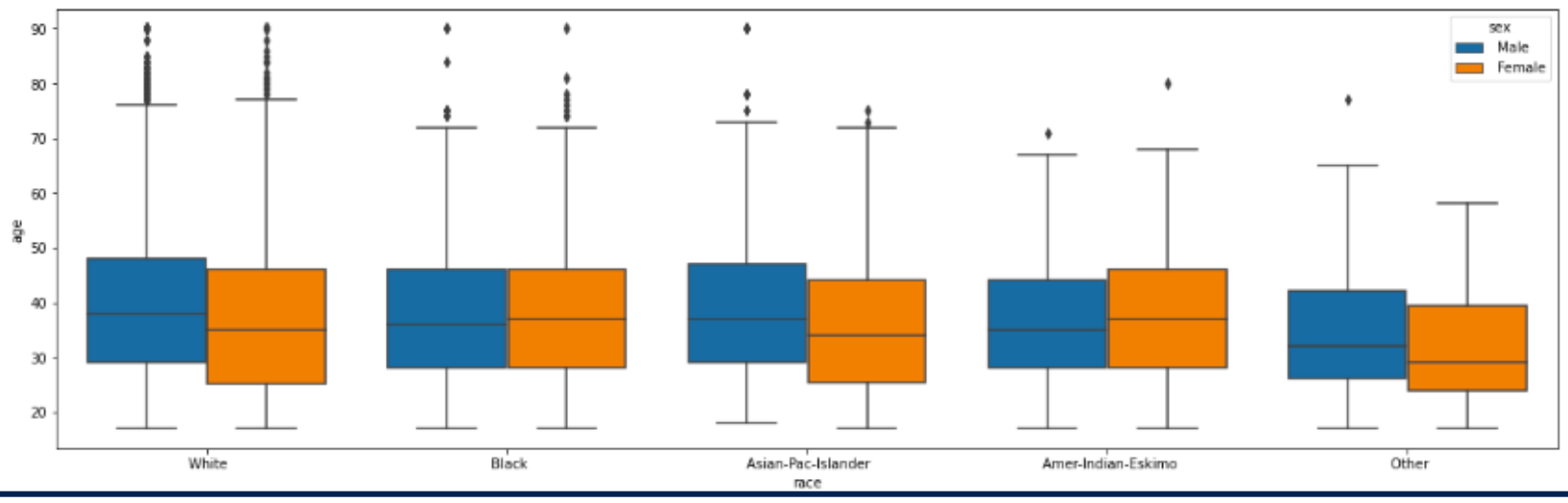
1. People who have never worked have a very young age. Women have a slightly higher median age than men for this working-class.

2. Men who work without pay have a median age of around 20-30, whereas women who work without pay have a median age of around 60-65.

3. Most of the working class have a median age of around 40-50.

In [56]:
```
# Box plot between race and age for different sex

plt.figure(figsize=(20,6))
sns.boxplot(x="race",y="age", hue="sex", data=censusData)
```

Out[56]: <AxesSubplot:xlabel='race', ylabel='age'>



People from "other" race have a younger median age than the rest of the races.

```
In [57]: # Cross tabulation between race and sex

         pd.crosstab(censusData['race'],censusData['sex'], margins=True)
```

Out[57]:

sex race	Female	Male	All
Amer-Indian-Eskimo	107	179	286
Asian-Pac-Islander	294	600	894
Black	1399	1417	2816
Other	87	144	231
White	7886	18026	25912
All	9773	20366	30139

This above cross-tabulation gives a distribution of various races and sex. The number of males and females are very similar in all races except whites.

```
In [58]: # Cross tabulation between native-country and sex

         pd.crosstab(censusData['native_country'],censusData['sex'], margins=True)
```

Out[58]:

sex native_country	Female	Male	All
Cambodia	2	16	18
Canada	34	73	107
China	18	50	68
Columbia	23	33	56
Cuba	38	54	92
Dominican-Republic	34	33	67
Ecuador	9	18	27
El-Salvador	33	67	100
England	30	56	86
France	10	17	27
Germany	54	74	128
Greece	5	24	29
Guatemala	18	43	61
Haiti	18	24	42
Holand-Netherlands	1	0	1
Honduras	6	6	12
Hong	5	14	19
Hungary	6	7	13
India	11	89	100
Iran	7	35	42
Ireland	7	17	24
Italy	18	50	68
Jamaica	42	38	80
Japan	18	41	59
Laos	7	10	17
Mexico	127	479	606
Nicaragua	12	21	33
Outlying-US(Guam-USVI-etc)	7	7	14
Peru	14	16	30
Philippines	72	116	188
Poland	17	39	56
Portugal	12	22	34

Puerto-Rico	48	61	109
Scotland	4	7	11
South	25	46	71
Taiwan	10	32	42
Thailand	10	7	17
Trinadad&Tobago	10	8	18
United-States	8926	18561	27487
Vietnam	22	42	64
Yugoslavia	3	13	16
All	9773	20366	30139

We can see that the cross tabulation values don't give much interesting results when the number of categories are huge. We could reduce/combine the categories, like native-country to Non-USA, USA, group ages to specific ranges, etc.

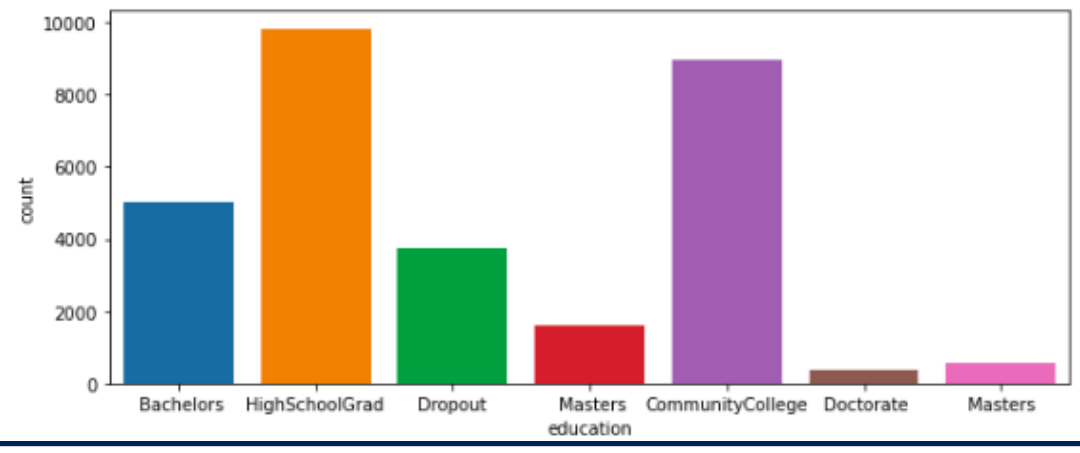
```
# Grouping Education
censusData['education'].replace(' Preschool', 'Dropout',inplace=True)
censusData['education'].replace(' 10th', 'Dropout',inplace=True)
censusData['education'].replace(' 11th', 'Dropout',inplace=True)
censusData['education'].replace(' 12th', 'Dropout',inplace=True)
censusData['education'].replace(' 1st-4th', 'Dropout',inplace=True)
censusData['education'].replace(' 5th-6th', 'Dropout',inplace=True)
censusData['education'].replace(' 7th-8th', 'Dropout',inplace=True)
censusData['education'].replace(' 9th', 'Dropout',inplace=True)
censusData['education'].replace(' HS-Grad', 'HighSchoolGrad',inplace=True)
censusData['education'].replace(' HS-grad', 'HighSchoolGrad',inplace=True)
censusData['education'].replace(' Some-college', 'CommunityCollege',inplace=True)
censusData['education'].replace(' Assoc-acdm', 'CommunityCollege',inplace=True)
censusData['education'].replace(' Assoc-voc', 'CommunityCollege',inplace=True)
censusData['education'].replace(' Prof-school', 'Masters',inplace=True)
censusData['education'].replace(' 9th', 'Below-HS', inplace=True)
censusData['education'].replace(' 1st-4th', 'Below-HS', inplace=True)
```

```
In [60]: censusData.education.unique()
```

```
Out[60]: array([' Bachelors', 'HighSchoolGrad', 'Dropout', ' Masters',
       'CommunityCollege', ' Doctorate', 'Masters'], dtype=object)
```

```
In [61]: plt.figure(figsize=(10,4))
sns.countplot(data=censusData, x='education')
```

```
Out[61]: <AxesSubplot:xlabel='education', ylabel='count'>
```




```
In [62]: # Cross tabulation between education and sex

pd.crosstab(censusData['education'],censusData['sex'], margins=True)
```

Out[62]:

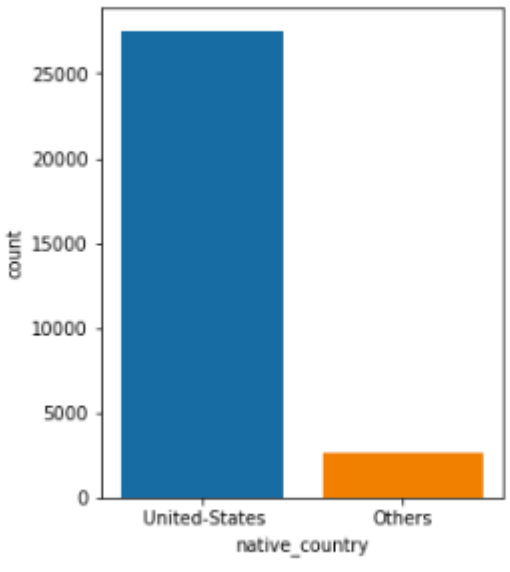
education \ sex	Female	Male	All
Bachelors	1521	3521	5042
Doctorate	81	294	375
Masters	509	1117	1626
CommunityCollege	3353	5631	8984
Dropout	1118	2618	3736
HighSchoolGrad	3104	6730	9834
Masters	87	455	542
All	9773	20366	30139

This gives a better idea about the relation between sex and education level. More men do Masters and Doctorate compared to women.

Similarly we group native_country

```
# Grouping native_country
censusData['native_country'] = censusData['native_country'].apply(lambda el: " United-States" if el == " United-States"
else "Others")

# Plotting count plot for native-country
fig = plt.figure(figsize=(4,5))
sns.countplot(x="native_country", data=censusData)
```

```
Out[63]: <AxesSubplot:xlabel='native_country', ylabel='count'>
```



```
In [64]:
# Plotting cross tabulation values for native_country and sex
pd.crosstab(censusData['native_country'],censusData['sex'], margins=True)
```

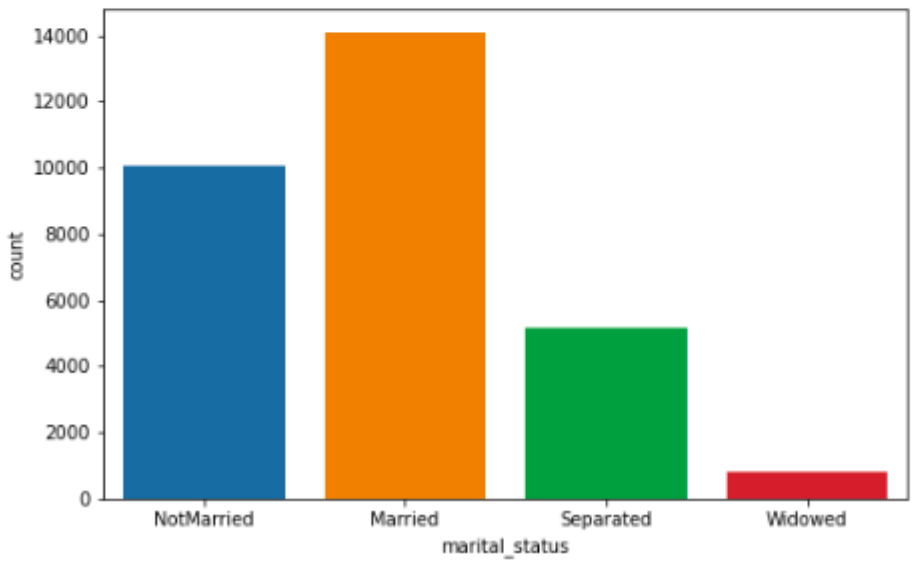
Out[64]:

native_country	sex	Female	Male	All
United-States		8926	18561	27487
Others		847	1805	2652
All		9773	20366	30139

The ratio of men and women from US as well as other natives are more or less similar, however, the number of US nationals are considerably more in the dataset.

```
In [65]:
# Grouping marital status
censusData['marital_status'].replace(' Never-married', 'NotMarried',inplace=True)
censusData['marital_status'].replace([' Married-AF-spouse'], 'Married',inplace=True)
censusData['marital_status'].replace([' Married-civ-spouse'], 'Married',inplace=True)
censusData['marital_status'].replace([' Married-spouse-absent'], 'NotMarried',inplace=True)
censusData['marital_status'].replace([' Separated'], 'Separated',inplace=True)
censusData['marital_status'].replace([' Divorced'], 'Separated',inplace=True)
censusData['marital_status'].replace([' Widowed'], 'Widowed',inplace=True)

#Plotting count plot for marital status
fig = plt.figure(figsize=(8,5))
sns.countplot(x="marital_status", data=censusData)
```

Out[65]: <AxesSubplot:xlabel='marital_status', ylabel='count'>

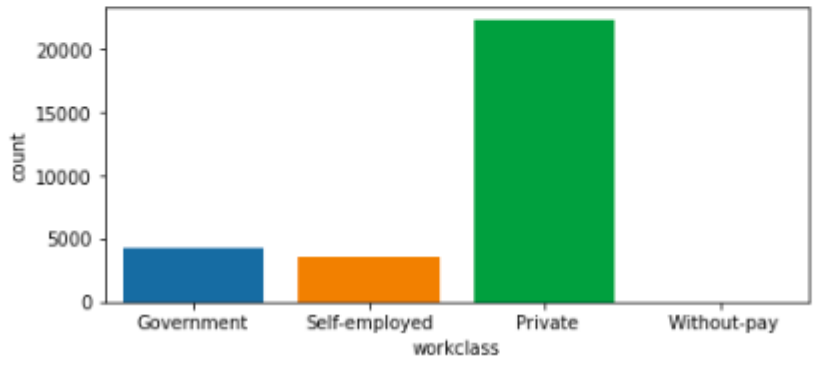



```
In [66]:
# Plotting cross tabulation values for marital status and sex
pd.crosstab(censusData['marital_status'],censusData['sex'], margins=True)
```

Out[66]:

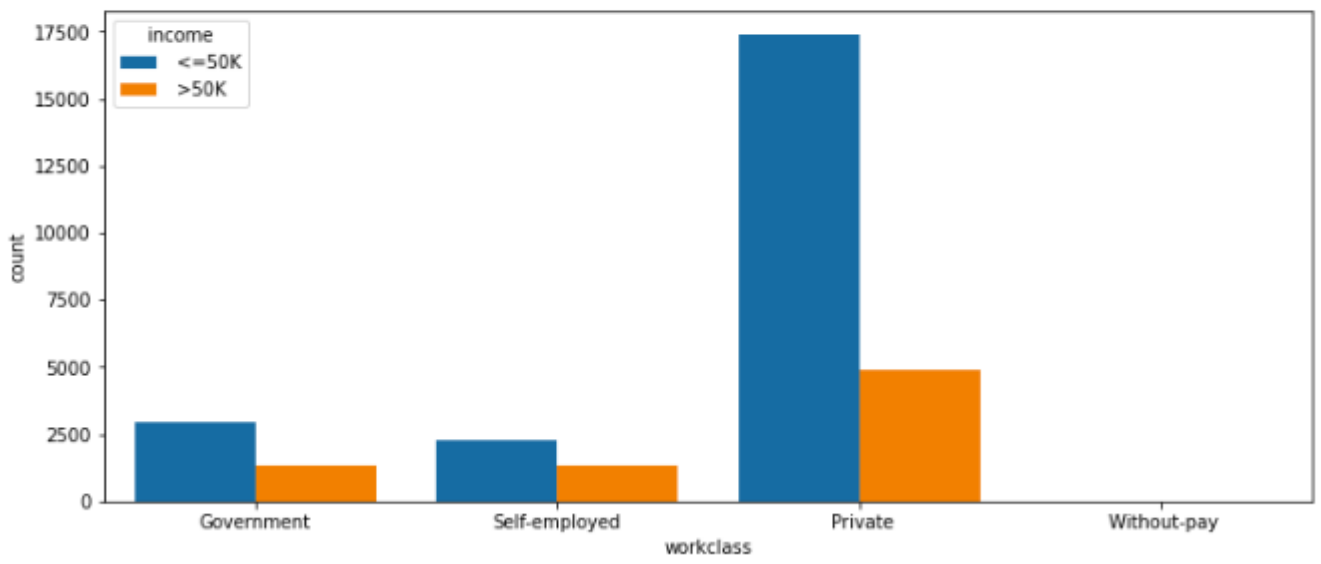
marital_status	sex	Female	Male	All
Married		1492	12588	14080
NotMarried		4494	5587	10081
Separated		3101	2050	5151
Widowed		686	141	827
All		9773	20366	30139

```
In [68]: censusData['workclass'].replace(' Self-emp-not-inc', 'Self-employed', inplace=True)
         censusData['workclass'].replace(' Self-emp-inc', 'Self-employed', inplace=True)
         censusData['workclass'].replace(' State-gov', 'Government', inplace=True)
         censusData['workclass'].replace(' Local-gov', 'Government', inplace=True)
         censusData['workclass'].replace(' Federal-gov', 'Government', inplace=True)

In [69]: plt.figure(figsize=(7,3))
         sns.countplot(data=censusData, x='workclass')

Out[69]: <AxesSubplot:xlabel='workclass', ylabel='count'>
```



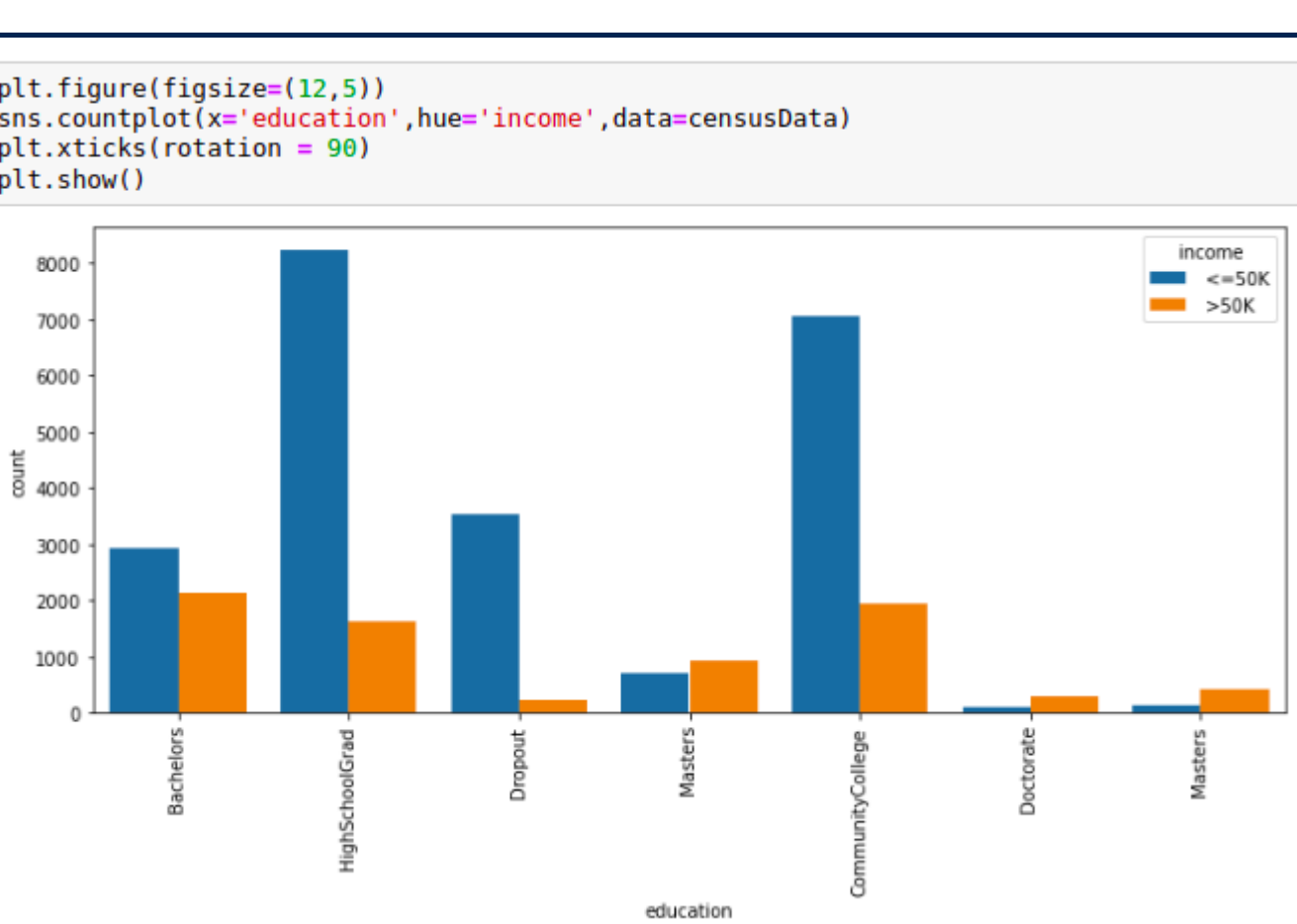
Categorical v/s Categorical

```
In [70]: plt.figure(figsize=(12,5))
         sns.countplot(x='workclass',hue='income',data=censusData)
         plt.show()
```



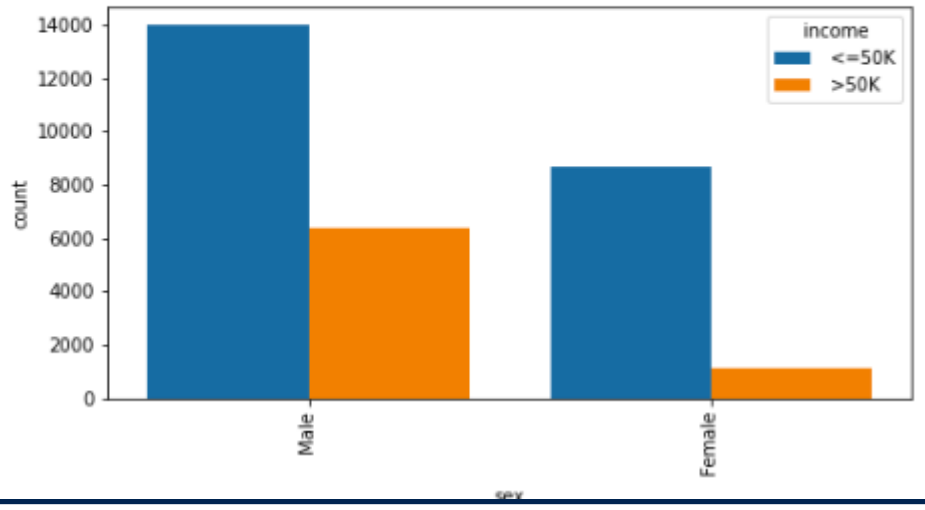
Private working class have highest no. people >50k

```
In [71]: plt.figure(figsize=(12,5))
         sns.countplot(x='education',hue='income',data=censusData)
         plt.xticks(rotation = 90)
         plt.show()
```



Higher education have better chance of earning >50k

```
In [72]:  plt.figure(figsize=(8,4))
          sns.countplot(x='sex',hue='income',data=censusData)
          plt.xticks(rotation = 90)
          plt.show()
```

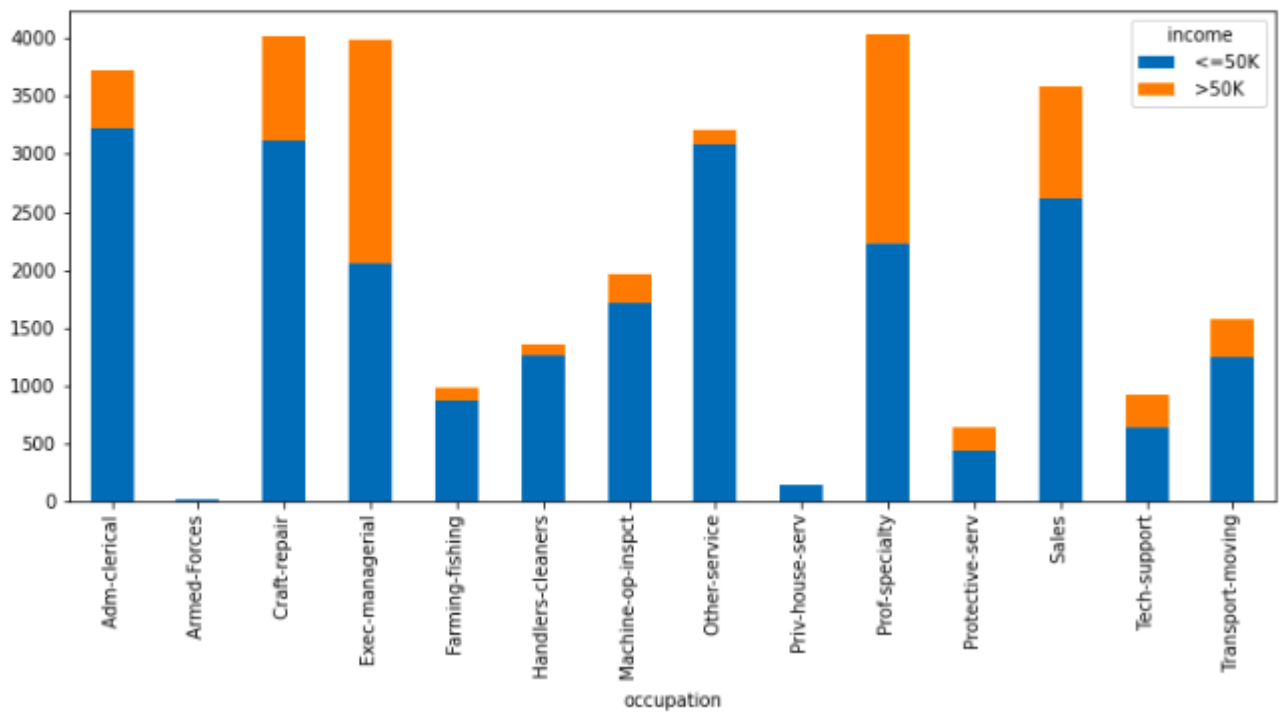


Females are more likely to be earning <= 50k.

Stacked bar for categorical v/s categorical

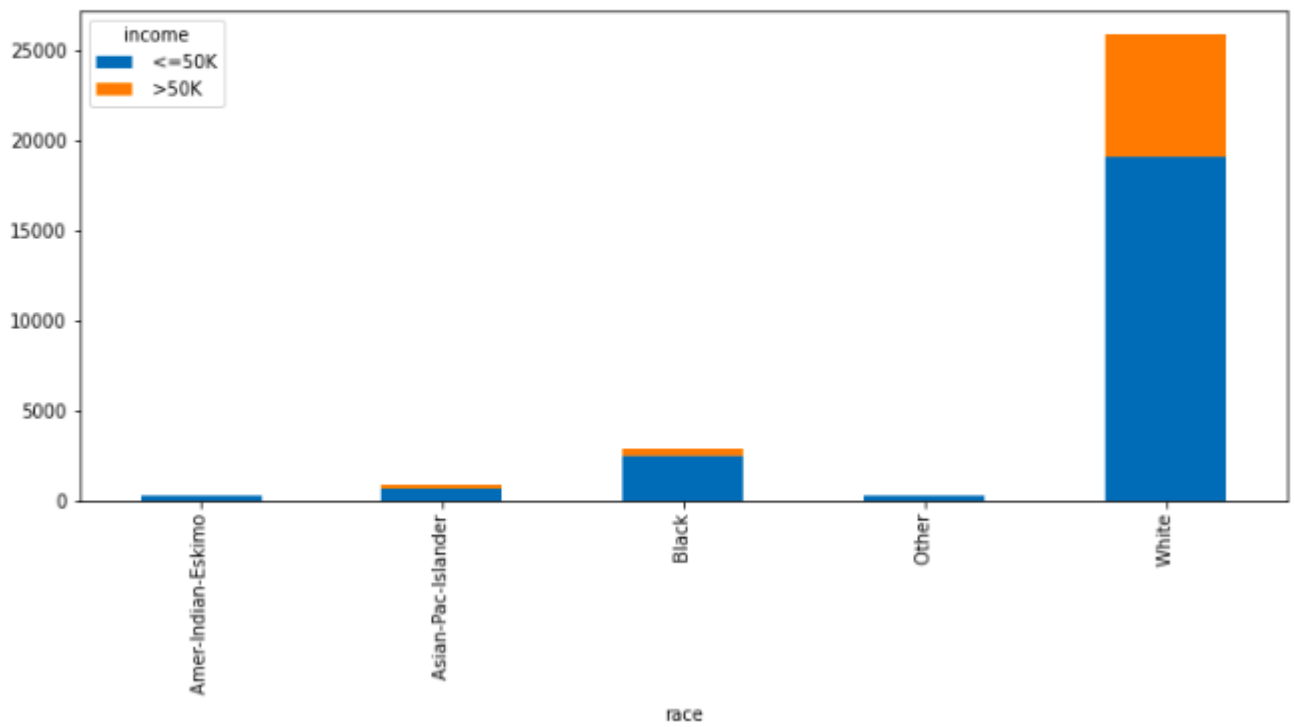
In every occupation, people who earn less than 50k is greater than people who earn >50k.

```
In [73]:  dd = pd.crosstab(censusData['occupation'],censusData['income'])
          dd.plot.bar(stacked=True,figsize=(12,5))
          plt.xticks(rotation = 90)
          plt.show()
```



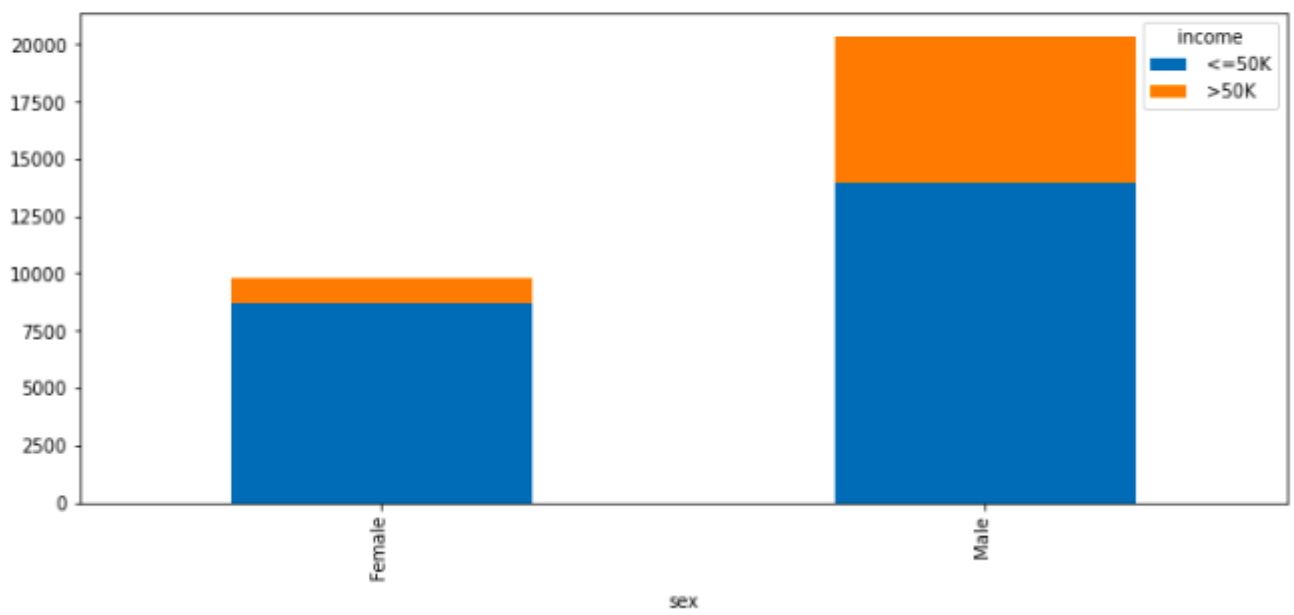
Exec-managerial and Prof-specialty stand out as having very high percentages of individuals making over 50,000. In addition, the percentages for Farming-fishing, Other-service and Handlers-cleaners are significantly lower than the rest of the distribution.

In [74]:
```
dd = pd.crosstab(censusData['race'],censusData['income'])
dd.plot.bar(stacked=True,figsize=(12,5))
plt.xticks(rotation = 90)
plt.show()
```



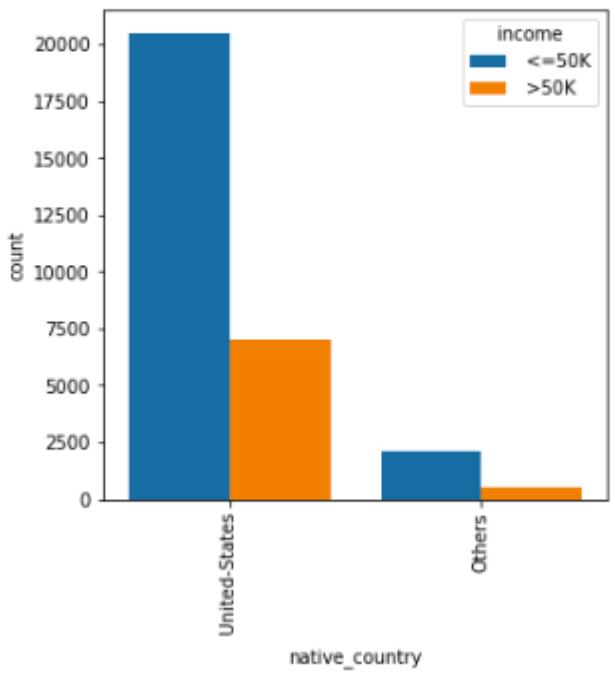
Whites and Asians have a larger percentage of entries greater than $50,000 than the rest of the races. However, the sample size of Whites in the dataset is disproportionately large in comparison to all other races. The second most represented group is Blacks with less than 5000 entries.

In [75]:
```
dd = pd.crosstab(censusData['sex'],censusData['income'])
dd.plot.bar(stacked=True,figsize=(12,5))
plt.xticks(rotation = 90)
plt.show()
```



The percentage of males who make greater than
$50,000 is much greater than the percentage of females that make the same amount.

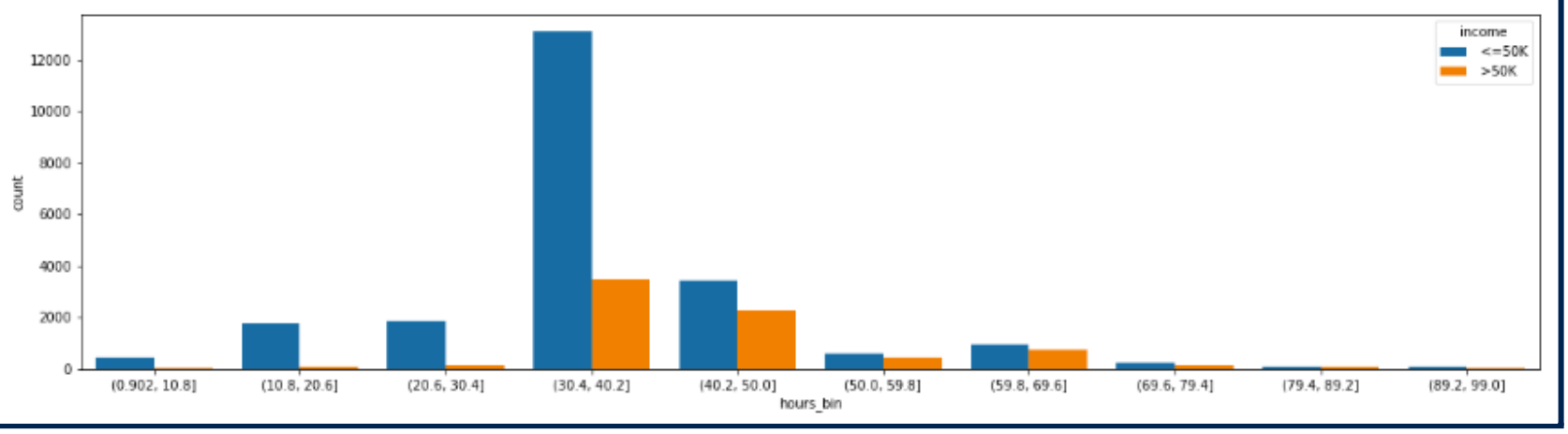
In [76]:
```
plt.figure(figsize=(5,5))
sns.countplot(x='native_country',hue='income',data=censusData)
plt.xticks(rotation = 90)
plt.show()
```



The ratio of people with native-country as US earning more than 50k is lesser than the ratio for people with other native-origins. This result is kind of surprising.

In [77]:
```
#Dividing hours per week into range values
censusData['hours_bin'] = pd.cut(censusData['hours_per_week'], 10)

# Plotting count plot for hours per week range and income
fig = plt.figure(figsize=(20,5))
sns.countplot(x="hours_bin", hue="income", data=censusData)
```

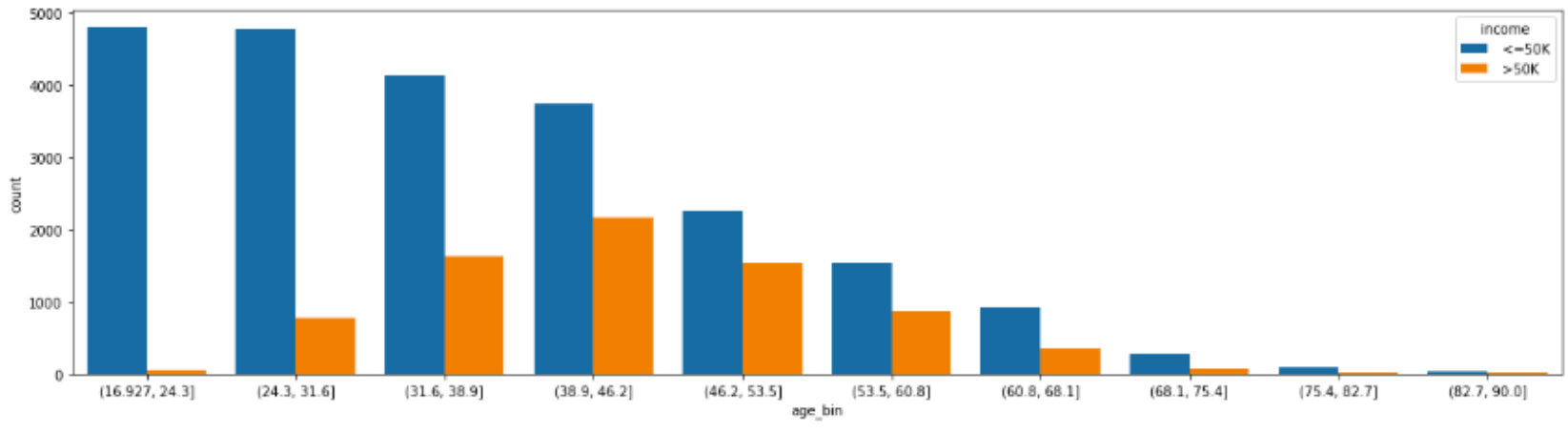
Out[77]: <AxesSubplot:xlabel='hours_bin', ylabel='count'>



People on average work 30-40 hours a week, but if the person works more, the probability of the person getting paid >50k increases.

In [78]:
```
# Dividing age into ranges
censusData['age_bin'] = pd.cut(censusData['age'], 10)

# Plotting count plot for age and income
fig = plt.figure(figsize=(20,5))
sns.countplot(x="age_bin", hue="income", data=censusData)
```

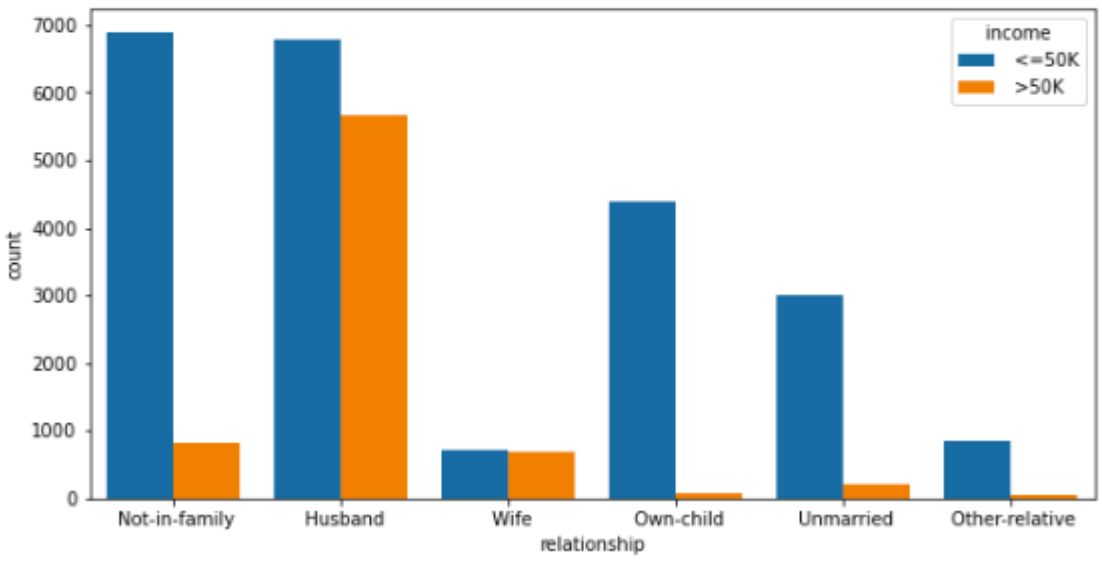
Out[78]: <AxesSubplot:xlabel='age_bin', ylabel='count'>



The above plot shows that experience matters to earn more income. More experienced people earn more than less experience people.
The ratio of males earning more than 50k is greater than the ratio of females earning more than 50k.

```
In [79]: # Plotting count plot for relationship and income
         fig = plt.figure(figsize=(10,5))
         sns.countplot(x="relationship", hue="income", data=censusData)
```

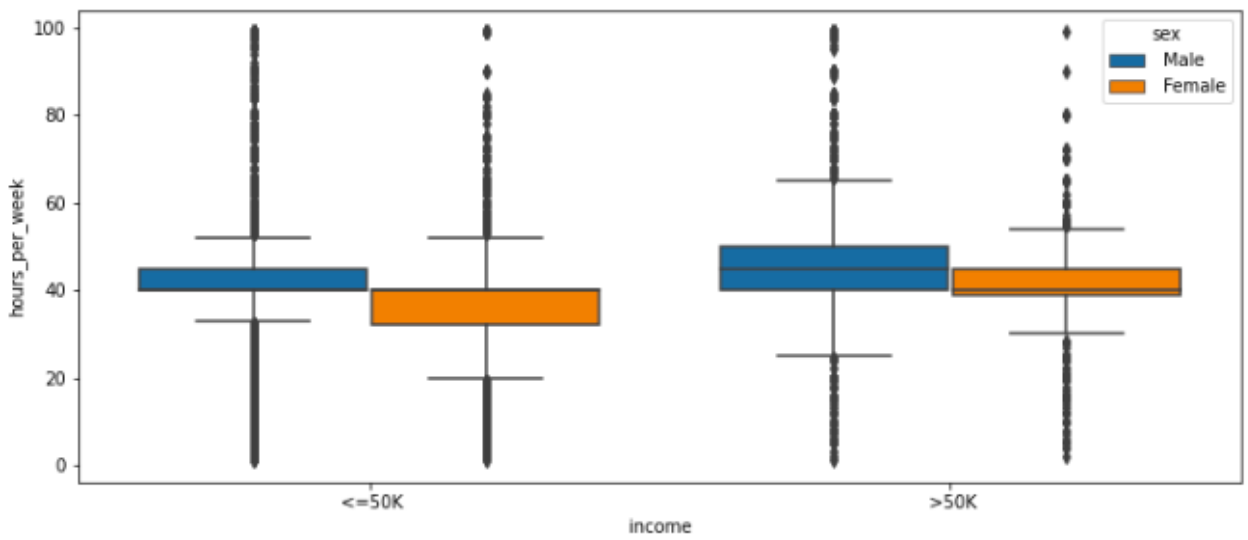
```
Out[79]: <AxesSubplot:xlabel='relationship', ylabel='count'>
```



Husbands and Wives have more probability to earn money >50k than others.

```
In [80]: plt.figure(figsize=(12,5))
         sns.boxplot(censusData['income'],censusData['hours_per_week'],hue=censusData['sex'])
         plt.show()
```

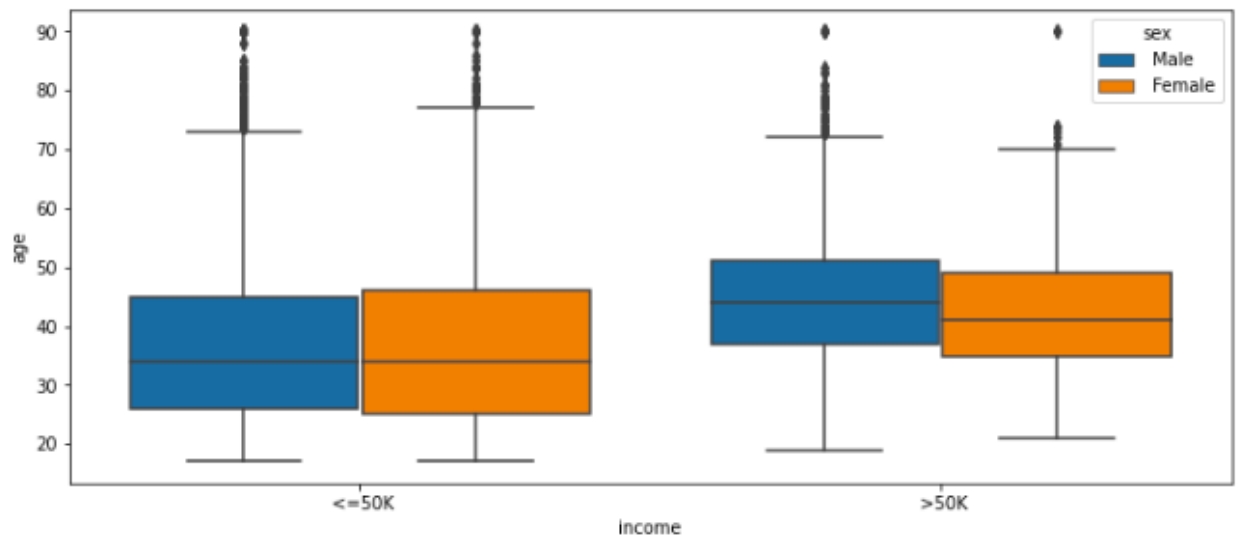
```
/home/ubuntu/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning:

Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will
be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretatio
n.
```



Average working hour of females is less compared to males.
However for income group >50k mens have more flexible working hrs compared to females.

In [81]:
```
plt.figure(figsize=(12,5))
sns.boxplot(censusData['income'],censusData['age'],hue=censusData['sex'])
plt.show()
```

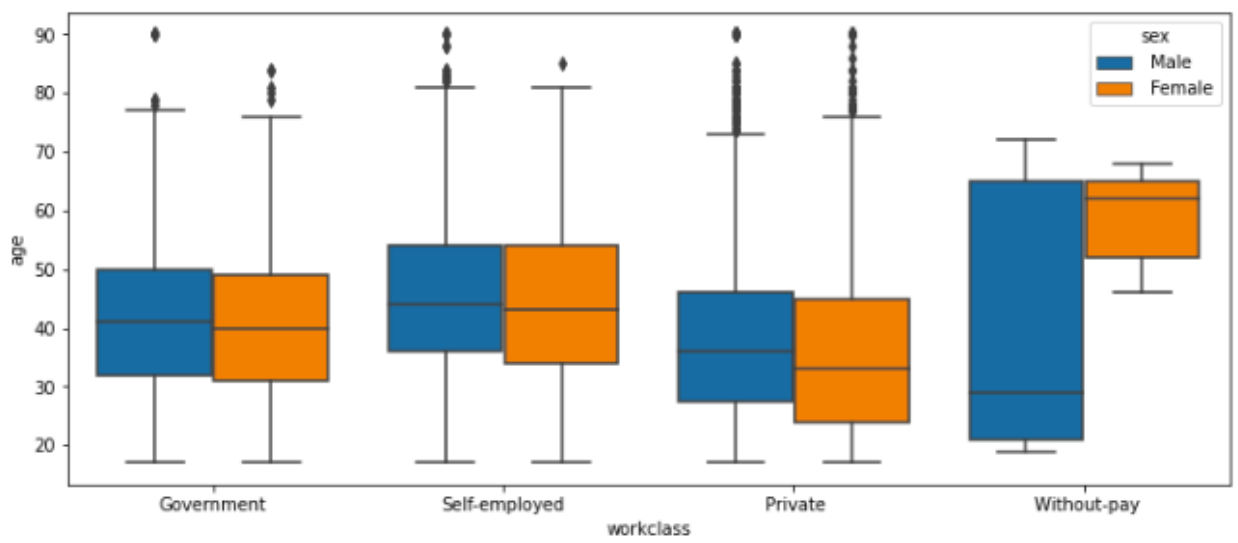
```
/home/ubuntu/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning:

Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will
be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretatio
n.
```



Average age of females is less compared to males for income group >50k.
However range is less for men compared to females for group <=50k.

In [82]:
```
plt.figure(figsize=(12,5))
sns.boxplot(censusData['workclass'],censusData['age'],hue=censusData['sex'])
plt.show()
```

```
/home/ubuntu/anaconda3/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarning:

Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will
be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretatio
n.
```



Average age of private and gov dept is less for income group >50k compared to group <=50k.

In [83]:
```
# Dropping outlier
i = censusData[censusData['capital_gain'] > 80000].index
censusData = censusData.drop(i)
# Dropping the column fnlwgt
censusData = censusData.drop(columns='fnlwgt')
```

In [84]:
```
censusData['capital_range'] = censusData['capital_gain'] - censusData['capital_loss']
censusData.head()
```

Out[84]:

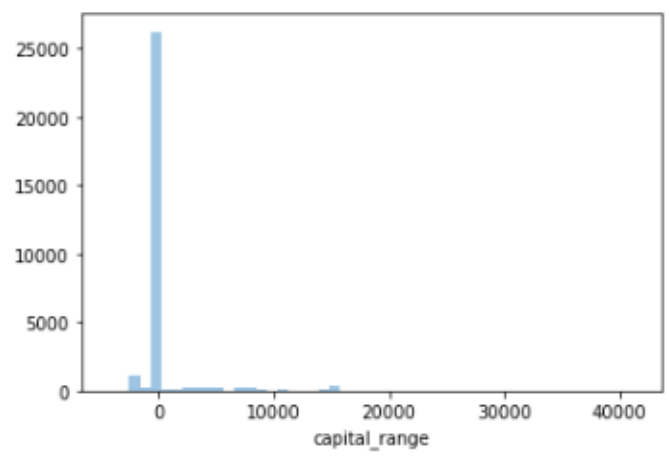
	age	workclass	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week
0	39	Government	Bachelors	13	NotMarried	Adm-clerical	Not-in-family	White	Male	2174	0	40
1	50	Self-employed	Bachelors	13	Married	Exec-managerial	Husband	White	Male	0	0	13
2	38	Private	HighSchoolGrad	9	Separated	Handlers-cleaners	Not-in-family	White	Male	0	0	40
3	53	Private	Dropout	7	Married	Handlers-cleaners	Husband	Black	Male	0	0	40
4	28	Private	Bachelors	13	Married	Prof-specialty	Wife	Black	Female	0	0	40

In [85]:
```
list(censusData.columns)
```

Out[85]:
```
['age',
 'workclass',
 'education',
 'education_num',
 'marital_status',
 'occupation',
 'relationship',
 'race',
 'sex',
 'capital_gain',
 'capital_loss',
 'hours_per_week',
 'native_country',
 'income',
 'hours_bin',
 'age_bin',
 'capital_range']
```

In [91]:
```
sns.distplot(censusData['capital_range'],kde=False)
plt.show()
```

```
/home/ubuntu/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2557: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
```



Capital range has same plot as capital gain/loss so we can replace capital gain and loss with one column.

In [88]:
```
censusData = censusData.drop(columns='capital_gain')

censusData = censusData.drop(columns='capital_loss')
```

In [89]:
```
censusData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 29991 entries, 0 to 32536
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype   
---  ------          --------------  -----   
 0   age             29991 non-null  int64   
 1   workclass       29991 non-null  object  
 2   education       29991 non-null  object  
 3   education_num   29991 non-null  int64   
 4   marital_status  29991 non-null  object  
 5   occupation      29991 non-null  object  
 6   relationship    29991 non-null  object  
 7   race            29991 non-null  object  
 8   sex             29991 non-null  object  
 9   hours_per_week  29991 non-null  int64   
 10  native_country  29991 non-null  object  
 11  income          29991 non-null  object  
 12  hours_bin       29991 non-null  category
 13  age_bin         29991 non-null  category
 14  capital_range   29991 non-null  int64   
dtypes: category(2), int64(4), object(9)
memory usage: 3.3+ MB
```

1. education_num - This attribute is same as education column, so we can use just one of those two.

2. fnlwgt - This attribute is useless, as it doesn't have any correlation with any attribute. We will remove this attribute.

3. capital-gain and capital-loss - We can combine capital gain and loss to have a column "Capital_range" rather than having two columns representing different values.

4. education - We could divide this into different classes like (Dropout, HighSchoolGrad, Community college, Bachelors, Masters, Doctorate). This gives us more solid categories, which could have more impact on prediction of the income.

5. marital-status - Similar to education, we could just have 4 classes like NotMarried, Married, Seperated and Widowed to categorize this attribute.

6. race - Remain as it is.

7. occupation - Remain as it is

8. working-class - Similar to education, we could just have 4 classes like Government, Private, Self-employed, Without_pay.

9. Sex - Remain as it is.

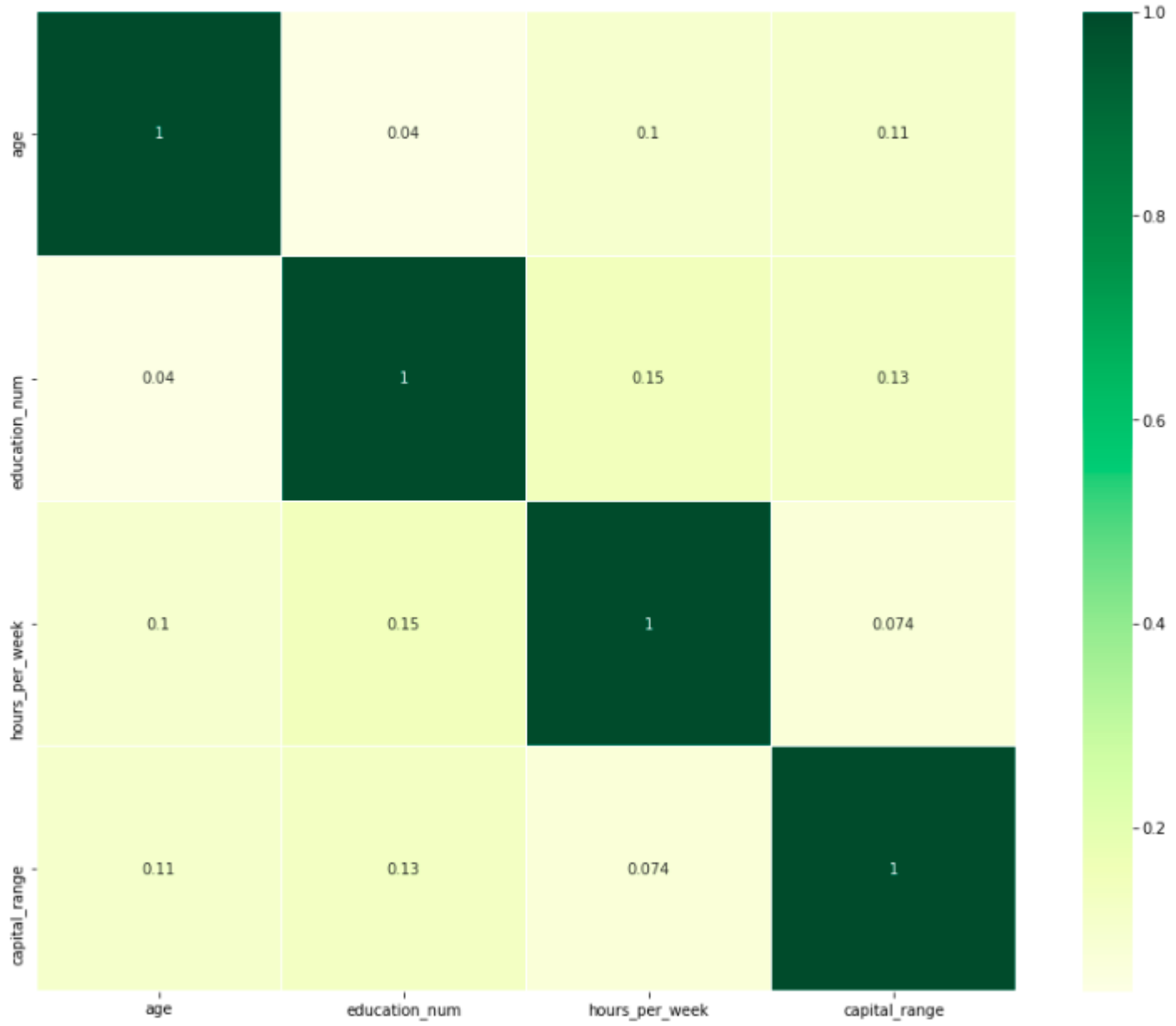
10. Hours-per-week - Remain as it is.

11. Income - Remain as it is.

12. Native-Country - Similar to education, we could just have 2 classes namely USA and others.

```
# Compute the correlation matrix
corr = censusData.corr()

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(16, 12))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap
_ = sns.heatmap(corr, cmap="YlGn", square=True, ax = ax, annot=True, linewidth = 0.1)

plt.title('Pearson Correlation of Features', y=1.05, size=15)
```

Out[90]: Text(0.5, 1.05, 'Pearson Correlation of Features')



We can see that the attributes have better correlation than before now.

PRE__PROCESSING OF TEST DATA :

As discussed earlier the test data is present in the adult.test file. The preprocessing techniques applied to test data are similar to that of train data.

CLeaning Test Datset

```
col_names=['age','workclass','fnlwgt','education','education_num','marital_status','occupation','relationship','race','sex','capi
test_data=pd.read_csv("adult.test",names=col_names,header=None)
test_data.head(5)
```

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	capital_gain	capital_loss	hours_per_week	nativ
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	Un
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	Un
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	Un
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	Un
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Female	0	0	30	Un

```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16281 entries, 0 to 16280
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             16281 non-null  int64 
 1   workclass       16281 non-null  object
 2   fnlwgt          16281 non-null  int64 
 3   education       16281 non-null  object
 4   education_num   16281 non-null  int64 
 5   marital_status  16281 non-null  object
 6   occupation      16281 non-null  object
 7   relationship    16281 non-null  object
 8   race            16281 non-null  object
 9   sex             16281 non-null  object
 10  capital_gain    16281 non-null  int64 
 11  capital_loss    16281 non-null  int64 
 12  hours_per_week  16281 non-null  int64 
 13  native_country  16281 non-null  object
 14  income          16281 non-null  object
dtypes: int64(6), object(9)
memory usage: 1.9+ MB
```

Dropping Duplicates in Test data

```
#dropping cuplicate rows if any
test_data=test_data.drop_duplicates(keep='first', inplace=False)
test_data=test_data.reset_index(drop=True)
test_data.shape
```

```
(16276, 15)
```

There are 5 dupluicte rows in test data

```
test_data.isnull().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
```

No null values in test data too

Handling Categorical Attributes

```
label_enc = LabelEncoder()
test_data['sex_label'] = label_enc.fit_transform(test_data['sex'])
sex_labels=label_enc.classes_
```

```
train_label_codes=pd.read_csv("label_encodings.csv",index_col='Column_name')
train_label_codes
```

Column_name	Unnamed: 0	Label_encoded
sex	0	[' Female' ' Male']
race	1	[' Amer-Indian-Eskimo' ' Asian-Pac-Islander' '...
work_class	2	[' ?' ' Federal-gov' ' Local-gov' ' Never-work...
marital_status	3	[' Divorced' ' Married-AF-spouse' ' Married-ci...
occupation	4	[' ?' ' Adm-clerical' ' Armed-Forces' ' Craft-...
relationship	5	[' Husband' ' Not-in-family' ' Other-relative'...
native_country	6	[' ?' ' Cambodia' ' Canada' ' China' ' Columbi...

```
str(sex_labels)==train_label_codes.loc['sex']['Label_encoded']
```

```
True
```

The labels assigned for both test and train cases match

Here we check if labels assigned to test and train datasets are same using the previously stored train dataset labels in "label_encodings.csv"

```
test_data['race_label'] = label_enc.fit_transform(test_data['race'])
race_labels=label_enc.classes_
str(race_labels)==train_label_codes.loc['race']['Label_encoded']
```

```
True
```

```
test_data['workclass_label'] = label_enc.fit_transform(test_data['workclass'])
workclass_labels=label_enc.classes_
str(workclass_labels)==train_label_codes.loc['work_class']['Label_encoded']
```

```
True
```

```
test_data['marital_status_label'] = label_enc.fit_transform(test_data['marital_status'])
marital_status_labels=label_enc.classes_
str(marital_status_labels)==train_label_codes.loc['marital_status']['Label_encoded']
```

```
True
```

```
test_data['occupation_label'] = label_enc.fit_transform(test_data['occupation'])
occupation_labels=label_enc.classes_
str(occupation_labels)==train_label_codes.loc['occupation']['Label_encoded']
```

```
True
```

```
test_data['relationship_label'] = label_enc.fit_transform(test_data['relationship'])
relationship_labels=label_enc.classes_
str(relationship_labels)==train_label_codes.loc['relationship']['Label_encoded']
```

```
True
```

```
test_data['native_country_label'] = label_enc.fit_transform(test_data['native_country'])
native_country_labels=label_enc.classes_
str(native_country_labels)==train_label_codes.loc['native_country']['Label_encoded']
```

```
False
```

Here we see that the labels assigned in train data are not matching with labels assigned in test data

```
str(native_country_labels)
```

```
"[' ?' ' Cambodia' ' Canada' ' China' ' Columbia' ' Cuba'\n ' Dominican-Republic' ' Ecuador' ' El-Salvador' ' England' ' Franc
e'\n ' Germany' ' Greece' ' Guatemala' ' Haiti' ' Honduras' ' Hong' ' Hungary'\n ' India' ' Iran' ' Ireland' ' Italy' ' Jamaic
a' ' Japan' ' Laos'\n ' Mexico' ' Nicaragua' ' Outlying-US(Guam-USVI-etc)' ' Peru'\n ' Philippines' ' Poland' ' Portugal' ' Pue
rto-Rico' ' Scotland' ' South'\n ' Taiwan' ' Thailand' ' Trinadad&Tobago' ' United-States' ' Vietnam'\n ' Yugoslavia']"
```

```
train_label_codes.loc['native_country']['Label_encoded']
```

```
"[' ?' ' Cambodia' ' Canada' ' China' ' Columbia' ' Cuba'\n ' Dominican-Republic' ' Ecuador' ' El-Salvador' ' England' ' Franc
e'\n ' Germany' ' Greece' ' Guatemala' ' Haiti' ' Holand-Netherlands'\n ' Honduras' ' Hong' ' Hungary' ' India' ' Iran' ' Irela
nd' ' Italy'\n ' Jamaica' ' Japan' ' Laos' ' Mexico' ' Nicaragua'\n ' Outlying-US(Guam-USVI-etc)' ' Peru' ' Philippines' ' Pola
nd'\n ' Portugal' ' Puerto-Rico' ' Scotland' ' South' ' Taiwan' ' Thailand'\n ' Trinadad&Tobago' ' United-States' ' Vietnam' '
Yugoslavia']"
```

Holand-Netherlands is country present in train data but is missing in the test data due to which the labels are mismatching

The labels of native_country column are not matching as shown .

in order to match those labels with train labels we will have to increment all the labels from Honduras to Yugoslavia by 1

]:
```
for index, row in test_data.iterrows():
    if row['native_country']>' Holand-Netherlands':
        test_data.at[index,'native_country_label']=row['native_country_label']+1
```

]:
```
test_labels=test_data[['native_country','native_country_label']]
test_labels=test_labels.drop_duplicates(keep='first', inplace=False)
test_labels=test_labels.sort_values(by=['native_country'])
test_labels=test_labels.reset_index()
test_labels=test_labels.drop("index",axis=1)
```

]:
```
train_data=pd.read_csv("cleaned_train_data.csv")
train_data=train_data.drop("Unnamed: 0",axis=1)
train_data.head()
```

]:

	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	...	native_country	income	sex_label	race_label
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	...	United-States	<=50K	1	4
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	...	United-States	<=50K	1	4
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	...	United-States	<=50K	1	4
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	...	United-States	<=50K	1	2
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	...	Cuba	<=50K	0	2

:
```
train_labels=train_data[['native_country','native_country_label']]
train_labels=train_labels.drop_duplicates(keep='first', inplace=False)
train_labels=train_labels.sort_values(by=['native_country'])
train_labels=train_labels.reset_index()
train_labels=train_labels.drop("index",axis=1)
```

```
train_labels
```

	native_country	native_country_label
0	?	0
1	Cambodia	1
2	Canada	2
3	China	3
4	Columbia	4
5	Cuba	5
6	Dominican-Republic	6
7	Ecuador	7
8	El-Salvador	8
9	England	9
10	France	10
11	Germany	11
12	Greece	12
13	Guatemala	13
14	Haiti	14
15	Holand-Netherlands	15
16	Honduras	16
17	Hong	17
18	Hungary	18
19	India	19
20	Iran	20
21	Ireland	21
22	Italy	22
23	Jamaica	23
24	Japan	24
25	Laos	25
26	Mexico	26
27	Nicaragua	27
28	Outlying-US(Guam-USVI-etc)	28
29	Peru	29
30	Philippines	30
31	Poland	31
32	Portugal	32
33	Puerto-Rico	33
34	Scotland	34
35	South	35
36	Taiwan	36
37	Thailand	37
38	Trinadad&Tobago	38
39	United-States	39
40	Vietnam	40
41	Yugoslavia	41

```
test_labels
```

	native_country	native_country_label
0	?	0
1	Cambodia	1
2	Canada	2
3	China	3
4	Columbia	4
5	Cuba	5
6	Dominican-Republic	6
7	Ecuador	7
8	El-Salvador	8
9	England	9
10	France	10
11	Germany	11
12	Greece	12
13	Guatemala	13
14	Haiti	14
15	Honduras	16
16	Hong	17
17	Hungary	18
18	India	19
19	Iran	20
20	Ireland	21
21	Italy	22
22	Jamaica	23
23	Japan	24
24	Laos	25
25	Mexico	26
26	Nicaragua	27
27	Outlying-US(Guam-USVI-etc)	28
28	Peru	29
29	Philippines	30
30	Poland	31
31	Portugal	32
32	Puerto-Rico	33
33	Scotland	34
34	South	35
35	Taiwan	36
36	Thailand	37
37	Trinadad&Tobago	38
38	United-States	39
39	Vietnam	40
40	Yugoslavia	41

Now we see that both test and train have been assigned same labels

CLASSIFICATION MODELS :

KNN:-

Model_eval is an user defined function which returns

- Accuracy
- Precision
- Recall
- Sensitivity
- Specificity
- Error_rate

```
def model_eval(actual, pred):

  confusion = pd.crosstab(actual, pred, rownames=['Actual'],  colnames=['Predicted'])
  TP = confusion.loc['>50K','>50K']
  TN = confusion.loc['<=50K','<=50K']
  FP = confusion.loc['<=50K','>50K']
  FN = confusion.loc['>50K','<=50K']

  accuracy = ((TP+TN))/(TP+FN+FP+TN)
  precision = (TP)/(TP+FP)
  recall = (TP)/(TP+FN)
  f_measure = (2*recall*precision)/(recall+precision)
  sensitivity = TP / (TP + FN)
  specificity = TN / (TN + FP)
  error_rate = 1 - accuracy

  out = {}
  out['accuracy'] =  accuracy
  out['precision'] = precision
  out['recall'] = recall
  out['f_measure'] = f_measure
  out['sensitivity'] = sensitivity
  out['specificity'] = specificity
  out['error_rate'] = error_rate

  return out
```

The train_data and test_data contains all the columns from the dataset except the income column .
train_label and test_label contain the information about income from train and test datasets respectively.

```
from sklearn.neighbors import KNeighborsClassifier
knn_outs = []
for i in range(1,50,2):
  knn = KNeighborsClassifier(n_neighbors=i)
  knn.fit(train_data, train_label)
  knn_pred = knn.predict(test_data)
  knn_perf = model_eval(test_label, knn_pred)
  knn_perf['k'] = i
  knn_outs.append(knn_perf)

ovl_knn = round(pd.DataFrame(knn_outs),4)
display(ovl_knn)
```

OUTPUT :

In [48]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_outs = []
for i in range(1,30,2):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(train_data, train_label)
    knn_pred = knn.predict(test_data)
    knn_perf = model_eval(test_label, knn_pred)
    knn_perf['k'] = i
    knn_outs.append(knn_perf)

ovl_knn = round(pd.DataFrame(knn_outs),4)
display(ovl_knn)
```

	accuracy	precision	recall	f_measure	sensitivity	specificity	error_rate	k
0	0.7881	0.5731	0.5466	0.5595	0.5466	0.8670	0.2119	1
1	0.8133	0.6365	0.5632	0.5976	0.5632	0.8949	0.1867	3
2	0.8200	0.6530	0.5735	0.6107	0.5735	0.9005	0.1800	5
3	0.8215	0.6636	0.5572	0.6058	0.5572	0.9078	0.1785	7
4	0.8252	0.6733	0.5635	0.6135	0.5635	0.9107	0.1748	9
5	0.8256	0.6756	0.5605	0.6127	0.5605	0.9121	0.1744	11
6	0.8286	0.6831	0.5665	0.6194	0.5665	0.9142	0.1714	13
7	0.8297	0.6867	0.5672	0.6212	0.5672	0.9155	0.1703	15
8	0.8293	0.6862	0.5652	0.6198	0.5652	0.9156	0.1707	17
9	0.8303	0.6898	0.5645	0.6209	0.5645	0.9171	0.1697	19
10	0.8297	0.6900	0.5599	0.6182	0.5599	0.9179	0.1703	21
11	0.8307	0.6944	0.5579	0.6187	0.5579	0.9198	0.1693	23
12	0.8312	0.6968	0.5566	0.6188	0.5566	0.9209	0.1688	25
13	0.8307	0.6949	0.5569	0.6183	0.5569	0.9202	0.1693	27
14	0.8302	0.6950	0.5526	0.6156	0.5526	0.9208	0.1698	29

From the observations, we can conclude that as k value increases , accuracy increases and error rate decreases.

LOGISTIC REGRESSION :

The train_data and test_data contains all the columns from the dataset except the income column .
train_label and test_label contain the information about income from train and test datasets respectively.

```
log_reg = LogisticRegression(penalty = 'l2', dual = False, tol = 1e-4, fit_intercept = True, solver = 'liblinear')
log_reg.fit(train_data, train_label)
log_reg_pred = log_reg.predict(test_data)
logistic_reg = model_eval(test_label, log_reg_pred)
print('Logistic Regression : %.2f percent.' % (round(logistic_reg['accuracy']*100,3)))
ovl_logreg = round(pd.DataFrame([logistic_reg], index = ['logistic_reg']),4)
display(ovl_logreg)
```

OUTPUT:

```
In [40]: log_reg = LogisticRegression(penalty = 'l2', dual = False, tol = 1e-4, fit_intercept = True, solver = 'liblinear')
         log_reg.fit(train_data, train_label)
         log_reg_pred = log_reg.predict(test_data)
         logistic_reg = model_eval(test_label, log_reg_pred)
         print('Logistic Regression : %.2f percent.' % (round(logistic_reg['accuracy']*100,3)))
         ovl_logreg = round(pd.DataFrame([logistic_reg], index = ['logistic_reg']),4)
         display(ovl_logreg)

         Logistic Regression : 85.44 percent.
```

	accuracy	precision	recall	f_measure	sensitivity	specificity	error_rate
logistic_reg	0.8544	0.7318	0.6002	0.6595	0.6002	0.9325	0.1456

NAIVE BAYES:

features contain all the data except the ‘income’ column.
Target is the ‘income column.

```
algorithm = GaussianNB(priors=None, var_smoothing=1e-9)
algorithm.fit(features, target)
print('ACCURACY - %.2f '%(algorithm.score(features, target)))
```

OUTPUT:

```
In [44]: target = dataframe['income']
         #dataframe=dataframe.drop('income',axis=1)
         features = dataframe.iloc[:,dataframe.columns != 'income']

         print('The Initial DataFrame Contained %d Rows And %d Columns'%(dataframe.shape))
         print('The Features Matrix Contains %d Rows And %d Columns'%(features.shape))
         print('The Target Vector Contains %d Rows And %d Columns'%(np.array(target).reshape(-1, 1).shape))

         The Initial DataFrame Contained 16276 Rows And 15 Columns
         The Features Matrix Contains 16276 Rows And 14 Columns
         The Target Vector Contains 16276 Rows And 1 Columns

In [45]: from sklearn.naive_bayes import GaussianNB
         algorithm = GaussianNB(priors=None, var_smoothing=1e-9)
         algorithm.fit(features, target)
         print(algorithm.classes_)

         [' <=50K.' ' >50K.']

In [46]: print('ACCURACY - %.2f '%(algorithm.score(features, target)))

         ACCURACY - 0.80
```

Decision Tree Classifier:

Required libraries and the train dataset

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
```

```
train_data=pd.read_csv("cleaned_train_data.csv")
train_data=train_data.drop("Unnamed: 0",axis=1)
train_data.head()
```

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

5 rows × 23 columns

Model 1:

In model 1 we have taken all the columns (attributes) as our features and have trained decision trees with different depths

```
feature_cols=['age','fnlwgt','workclass_label','education_num','marital_status_label','occupation_label','relationship_label','sex_label', 'race_label','capital_gain', 'capital_loss', 'hours_per_week','native_country_label']
```

- **Model 1.1** : Depth of tree = 5

```
X_train=train_data[feature_cols]
y_train=train_data['income_label']
print( "X_train_shape = ",X_train.shape,"\ny_train_shape = ",y_train.shape)
```

```
X_train_shape =  (32537, 13)
y_train_shape =  (32537,)
```

```
X_test=test_data[feature_cols]
y_test=test_data['income_label']
print( "X_test_shape = ",X_test.shape,"\ny_test_shape = ",y_test.shape)
```

```
X_test_shape =  (16276, 13)
y_test_shape =  (16276,)
```

```
#making desicion tree classifirer using gini index as selection parameter
classifier = DecisionTreeClassifier(criterion="gini")
classifier = classifier.fit(X_train,y_train) #training data
y_pred = classifier.predict(X_test) # predict test data
print("Accuracy using Gini index as selection factor :",metrics.accuracy_score(y_test, y_pred))
```

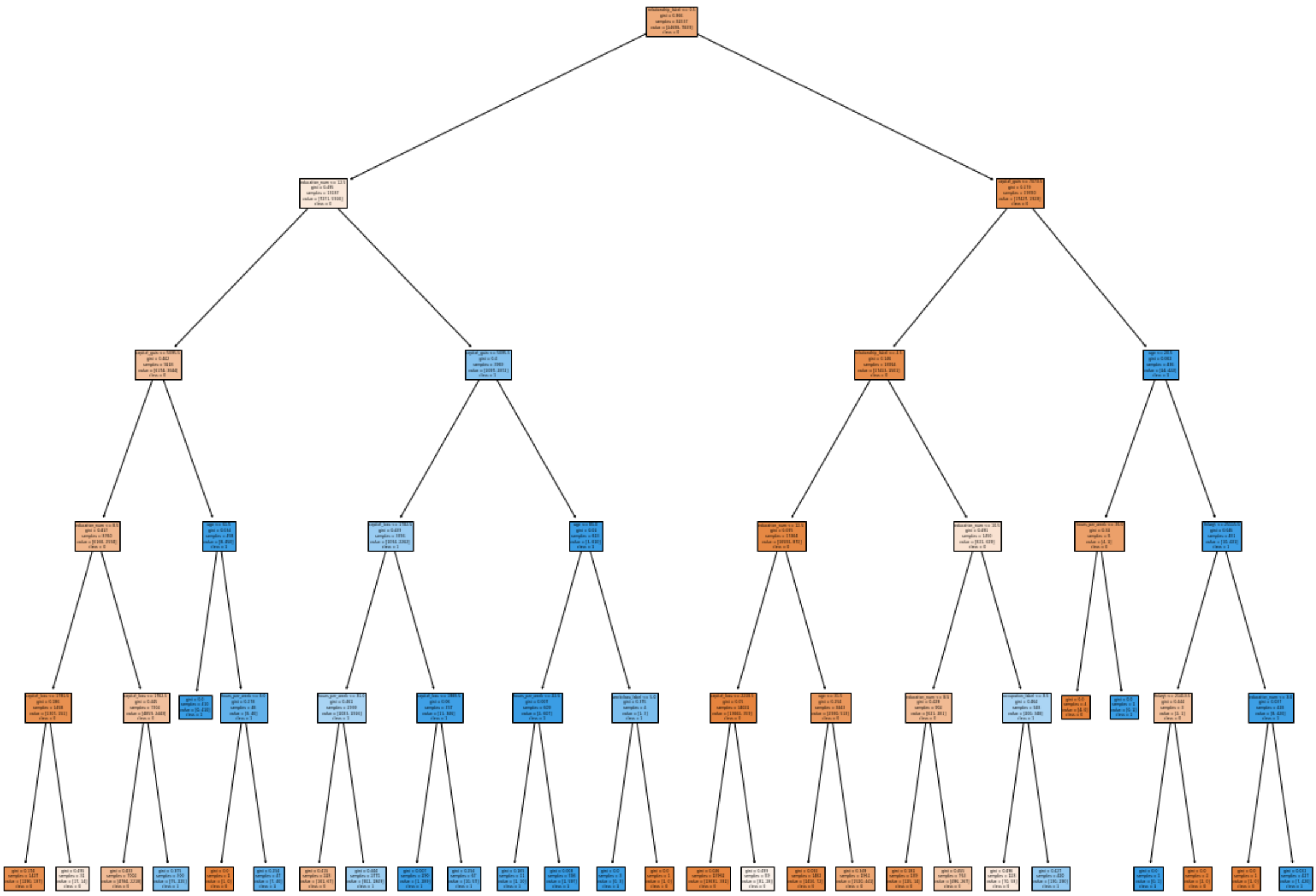
```
Accuracy using Gini index as selection factor : 0.8089825509953306
```

```
#making desicion tree classifirer using gini index as selection parameter
classifier = DecisionTreeClassifier(criterion="gini",max_depth=5)
classifier = classifier.fit(X_train,y_train) #training data
y_pred = classifier.predict(X_test) # predict test data
print("Accuracy using Gini index as selection factor :",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy using Gini index as selection factor : 0.8513762595232244
```

Graph representation of decision tree

```
fig = plt.figure(figsize=(20,16))
_ = plot_tree(classifier,
                    feature_names=feature_cols,
                    class_names=['0','1'],
                    filled=True)
```

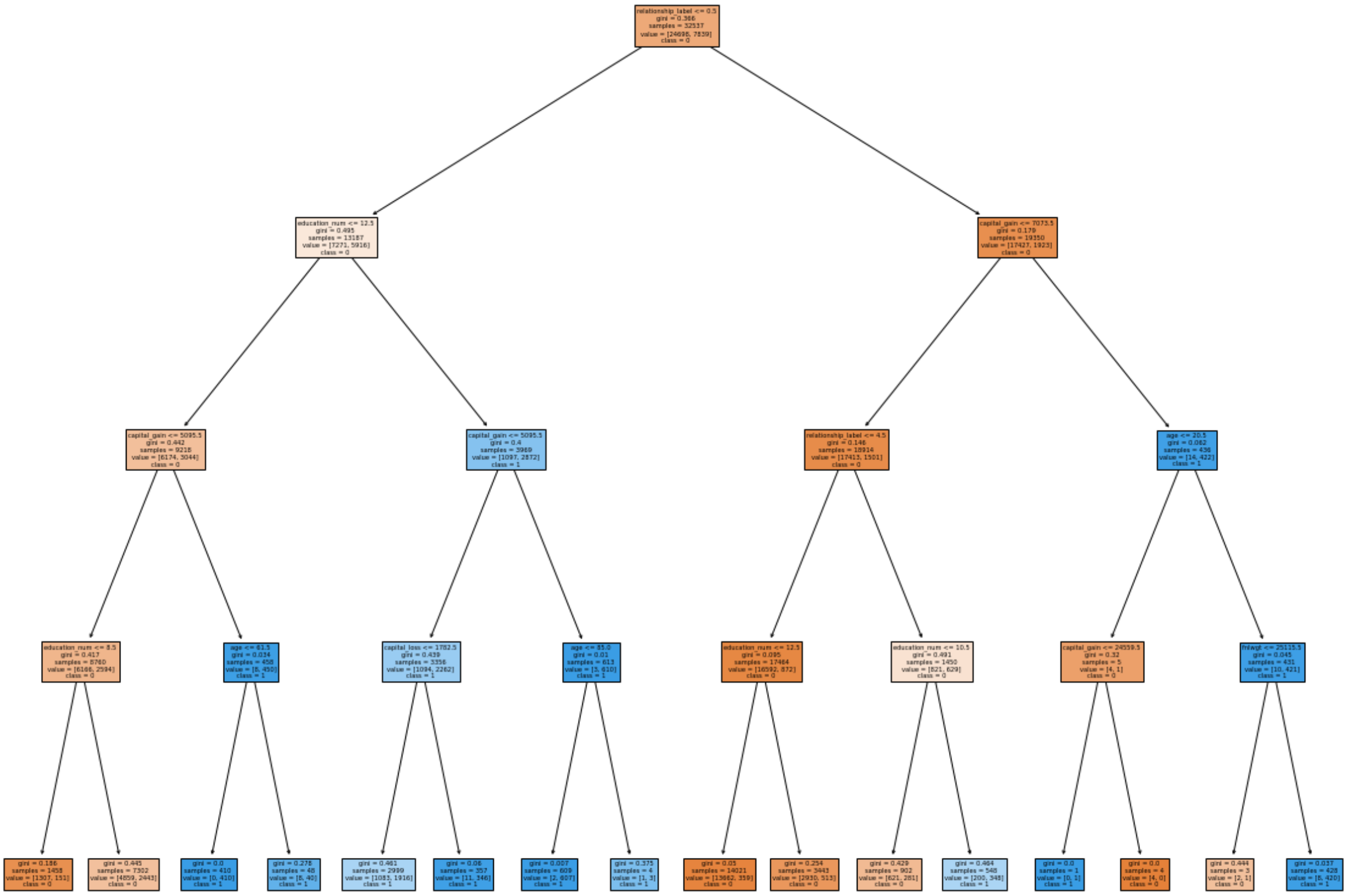


- **Model 1.2 :** Depth of tree = 4

```
#making desicion tree classifirer using gini index as selection parameter
classifier = DecisionTreeClassifier(criterion="gini",max_depth=4)
classifier = classifier.fit(X_train,y_train) #training data
y_pred = classifier.predict(X_test) # predict test data
print("Accuracy using Gini index as selection factor :",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy using Gini index as selection factor : 0.844310641435242
```

```
fig = plt.figure(figsize=(20,16))
_ = plot_tree(classifier,
                    feature_names=feature_cols,
                    class_names=['0','1'],
                    filled=True)
```

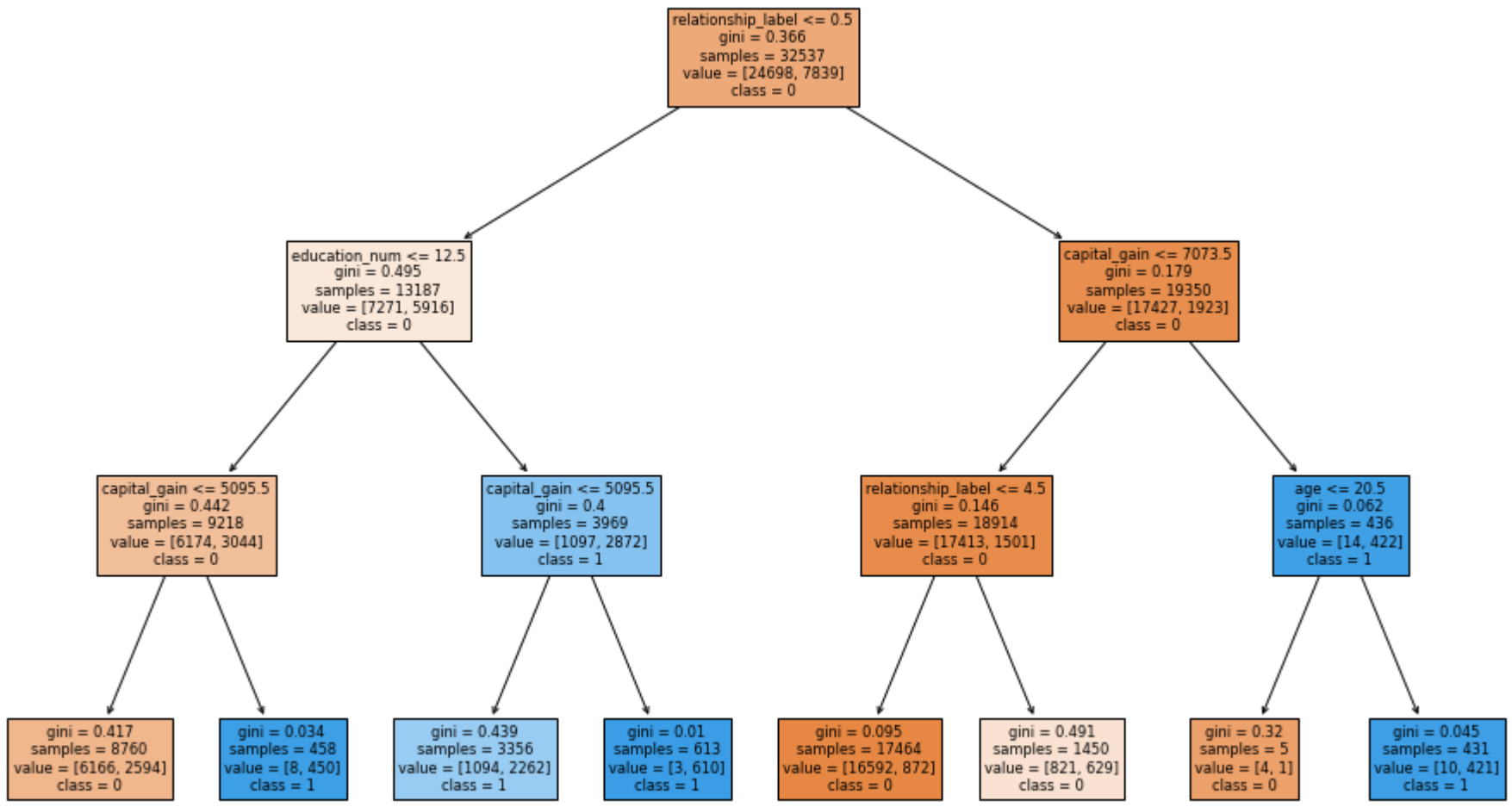


❖ Model 1.3 : Depth of tree = 3

```
#making desicion tree classifirer using gini index as selection parameter
classifier = DecisionTreeClassifier(criterion="gini",max_depth=3)
classifier = classifier.fit(X_train,y_train) #training data
y_pred = classifier.predict(X_test) # predict test data
print("Accuracy using Gini index as selection factor :",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy using Gini index as selection factor : 0.8392111083804374
```

```
fig = plt.figure(figsize=(16,10))
_ = plot_tree(classifier,
                    feature_names=feature_cols,
                    class_names=['0','1'],
                    filled=True)
```

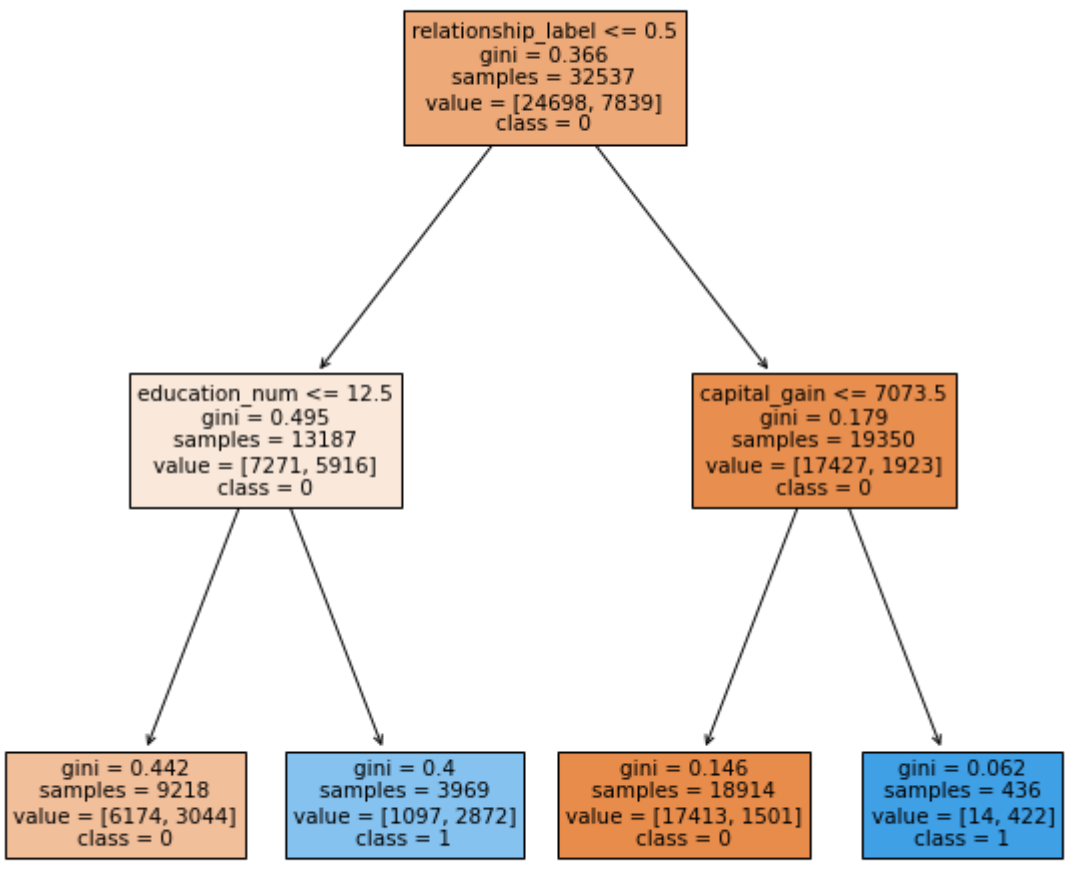


- **Model 1.4 :** Depth of tree = 2

```
#making desicion tree classifirer using gini index as selection parameter
classifier = DecisionTreeClassifier(criterion="gini",max_depth=2)
classifier = classifier.fit(X_train,y_train) #training data
y_pred = classifier.predict(X_test) # predict test data
print("Accuracy using Gini index as selection factor :",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy using Gini index as selection factor : 0.8268001966085033
```

```
fig = plt.figure(figsize=(10,10))
_ = plot_tree(classifier,
                   feature_names=feature_cols,
                   class_names=['0','1'],
                   filled=True)
```

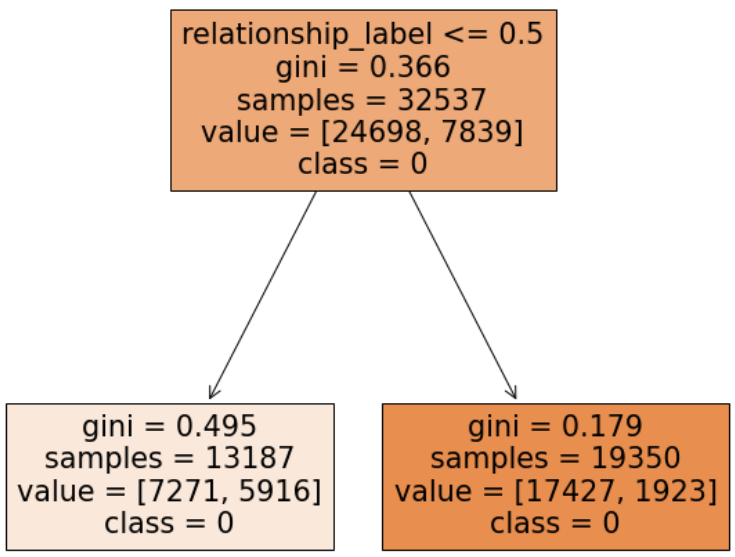


❖ **Model 1.5 :** Depth of tree = 1

```
#making desicion tree classifirer using gini index as selection parameter
classifier = DecisionTreeClassifier(criterion="gini",max_depth=1)
classifier = classifier.fit(X_train,y_train) #training data
y_pred = classifier.predict(X_test) # predict test data
print("Accuracy using Gini index as selection factor :",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy using Gini index as selection factor : 0.763701155074957
```

```
fig = plt.figure(figsize=(10,10))
_ = plot_tree(classifier,
                   feature_names=feature_cols,
                   class_names=['0','1'],
                   filled=True)
```



Model 2:

In model 2 we have taken 'age','education_num','sex_label','capital_gain','capital_loss','hours_per_week' as our features and have trained decision trees with different depths. We have dropped workclass, occupation and native_country columns which had missing values represented as " ? "

```
feature_cols=['age','education_num','sex_label','capital_gain','capital_loss','hours_per_week']
```

- **Model 2.1 :** Depth of tree = 4

```
X_train=train_data[feature_cols]
y_train=train_data['income_label']
print( "X_train_shape = ",X_train.shape,"\ny_train_shape = ",y_train.shape)
```

```
X_train_shape =  (32537, 6)
y_train_shape =  (32537,)
```

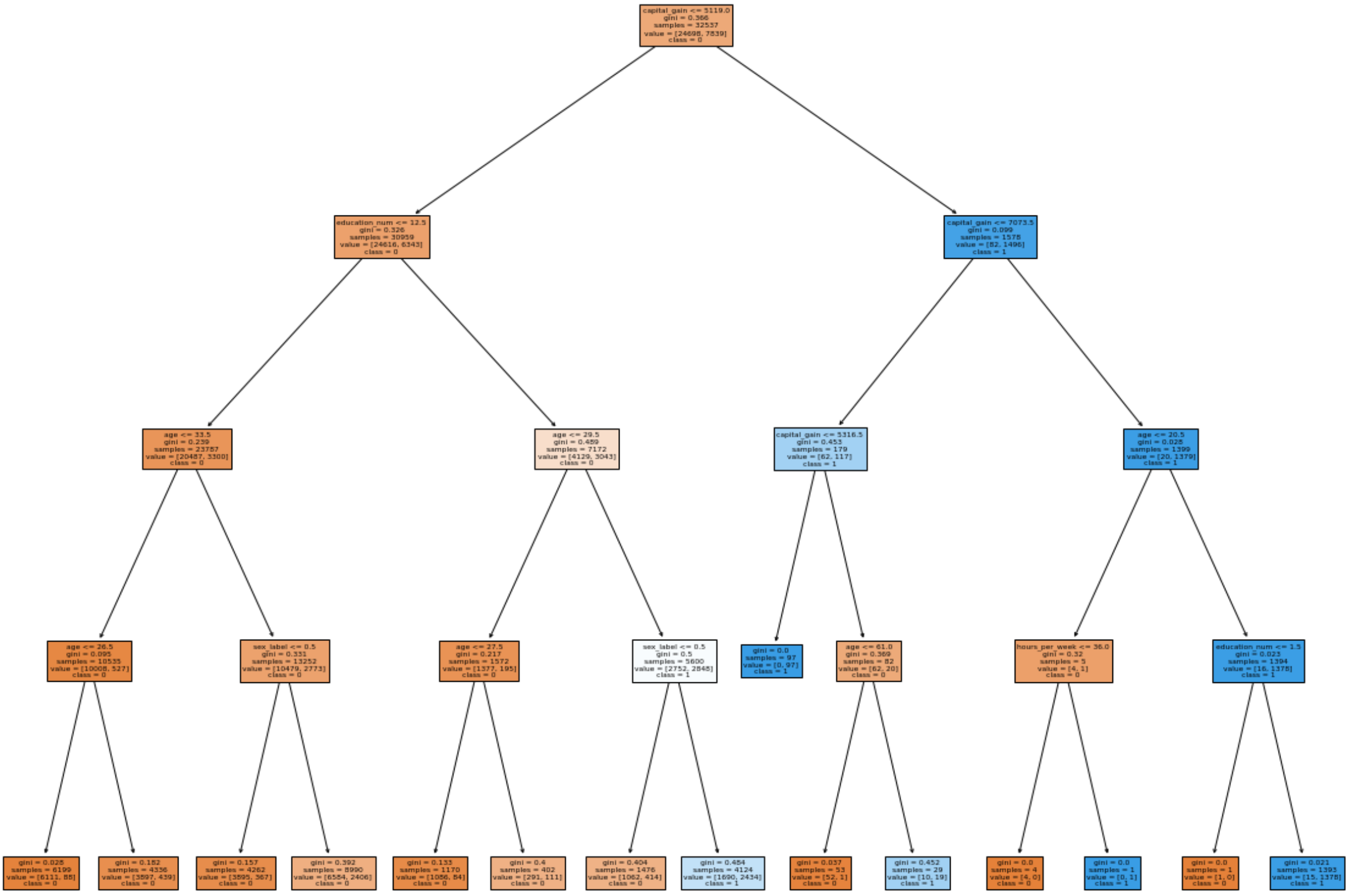
```
X_test=test_data[feature_cols]
y_test=test_data['income_label']
print( "X_test_shape = ",X_test.shape,"\ny_test_shape = ",y_test.shape)
```

```
X_test_shape =  (16276, 6)
y_test_shape =  (16276,)
```

```
#making desicion tree classifirer using gini index as selection parameter
classifier = DecisionTreeClassifier(criterion="gini",max_depth=4)
classifier = classifier.fit(X_train,y_train) #training data
y_pred = classifier.predict(X_test) # predict test data
print("Accuracy using Gini index as selection factor :",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy using Gini index as selection factor : 0.8251413123617597
```

```
fig = plt.figure(figsize=(20,16))
_ = plot_tree(classifier,
                   feature_names=feature_cols,
                   class_names=['0','1'],
                   filled=True)
```

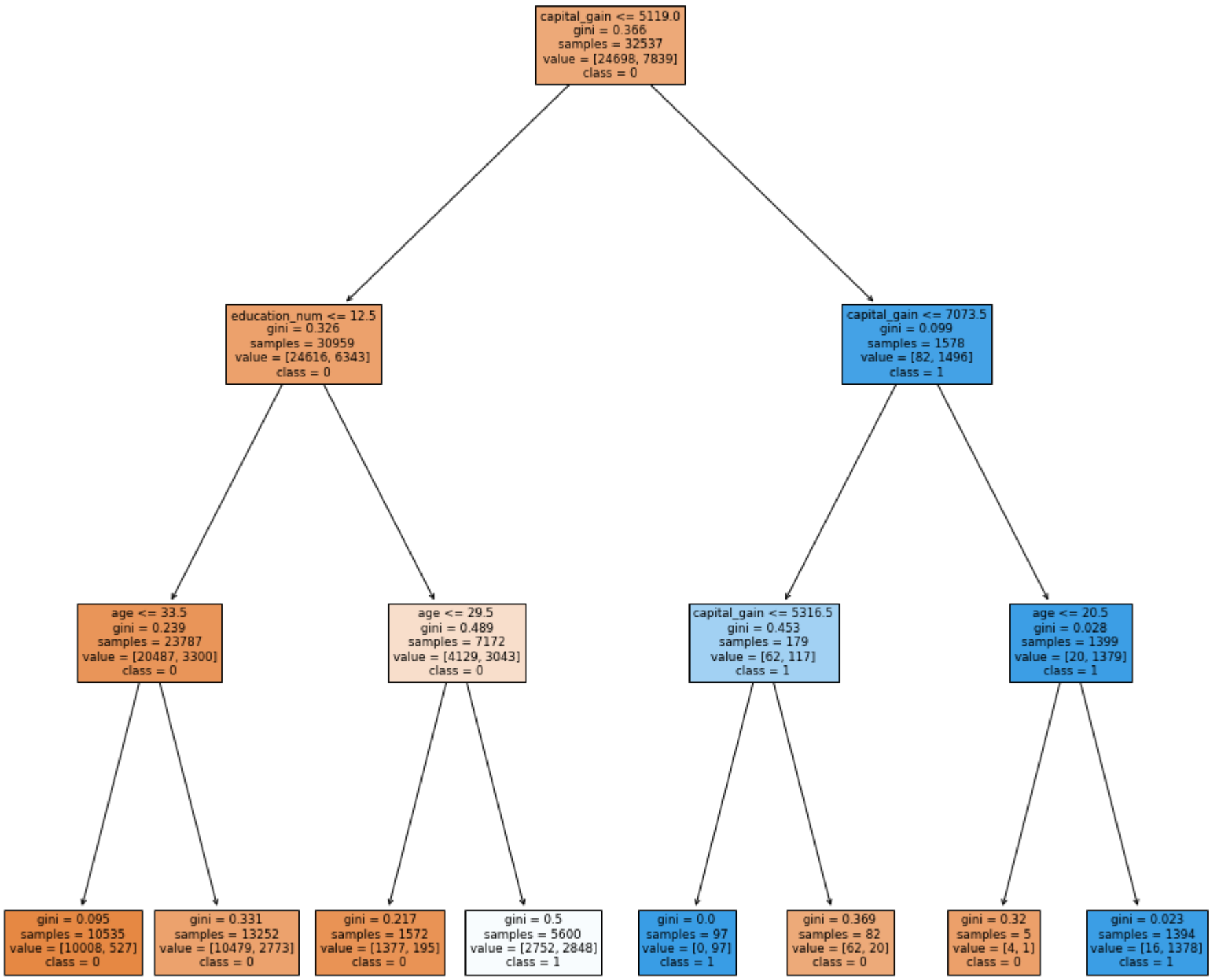

capital_gain <= 5119.0
gini = 0.366
samples = 32537
value = [24698, 7839]
class = 0
education_num <= 12.5
gini = 0.326
samples = 30959
value = [24616, 6343]
class = 0
capital_gain <= 7073.5
gini = 0.099
samples = 1578
value = [82, 1496]
class = 1
age <= 33.5
gini = 0.239
samples = 23787
value = [20487, 3300]
class = 0
age <= 29.5
gini = 0.489
samples = 7172
value = [4129, 3043]
class = 0
capital_gain <= 5316.5
gini = 0.453
samples = 179
value = [62, 117]
class = 1
age <= 20.5
gini = 0.025
samples = 1399
value = [20, 1379]
class = 1
age <= 26.5
gini = 0.095
samples = 10535
value = [10008, 527]
class = 0
sex_label <= 0.5
gini = 0.331
samples = 13252
value = [10479, 2773]
class = 0
age <= 27.5
gini = 0.217
samples = 1572
value = [1377, 195]
class = 0
sex_label <= 0.5
gini = 0.5
samples = 5600
value = [2752, 2848]
class = 1
gini = 0.0
samples = 97
value = [0, 97]
class = 1
age <= 61.0
gini = 0.369
samples = 82
value = [62, 20]
class = 0
hours_per_week <= 36.0
gini = 0.32
samples = 5
value = [4, 1]
class = 0
education_num <= 1.5
gini = 0.023
samples = 1394
value = [16, 1378]
class = 1
gini = 0.028
samples = 6199
value = [6113, 86]
class = 0
gini = 0.182
samples = 4336
value = [3597, 439]
class = 0
gini = 0.157
samples = 4262
value = [3595, 367]
class = 0
gini = 0.392
samples = 8990
value = [6554, 2406]
class = 0
gini = 0.133
samples = 1170
value = [1056, 84]
class = 0
gini = 0.4
samples = 402
value = [291, 111]
class = 0
gini = 0.404
samples = 1476
value = [1062, 414]
class = 0
gini = 0.484
samples = 4124
value = [1690, 2434]
class = 1
gini = 0.037
samples = 53
value = [52, 1]
class = 0
gini = 0.452
samples = 29
value = [10, 19]
class = 1
gini = 0.0
samples = 4
value = [4, 0]
class = 0
gini = 0.0
samples = 1
value = [0, 1]
class = 1
gini = 0.0
samples = 1
value = [1, 0]
class = 0
gini = 0.021
samples = 1393
value = [15, 1378]
class = 1

❖ **Model 2.2 :** Depth of tree = 3

```
#making desicion tree classifirer using gini index as selection parameter
classifier = DecisionTreeClassifier(criterion="gini",max_depth=3)
classifier = classifier.fit(X_train,y_train) #training data
y_pred = classifier.predict(X_test) # predict test data
print("Accuracy using Gini index as selection factor :",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy using Gini index as selection factor : 0.8043745391988204
```

```
fig = plt.figure(figsize=(16,16))
_ = plot_tree(classifier,
                    feature_names=feature_cols,
                    class_names=['0','1'],
                    filled=True)
```

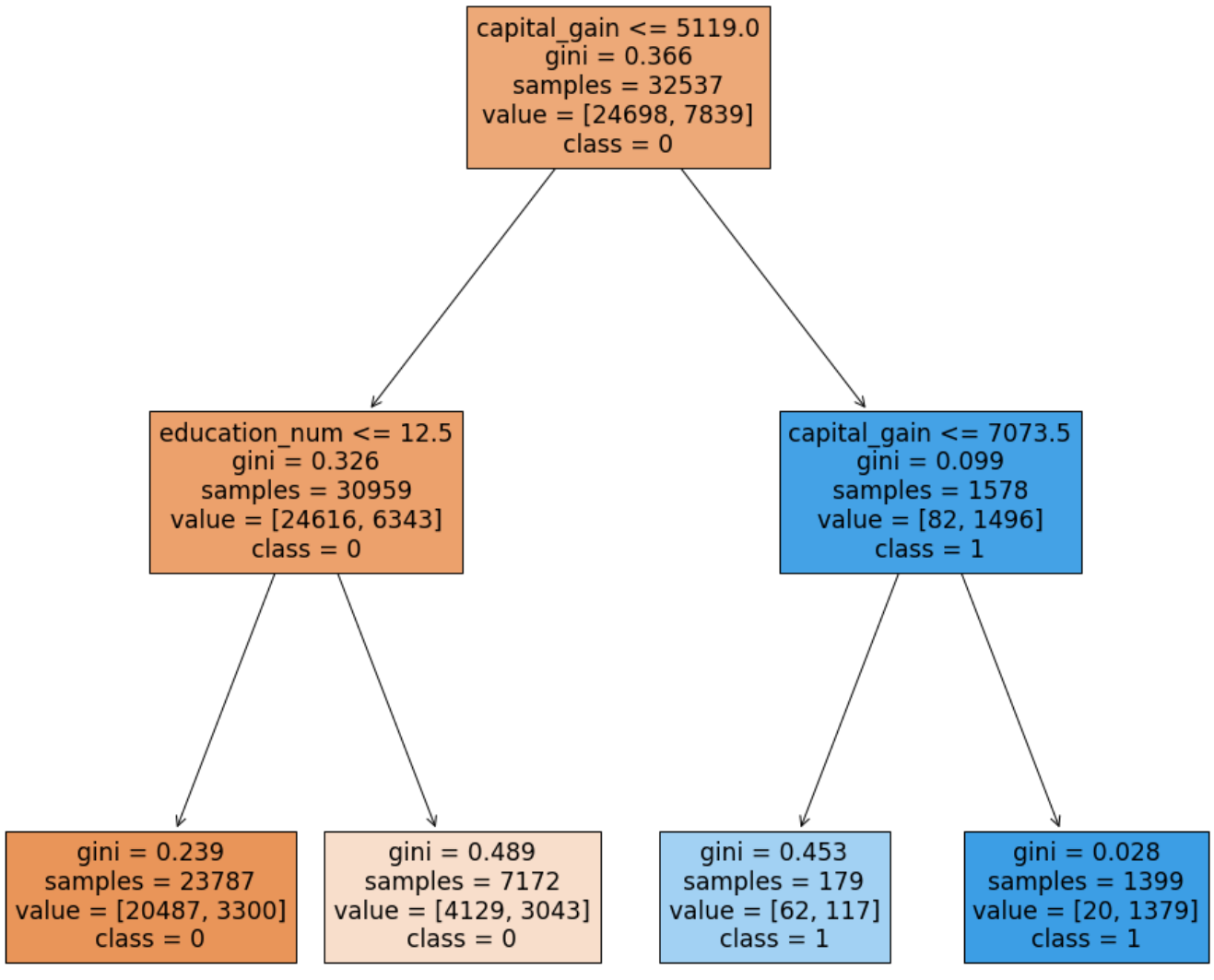


❖ Model 2.3 : Depth of tree = 2

```
#making desicion tree classifirer using gini index as selection parameter
classifier = DecisionTreeClassifier(criterion="gini",max_depth=2)
classifier = classifier.fit(X_train,y_train) #training data
y_pred = classifier.predict(X_test) # predict test data
print("Accuracy using Gini index as selection factor :",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy using Gini index as selection factor : 0.8048660604571147
```

```
fig = plt.figure(figsize=(16,16))
_ = plot_tree(classifier,
                   feature_names=feature_cols,
                   class_names=['0','1'],
                   filled=True)
```

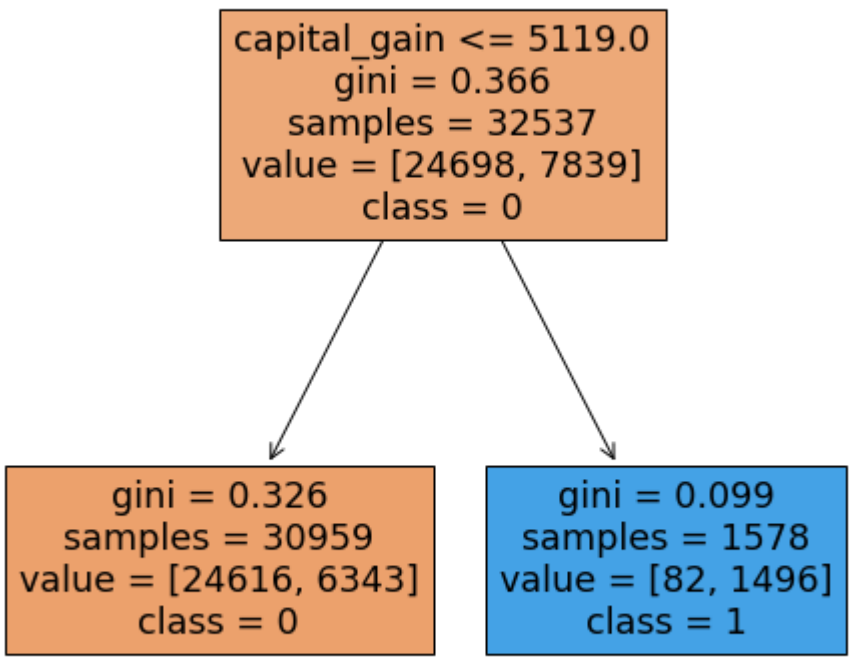


- Model 2.4 : Depth of tree = 1

```
#making desicion tree classifirer using gini index as selection parameter
classifier = DecisionTreeClassifier(criterion="gini",max_depth=1)
classifier = classifier.fit(X_train,y_train) #training data
y_pred = classifier.predict(X_test) # predict test data
print("Accuracy using Gini index as selection factor :",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy using Gini index as selection factor : 0.8048660604571147
```

```
fig = plt.figure(figsize=(8,8))
_ = plot_tree(classifier,
                    feature_names=feature_cols,
                    class_names=['0','1'],
                    filled=True)
```



Model 3 :

In model 2 we have taken 'age','education_num','sex_label' as our features and have trained decision trees with different depths.

```
feature_cols=['age','education_num','sex_label']
```

- Model 3.1 : Depth of tree = 3

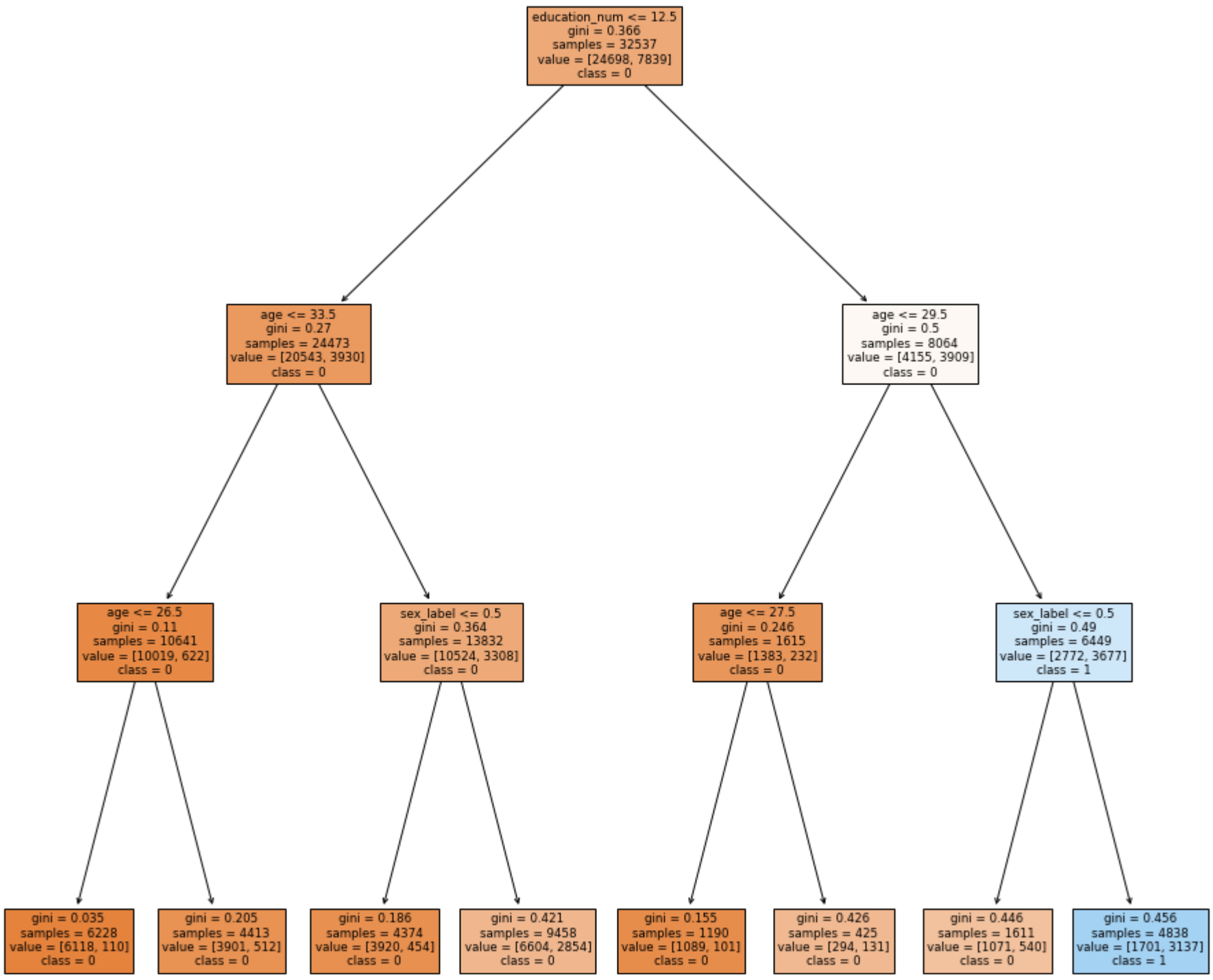
```
feature_cols=['age','education_num','sex_label']
```

```
X_train=train_data[feature_cols]
y_train=train_data['income_label']
print( "X_train_shape = ",X_train.shape,"\ny_train_shape = ",y_train.shape)
```

```
X_train_shape =  (32537, 3)
y_train_shape =  (32537,)
```

```
X_test=test_data[feature_cols]
y_test=test_data['income_label']
print( "X_test_shape = ",X_test.shape,"\ny_test_shape = ",y_test.shape)
```

```
X_test_shape =  (16276, 3)
y_test_shape =  (16276,)
```



❖ Model 3.2 : Depth of tree =2

```
feature_cols=['age','education_num','sex_label']
```

```
X_train=train_data[feature_cols]
y_train=train_data['income_label']
print( "X_train_shape = ",X_train.shape,"\ny_train_shape = ",y_train.shape)
```

```
X_train_shape =  (32537, 3)
y_train_shape =  (32537,)
```

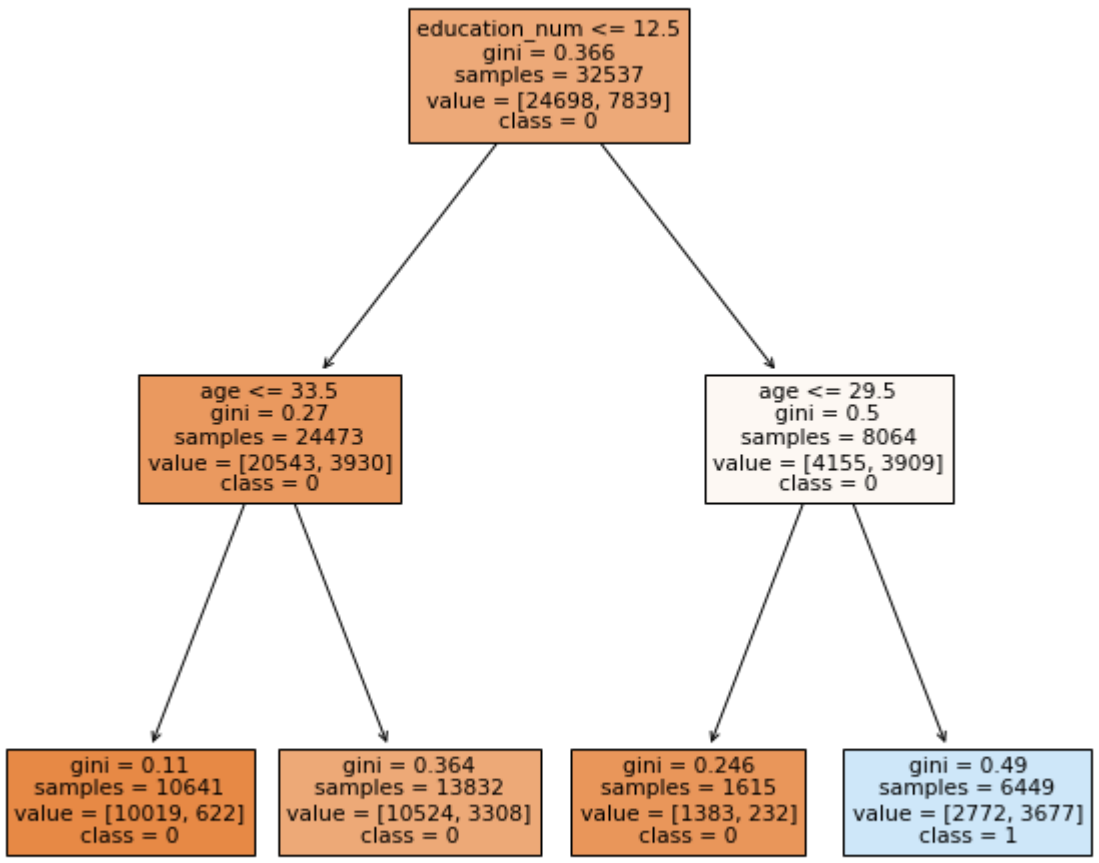
```
X_test=test_data[feature_cols]
y_test=test_data['income_label']
print( "X_test_shape = ",X_test.shape,"\ny_test_shape = ",y_test.shape)
```

```
X_test_shape =  (16276, 3)
y_test_shape =  (16276,)
```

```
#making desicion tree classifirer using gini index as selection parameter
classifier = DecisionTreeClassifier(criterion="gini",max_depth=3)
classifier = classifier.fit(X_train,y_train) #training data
y_pred = classifier.predict(X_test) # predict test data
print("Accuracy using Gini index as selection factor :",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy using Gini index as selection factor : 0.8029614155812239
```

```
fig = plt.figure(figsize=(16,16))
_ = plot_tree(classifier,
                   feature_names=feature_cols,
                   class_names=['0','1'],
                   filled=True)
```

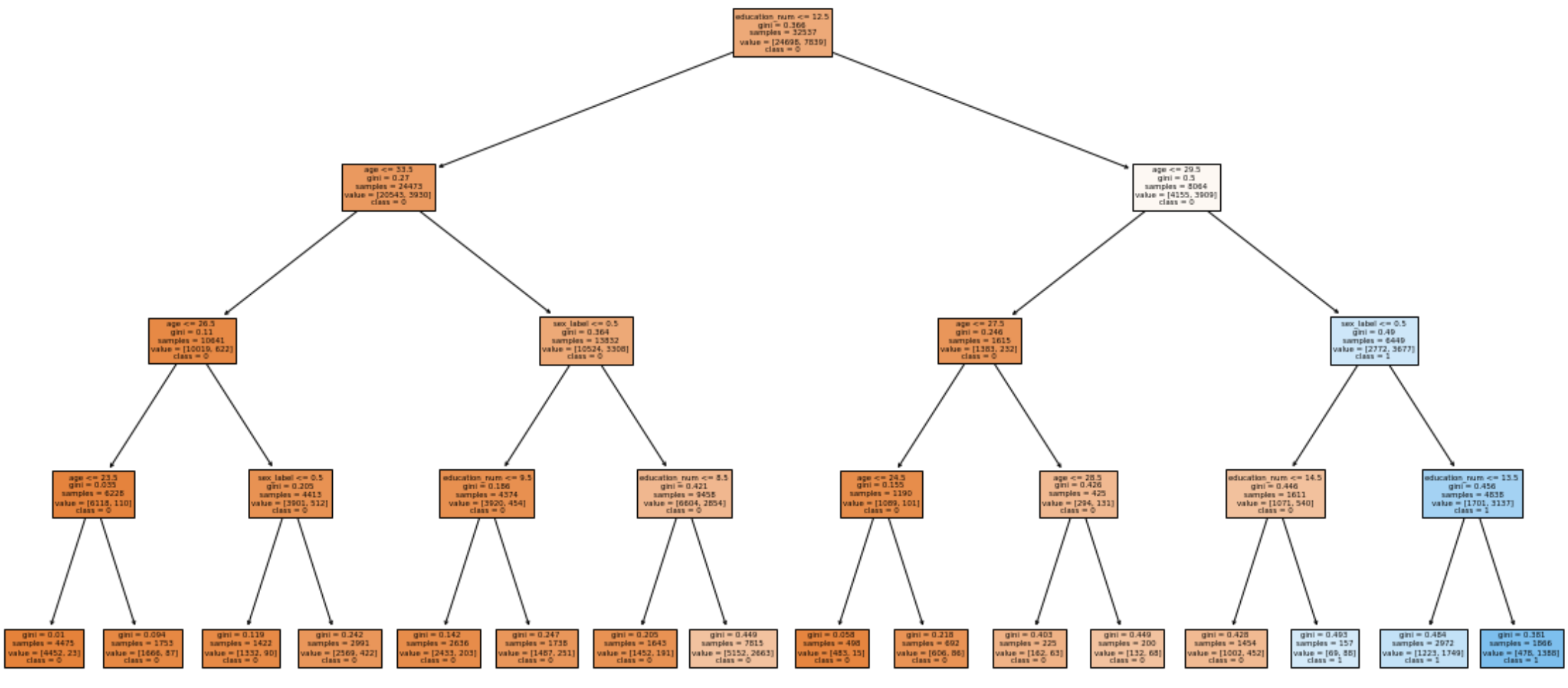


❖ Model 3.3 : Depth of tree = 4

```
#making desicion tree classifirer using gini index as selection parameter
classifier = DecisionTreeClassifier(criterion="gini",max_depth=4)
classifier = classifier.fit(X_train,y_train) #training data
y_pred = classifier.predict(X_test) # predict test data
print("Accuracy using Gini index as selection factor :",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy using Gini index as selection factor : 0.8034529368395184
```

```
fig = plt.figure(figsize=(20,10))
_ = plot_tree(classifier,
                    feature_names=feature_cols,
                    class_names=['0','1'],
                    filled=True)
```



Model 1 :

```
feature_cols=['age','fnlwgt','workclass_label','education_num','marital_status_label','occupation_label','relationship_label','sex_label', 'race_label','capital_gain', 'capital_loss', 'hours_per_week','native_country_label']
```

Model 2:

```
feature_cols=['age','education_num','sex_label','capital_gain','capital_loss','hours_per_week']
```

Model 3:

```
feature_cols=['age','education_num','sex_label']
```

Accuracy Measures

Depth of tree →	1	2	3	4	5
Model 1	0.763701155074957	0.8268001966085033	0.8392111083804374	0.8443106414352422	0.8513762595232244
Model 2	0.8048660604571147	0.8048660604571147	0.8043745391988204	0.8251413123617597	-
Model 3	-	0.786126812484644	0.8029614155812239	0.8034529368395184	-

Inferences :

- Though using all the columns in the dataset is providing more accuracy comparatively , removing most columns is not changing the accuracy much . So using Back selection we can say that sex, education and age are 3 attributes that are contributing much in classification

K - Means Clustering :

Model 1:

```
feature_cols=['age','education_num','sex_label','capital_gain','capital_loss','hours_per_week']
```

```
feature_cols=['age','education_num','sex_label','capital_gain','capital_loss','hours_per_week']
```

```
X=train_data[feature_cols]
y=train_data['income_label']
print( "x_shape = ",X.shape,"\ny_shape = ",y.shape)
```

```
x_shape =  (32537, 6)
y_shape =  (32537,)
```

```
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(X)
```

```
KMeans(n_clusters=2, random_state=0)
```

```
kmeans.cluster_centers_
```

```
array([[3.85473778e+01, 1.00678856e+01, 6.68293286e-01, 5.92670424e+02,
        8.77972698e+01, 4.03943727e+01],
       [4.63584906e+01, 1.29182390e+01, 8.61635220e-01, 9.99990000e+04,
        0.00000000e+00, 4.97987421e+01]])
```

```
labels = kmeans.labels_
correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
```

```
Result: 24857 out of 32537 samples were correctly labeled.
```

```
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Accuracy score: 0.76
```

Model 2:

feature_cols=['age','education_num','sex_label','capital_gain']

```
feature_cols=['age','education_num','sex_label','capital_gain']
```

```
X=train_data[feature_cols]
y=train_data['income_label']
print( "x_shape = ",X.shape,"\ny_shape = ",y.shape)
```

```
x_shape =  (32537, 4)
y_shape =  (32537,)
```

```
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(X)
```

```
KMeans(n_clusters=2, random_state=0)
```

```
kmeans.cluster_centers_
```

```
array([[3.85473778e+01, 1.00678856e+01, 6.68293286e-01, 5.92670424e+02],
       [4.63584906e+01, 1.29182390e+01, 8.61635220e-01, 9.99990000e+04]])
```

```
labels = kmeans.labels_
correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 24857 out of 32537 samples were correctly labeled.
Accuracy score: 0.76
```

Model 3:

feature_cols=['age','education_num','sex_label']

```
feature_cols=['age','education_num','sex_label']
```

```
X=train_data[feature_cols]
y=train_data['income_label']
print( "x_shape = ",X.shape,"\ny_shape = ",y.shape)
```

```
x_shape =  (32537, 3)
y_shape =  (32537,)
```

```
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(X)
```

```
KMeans(n_clusters=2, random_state=0)
```

```
kmeans.cluster_centers_
```

```
array([[29.13318674, 10.01617716,  0.64305534],
       [52.02329562, 10.17512653,  0.70646026]])
```

```
labels = kmeans.labels_
correct_labels = sum(y == labels)
print("Result: %d out of %d samples were correctly labeled." % (correct_labels, y.size))
print('Accuracy score: {0:0.2f}'. format(correct_labels/float(y.size)))
```

```
Result: 20766 out of 32537 samples were correctly labeled.
Accuracy score: 0.64
```

Clustering Accuracies :

Model 1	0.76
Model 2	0.76
Model 3	0.64

Decision Tree model performed better than the Clustering

BIG DATA

Training Decision tree in pyspark

Initialization and libraries

```
import findspark
findspark.init()
```

```
from pyspark import SparkContext
from pyspark.sql import SQLContext
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.feature import VectorAssembler
from sklearn.metrics import confusion_matrix
import pandas as pd
```

```
sc=SparkContext('local')
```

```
bd_train_data=pd.read_csv("cleaned_train_data.csv")
bd_train_data.head()
```

	Unnamed: 0	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	...	native_country	income	sex_label	race_la
0	0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	...	United-States	<=50K	1	
1	1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	...	United-States	<=50K	1	
2	2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	...	United-States	<=50K	1	
3	3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	...	United-States	<=50K	1	
4	4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	...	Cuba	<=50K	0	

5 rows × 24 columns

```
sc = SparkContext().getOrCreate()
sqlContext = SQLContext(sc)
```

Creating Dataframe object in pyspark

```
data = sqlContext.createDataFrame(bd_train_data)
print(data.printSchema())
```

```
root
 |-- Unnamed: 0: long (nullable = true)
 |-- age: long (nullable = true)
 |-- workclass: string (nullable = true)
 |-- fnlwgt: long (nullable = true)
 |-- education: string (nullable = true)
 |-- education_num: long (nullable = true)
 |-- marital_status: string (nullable = true)
 |-- occupation: string (nullable = true)
 |-- relationship: string (nullable = true)
 |-- race: string (nullable = true)
 |-- sex: string (nullable = true)
 |-- capital_gain: long (nullable = true)
 |-- capital_loss: long (nullable = true)
 |-- hours_per_week: long (nullable = true)
 |-- native_country: string (nullable = true)
 |-- income: string (nullable = true)
 |-- sex_label: long (nullable = true)
 |-- race_label: long (nullable = true)
 |-- workclass_label: long (nullable = true)
 |-- marital_status_label: long (nullable = true)
 |-- occupation_label: long (nullable = true)
 |-- relationship_label: long (nullable = true)
 |-- native_country_label: long (nullable = true)
 |-- income_label: long (nullable = true)

None
```

Taking 'age','fnlwgt','education_num','sex_label','capital_gain','capital_loss','hours_per_week' as features. In pyspark we

need to convert these input features as vectors as shown below . We then fit this data of input vectors and label into decision tree

```
features=['age','fnlwgt','education_num','sex_label','capital_gain','capital_loss','hours_per_week']

va = VectorAssembler(inputCols = features, outputCol='features')
va_df = va.transform(data)
va_df = va_df.select(['features', 'income_label'])
va_df.show(3)
```

```
+--------------------+------------+
|            features|income_label|
+--------------------+------------+
|[39.0,77516.0,13....|           0|
|[50.0,83311.0,13....|           0|
|[38.0,215646.0,9....|           0|
+--------------------+------------+
only showing top 3 rows
```

```
dtc = DecisionTreeClassifier(maxDepth=3,featuresCol="features", labelCol="income_label")
dtc = dtc.fit(va_df)
```

Vectorizing test data and predicting the labels of input test data

```
test_va = VectorAssembler(inputCols = features, outputCol='features')
test_va_df = test_va.transform(test_data)
test_va_df = test_va_df.select(['features', 'income_label'])
test_va_df.show(3)
```

```
+--------------------+------------+
|            features|income_label|
+--------------------+------------+
|[25.0,226802.0,7....|           0|
|[38.0,89814.0,9.0...|           0|
|[28.0,336951.0,12...|           1|
+--------------------+------------+
only showing top 3 rows
```

```
pred = dtc.transform(test_va_df)
pred.show(3)
```

```
+--------------------+------------+----------------+--------------------+----------+
|            features|income_label|   rawPrediction|         probability|prediction|
+--------------------+------------+----------------+--------------------+----------+
|[25.0,226802.0,7....|           0|[20530.0,3526.0]|[0.85342534087130...|       0.0|
|[38.0,89814.0,9.0...|           0|[20530.0,3526.0]|[0.85342534087130...|       0.0|
|[28.0,336951.0,12...|           1|[20530.0,3526.0]|[0.85342534087130...|       0.0|
+--------------------+------------+----------------+--------------------+----------+
only showing top 3 rows
```

Making Confusion matrix

```
y_pred=pred.select("prediction").collect()
y_orig=pred.select("income_label").collect()

cm = confusion_matrix(y_orig, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[10914  1516]
 [ 1822  2024]]
```

```
evaluator=MulticlassClassificationEvaluator().setLabelCol("income_label").setPredictionCol("prediction").setMetricName("accuracy"
acc=evaluator.evaluate(pred)
print(acc)
```

Accuracy = 0.7949127549766527

Model 2 : Taking 'age','education_num','sex_label' as features

```
features=['age','education_num','sex_label']
```

```
va = VectorAssembler(inputCols = features, outputCol='features')
va_df = va.transform(data)
va_df = va_df.select(['features', 'income_label'])
va_df.show(3)
```

```
+---------------+------------+
|       features|income_label|
+---------------+------------+
|[39.0,13.0,1.0]|           0|
|[50.0,13.0,1.0]|           0|
| [38.0,9.0,1.0]|           0|
+---------------+------------+
only showing top 3 rows
```

```
dtc = DecisionTreeClassifier(maxDepth=3,featuresCol="features", labelCol="income_label")
dtc = dtc.fit(va_df)
```

```
test_va = VectorAssembler(inputCols = features, outputCol='features')
test_va_df = test_va.transform(test_data)
test_va_df = test_va_df.select(['features', 'income_label'])
test_va_df.show(3)
```

```
+---------------+------------+
|       features|income_label|
+---------------+------------+
| [25.0,7.0,1.0]|           0|
| [38.0,9.0,1.0]|           0|
```

```
pred = dtc.transform(test_va_df)
pred.show(3)
```

```
+---------------+------------+----------------+--------------------+----------+
|       features|income_label|   rawPrediction|         probability|prediction|
+---------------+------------+----------------+--------------------+----------+
| [25.0,7.0,1.0]|           0|[20543.0,3930.0]|[0.83941486536182...|       0.0|
| [38.0,9.0,1.0]|           0|[20543.0,3930.0]|[0.83941486536182...|       0.0|
|[28.0,12.0,1.0]|           1|[20543.0,3930.0]|[0.83941486536182...|       0.0|
+---------------+------------+----------------+--------------------+----------+
only showing top 3 rows
```

```
y_pred=pred.select("prediction").collect()
y_orig=pred.select("income_label").collect()

cm = confusion_matrix(y_orig, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[11524   906]
 [ 2312  1534]]
```

```
evaluator=MulticlassClassificationEvaluator().setLabelCol("income_label").setPredictionCol("prediction").setMetricName("accuracy'
acc=evaluator.evaluate(pred)
print(acc)
```

```
0.802285573851069
```

For features ['age','education_num','sex_label'] and depth 3 , decision tree model without pyspark gave 0.8029614155812239 while model in pyspark gave 0.80228557 . These values are very close

Naive Bayes in Pyspark :

Model 1 : features = ['age','education_num','sex_label']

Naive Bayes

```
from pyspark.ml.classification import NaiveBayes
```

```
nb = NaiveBayes(featuresCol="features", labelCol="income_label",smoothing=1.0, modelType="multinomial")
nb = nb.fit(va_df)
```

```
pred = nb.transform(test_va_df)
pred.show(3)
```

```
+---------------+------------+--------------------+--------------------+----------+
|       features|income_label|       rawPrediction|         probability|prediction|
+---------------+------------+--------------------+--------------------+----------+
| [25.0,7.0,1.0]|           0|[-21.859997562801...|[0.74436308695773...|       0.0|
| [38.0,9.0,1.0]|           0|[-28.221053569478...|[0.75123890494562...|       0.0|
|[28.0,12.0,1.0]|           1|[-30.538163482660...|[0.74358294376085...|       0.0|
+---------------+------------+--------------------+--------------------+----------+
only showing top 3 rows
```

```
y_pred=pred.select("prediction").collect()
y_orig=pred.select("income_label").collect()

cm = confusion_matrix(y_orig, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[12430     0]
 [ 3846     0]]
```

```
evaluator=MulticlassClassificationEvaluator().setLabelCol("income_label").setPredictionCol("prediction").setMetricName("accuracy"
acc=evaluator.evaluate(pred)
print(acc)
```

```
0.763701155074957
```

Model 2 :

Features : ['age','fnlwgt','education_num','sex_label','capital_gain','capital_loss','hours_per_week']

```
features=['age','fnlwgt','education_num','sex_label','capital_gain','capital_loss','hours_per_week']

va = VectorAssembler(inputCols = features, outputCol='features')
va_df = va.transform(data)
va_df = va_df.select(['features', 'income_label'])
va_df.show(3)
```

```
+--------------------+------------+
|            features|income_label|
+--------------------+------------+
|[39.0,77516.0,13....|           0|
|[50.0,83311.0,13....|           0|
|[38.0,215646.0,9....|           0|
+--------------------+------------+
only showing top 3 rows
```

```
test_va = VectorAssembler(inputCols = features, outputCol='features')
test_va_df = test_va.transform(test_data)
test_va_df = test_va_df.select(['features', 'income_label'])
test_va_df.show(3)
```

```
+--------------------+------------+
|            features|income_label|
+--------------------+------------+
|[25.0,226802.0,7....|           0|
|[38.0,89814.0,9.0...|           0|
|[28.0,336951.0,12...|           1|
+--------------------+------------+
only showing top 3 rows
```

```
nb = NaiveBayes(featuresCol="features", labelCol="income_label",smoothing=1.0, modelType="multinomial")
nb = nb.fit(va_df)
```

```
pred = nb.transform(test_va_df)
pred.show(3)
```

```
+--------------------+------------+--------------------+----------+----------+
|            features|income_label|       rawPrediction|probability|prediction|
+--------------------+------------+--------------------+----------+----------+
|[25.0,226802.0,7....|           0|[-978.76663145419...|  [1.0,0.0]|       0.0|
|[38.0,89814.0,9.0...|           0|[-987.68535754086...|  [1.0,0.0]|       0.0|
|[28.0,336951.0,12...|           1|[-1220.3927487898...|  [1.0,0.0]|       0.0|
+--------------------+------------+--------------------+----------+----------+
only showing top 3 rows
```

```
y_pred=pred.select("prediction").collect()
y_orig=pred.select("income_label").collect()

cm = confusion_matrix(y_orig, y_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[11888   542]
 [ 2957   889]]
```

```
valuator=MulticlassClassificationEvaluator().setLabelCol("income_label").setPredictionCol("prediction").setMetricName("accuracy")
cc=evaluator.evaluate(pred)
rint(acc)
```

```
0.7850208896534775
```

```
sc.stop()
```

Decision tree performed slightly better than Naive Bayes model